РОССИЙСКАЯ АКАДЕМИЯ НАУК

Институт проблем управления им. В.А. Трапезникова

На правах рукописи

Выхованец Валерий Святославович

СИНТЕЗ ЭФФЕКТИВНЫХ МАТЕМАТИЧЕСКИХ МОДЕЛЕЙ ДИСКРЕТНОЙ ОБРАБОТКИ ДАННЫХ НА ОСНОВЕ АЛГЕБРАИЧЕСКОЙ И ПОНЯТИЙНОЙ ДЕКОМПОЗИЦИИ ПРЕДМЕТНОЙ ОБЛАСТИ

Специальность 05.13.11

Математическое и программное обеспечение

вычислительных машин, комплексов и компьютерных сетей

Специальность 05.13.15

Вычислительные машины и системы

Диссертация

на соискание ученой степени

доктора технических наук

Москва 2007

Содержание

Введение

Диссертация содержит изложение основных результатов, полученных автором при исследовании проблемы синтеза эффективных математических моделей дискретной обработки данных на двух уровнях: формально-логическом и концептуально-онтологическом.

Дискретная обработка – самый распространенный метод вычислений, лежащий в основе современных вычислительных средств. На аппаратурном уровне дискретная обработка реализуется устройствами, состоящими из блоков, каждый из которых выполняет преобразование входных данных в выходные. Трассировка данных от выходов одних блоков ко входам других осуществляется их соединениями. Каждый такой блок, в свою очередь, может быть представлен как отдельное устройство, состоящее из других, более мелких блоков. В пределе, необходимом для практической реализации устройства, в качестве элементарных блоков используются логические элементы, имеющие физическую природу преобразования входных данных в выходные.

На программно-аппаратном уровне преобразование данных осуществляется в виде смены состояния информационной среды под управлением программы, сама информационная среда рассматривается как совокупность носителей данных, а программа представляется формализованным описанием этого процесса. Элементарными компонентами программ являются команды, выполняемые техническими средствами информационной системы. Более сложные компоненты – подпрограммы, являются элементарными единицами вызова (адресации) и рассматриваются как именованные совокупности команд. Одна или несколько подпрограмм объединяются в модули, представляющие собой единицы загрузки и хранения программ. В свою очередь совокупность модулей образуют следующий уровень иерархии – программные средства, предназначенные для выполнения той или иной задачи по обработке данных. И, наконец, программы средства объединяются в комплексы и служат для решения целого класса задач.

Как в первом, так и во втором случае эффективность обработки данных определяется количеством операций (логических элементов, команд), которые необходимо выполнить. Однако как при высокоуровневом моделировании – в рамках концептуально-онтологического подхода, так и при низкоуровневом моделировании – в рамках формально-логического подхода, не решена проблема, связанная с выбором методологии анализа и технологии декомпозиции предметной области, позволяющих получать формальное описание дискретной обработки данных, обеспечивающее эффективное решение стоящих прикладных задач. Более того, на настоящий момент не существует единой теории, позво-

ляющей выработать критерии и оценить эффективность произвольной дискретной обработки данных не прибегая к сравнению с другими ее реализациями.

Цель предпринятого исследования – решение важной прикладной задачи преобразования высокоуровневого (первичного) описания предметной области в терминах содержательной постановки задачи в ее эффективное низкоуровневое представление, состоящее из последовательности команд (операций) вычислительного средства. Для достижения поставленной цели в диссертации разработана теория и обоснованы методы синтеза математических моделей предметной области, которые предназначены для эффективного решения задач дискретной обработки данных аппаратурными и программно-аппаратными средствами.

Объектом исследования является процесс обработки данных, реализуемый вычислительными средствами дискретного действия, а предметом исследования – математические модели предметной области, полученные на основе понятийной, объектной, структурной и функциональной декомпозиции.

Общей задачей, решаемой в диссертации, является получение эффективных математических моделей дискретной обработки данных на основе точной формальной спецификации предметной области и решаемых в ней прикладных задач. Частными задачами, вытекающими из общей, являются:

– разработка методологии анализа предметной области, позволяющей строить ее эффективные декомпозиционные схемы в виде семантически замкнутых формальных спецификаций;

– создание технологии обработки данных, основанной на отражении декомпозиционных схем предметной области в конструкциях проблемного языка;

– разработка методов описания семантики проблемного языка, обеспечивающих решение заданных прикладных задач путем дискретной обработки данных;

– обоснование методики определения эффективности дискретной обработки данных и получение точных, приближенных и асимптотических оценок эффективности синтезируемых математических моделей;

– развитие общей теории дискретных функций на основе аппарата алгебраической декомпозиции и его использования для синтеза эффективных описаний дискретной обработки данных.

Методика исследования основана на формально-логическом и концептуально-онтологическом моделировании. Концептуально-онтологическое моделирование осуществляется путем формальной спецификации результатов понятийного анализа предметной области, а формально-логическое – при алгебраической декомпозиции дискретных функ-

ций. Как в первом, так и во втором случае ищутся декомпозиционные схемы, позволяющие получать эффективные математические модели дискретной обработки данных..

Основной результат диссертационной работы состоит в теоретической разработке и практическом решении задачи синтеза эффективных математических моделей дискретной обработки данных. Научная новизна полученных результатов определяется тем, что:

– предложена методология понятийного анализа, позволяющая получать синтаксически и семантически замкнутые формальные спецификации предметной области;

– разработана технология контекстной обработки данных, предназначенная для сокращения семантического разрыва между языком моделирования и содержательными представлениями относительно предметной области;

– решена задача описания семантики формальных языков на основе семантической индукции путем определения семантических категорий в процессе описания языка и описанными ранее средствами;

– обоснована методика алгебраического синтеза дискретных функций и найдены точные оценки сложности синтезируемых формул при конечной размерности задачи;

– обобщена теория алгебраической декомпозиции дискретных функций в широком классе образующих алгебр, различающихся требованиями к операциям.

Практическая значимость полученных результатов заключается в разработке контекстной технологии программирования, при использовании которой получаются более эффективные и качественные программы, а также в обосновании методик алгебраического синтеза формул и оценки эффективности математических моделей дискретной обработки данных, и определяется тем, что:

– высокоуровневые формы декомпозиции дискретных систем основаны на трудно формализуемых методологиях структурного и объектно-ориентированного анализа, а соответствующие им технологии разработки программных средств не гарантируют получение качественных и надежных программ;

– не существует практически реализуемых методик эффективной декомпозиции дискретных функций и соответствующих им методов синтеза формального описания дискретной обработки данных и оценки ее эффективности.

Проверка полученных результатов осуществлена путем вычислительного эксперимента, при котором, в частности:

– разработана и исследована система контекстного программирования, позволяющая автоматизировать процесс перехода от высокоуровневой формальной спецификации предметной области к последовательности команд целевой вычислительной платформы, реализующей решение стоящих прикладных задач путем дискретной обработки данных;

– реализованы алгоритмы спектрального и алгебраического синтеза дискретной обработки данных в виде формул, заданных в базисе произвольных бинарных операций;

– подсчитана порождающая способность аналитических конструкций формул и оценена максимальная сложность спектральной и алгебраической декомпозиции дискретных функций.

По результатам исследований опубликовано 50 работ, причем 8 из них, содержащие основные результаты, опубликованы в рецензируемых журналах из перечня ВАК.

Диссертация состоит из введения, 6-ти глав, заключения и 5-ти приложений. Список цитированной литературы содержит 369 источников. Объем диссертации 452 страницы.

Глава 1.
Дискретная обработка данных

В настоящей главе приводится обзор известных результатов в области математического моделирования и дискретной обработки данных, которые тем или иным образом связаны с темой диссертационной работы, оказали на ее влияние или были обобщены (развиты).

Изложение материала осуществляется с учетом двух на первый взгляд исключающих друг друга подходов – формально-логического и концептуально-онтологического. Однако результаты исследования позволяют сделать вывод, что в рамках этих подходов одна и та же задача решается различными средствами: получение математических моделей предметной области для эффективной ее реализации на вычислительных средствах дискретного действия.

1.1. Математические модели

Одним из универсальных подходов при решении различных научных и технических проблем является представление исследуемых объектов (явлений, процессов) в виде математических моделей, в основе которых лежат предельно абстрактные понятия множества, как совокупности различимых элементов, и отношения, связывающего элементы множеств между собой. Декларируется тезис о подобии, утверждающий, что в аспекте решаемой проблемы математическая модель адекватна объекту. Исследование объекта заменяется исследованием модели, а реализация результатов моделирования осуществляется моделирующими устройствами.

Переход от объекта к модели основан на сопоставлении проявленных качественных характеристик объекта – конечным множествам, а количественных – счетным или континуальным. Описание объекта представляется в виде отношений, устанавливающих связи между его закодированными характеристиками.

Для нахождения неизвестных характеристик объекта по известным (найденным, измеренным) его характеристикам отношения рассматриваются как отображения путем выделения (фиксации, специализации) одних множеств, определяющих область значения отображения, по отношению к другим множествам, задающим область определения. Это позволяет находить (определять, вычислять) одни характеристики объекта по совокупности других его характеристик.

Отсюда появление двух типов математических моделей – дискретных и континуальных (непрерывных), требующих различного математического аппарата для своего ис-

следования и различных моделирующих устройств для своей реализации. Многие прикладные задачи ставятся и имеют решения при дискретном кодировании данных. Дискретная модель задается на конечных и счетных множествах, а моделирующими в этом случае выступают устройства дискретного действия.

В связи с этим возникает задача представления отображений большой размерности в виде композиции отображений, имеющих меньшую размерность. Здесь под размерностью понимается количество элементов или мощность области определения отображения. В частном случае, когда отображения обладают свойством однозначности (детерминированности), задача представления отображений сводится к декомпозиции дискретных функций.

При декомпозиции функция представляется как композиция более простых функций. В пределе, необходимом для практической реализации модели, композиция строится из переменных и операций, непосредственно реализуемых моделирующим устройством. Заметим, что в связи с детерминистической природой дискретных моделирующих устройств, отображения вынужденно сводятся к эквивалентной совокупности дискретных функций. Только в этом случае появляется возможность вычислять одни характеристики модели (неизвестные) через другие (известные).

Таким образом, основной проблемой дискретной обработки является декомпозиция дискретной функции, или ее представление в виде математической модели, например в виде формулы, рассматриваемой как последовательность операций над входными данными (переменными).

1.2. Декомпозиция дискретных функций

История декомпозиции дискретных функций связана, в основном, с декомпозицией булевых функций, задаваемых на множествах из двух элементов. Первая работа из этой области – фундаментальная книга Буля [265] (1854), положившая начало теории декомпозиции. После длительного перерыва Жегалкин опубликовал работу [75] (1927), посвященную преобразованиям булевых функций, и заложил основы современных методов ортогональных преобразований. Начало эпохи прикладного логического синтеза связано с именами Шеннона [339, 340] (1938, 1949) и Гаврилова [78, 79] (1945, 1946).

Первые результаты по табличной декомпозиции функции на основе разделения переменных получены Ашенхерстом [262] (1952) и Семоном [338] (1952), а аналитическое описание разделительной декомпозиции дано Поваровым [190] (1954). В развитие результатов Жегалкина Рид [328] (1954) и Маллер [318] (1954) ввели понятие аналитической конструкции и использовали разложение функций в форме, известной в настоящее время

как каноническая форма Рида-Маллера. Множественная или кратная декомпозиция исследована Кертисом [279] (1958), концепция дифференцирования булевых функций предложена Акерсом [259] (1959), а совместная минимизация систем булевых функций при разделительной декомпозиции осуществлена Ротом и Карпом [329, 330] (1960, 1962). Теория параллельного логического вычисления, осуществляемого в модулярной арифметике, разработана Финько [227] (2003).

Изучению ограниченных классов дискретных функций и частных случаев декомпозиции посвящены работы Кузнецова [135] (1958) – по бесповторной декомпозиции, Нечипорука [177] (1958) – по линейным преобразованиям переменных при минимизации, Майтры [313] (1962) – по каскадной декомпозиции. Представление функций решающими диаграммами исследовано Ли [309] (1959) и Акерсом [260] (1978). Спектральные формы использованы Карповским и Москалевым [117] (1970), а метод декомпозиции, основанный на спектральных разложениях, – Лечнером [308] (1971). Многокритериальная оптимизация синтеза дискретных функций на основе параллельной и последовательной декомпозиции исследована Чебурахиным [239] (2004).

Развитие теории декомпозиции связано с работами в области синтеза схем из функциональных элементов. Метод каскадов, основанный на разделительной декомпозиции, разработан Поваровым [191] (1955). Каноническому синтезу на базе табличных представлений посвящена работа Блоха [21] (1961). Декомпозиция в базисе произвольной сложности, осуществляемая путем замены выходных функций, предложена Пархоменко [180] (1964) и развита Горовым [87] (1967). Аппарат теории графов для функциональной декомпозиции в произвольном базисе использован Горбатовым [86] (1967), что послужило толчком для развития самой теории. Методы решения логических уравнений и их применение при декомпозиции разработаны Рудяну [332] (1974) и Закревским [100] (1975).

Исследование сложности представления булевых функций предпринято Шенноном [249] (1949), Яблонским [256] (1959) и Лупановым [155] (1963). Основной их результат: почти все функции реализуются со сложностью, близкой к максимальной. Более того, показано существование декомпозиции, удовлетворяющей найденным асимптотическим оценкам. Однако выдвинуто предположение о том, что нахождение минимальных декомпозиций сопряжено с полным или почти полным перебором возможных решений и осуществимо только для функций небольшой размерности.

Представимость дискретных функций со сложностью, не превышающей некоторую предельную, привело к развитию форм совместного описания. Хотя сложность синтезируемого выражения в этом случае максимальна, появляется возможность описать все функции сразу (одновременно). При этом базис расширяется, и в него включаются много-

значные (многоразрядные) операции. Среди результатов, полученных в этом направлении, следует отметить использование многозначной логики для представления булевых функций Враневичем, Ли и Смитом [356] (1970), совместное описание систем функций арифметическим полиномом – Малюгиным [161] (1982), векторную реализацию функций – Лапкиным [145] (1983).

В области декомпозиций многозначных функций получены аналогичные результаты. Полнота и замкнутость исследована Яблонским [255] (1958), обобщение разделительной декомпозиции на многозначный случай выполнено Вилиузманом и Враневичем [361] (1970), минимизация рассмотрена Су и Ченгом [349] (1972). В свою очередь, метод декомпозиции, основанный на спектральном анализе, предложен Токменом [354] (1980), а полиномиальное разложение – Страздинсом [347] (1983).

Алгебраические представления многозначных функций связаны с именами Поста [323] (1921, расширение булевой алгебры), Веба [363] (1935, алгебра из одной операции), Бернштейна [264] (1924, модулярная арифметика), Яблонского [255] (1958, алгебра литеральных операций), Коха [272] (1960, конечное поле), Тошича [355] (1972, кольцо модулярных операций), Мак-Класки [314] (1979, алгебра интервальных операций). Линейная независимость спектральных функций в ортогональных разложениях исследована Перковским [322] (1992).

Многообразие методов декомпозиции булевых функций является следствием широкого спектра задач, решаемых на основе дискретного моделирования. Для каждого класса задач получены свои методы дискретной декомпозиции. Последнее связано со стремление найти эффективные решения в той или иной области, в то время как при общей постановке задачи получение эффективных математических моделей сопряжено с высокими вычислительными трудностями. Более того, эффективность найденной декомпозиции не поддается сравнительной оценке, так как практически отсутствует возможность получения других решений за приемлемое время. Отсюда, в частности, следует проблема нахождения минимальных декомпозиций, сводимая к перебору всех возможных вариантов решения.

1.3. Логическая обработка данных

Исторически сложилось так, что основные результаты по дискретной обработке данных следует искать в области логической обработки. Термин «логическая обработка данных» фактически эквивалентен дискретной обработке, с тем отличием, что под логической обработкой часто понимают обработку данных, представленных булевыми функциями, т.е. функциями, переменные которых, как и сами функции, принимают значения

на двоичном множестве. Заметим, что дискретная обработка в общей постановке задачи рассматривается как обработка данных, представляемая дискретными функциями, принимающими значения на произвольных конечных множествах элементов, а двоичные множества используются только в частном случае.

Декомпозиция дискретных функций в общем виде оказалась практически нереализуемой в связи с необходимостью перебора большого числа вариантов решения. Для сокращения трудоемкости синтеза используются не общая, а частные декомпозиционные схемы, примерами которых может служить решающая [339] и спектральная [117, 308] декомпозиции, выполняемые в хорошо изученных алгебраических системах – в булевой алгебре, на кольце, в конечном поле.

Условно можно выделить два крайних подхода в реализации логических вычислений. Первый – когда преобразование входных данных в выходные осуществляется на основе нерегулярных представлений, и второй – когда вычисления осуществляются на основе регулярных форм (табл. 1.1).

Таблица 1.1. Декомпозиции дискретных функций

Декомпозиция	Конструкция	Переменные
Пересекающаяся	$f(X) = \theta(X', a(X''))$	$X' \cup X'' = X$ $X' \cap X'' \neq \varnothing$
Разделительная	$f(X) = \theta(X', a(X''))$	$X' \cup X'' = X$ $X' \cap X'' = \varnothing$
Кратная	$f(X) = \theta(X', a_1(X''), a_2(X''), \ldots, a_m(X''))$	$X' \cup X'' = X$ $X' \cap X'' = \varnothing$
Промежуточная	$f(X) = \Sigma(\theta_1(X', a_1(X'')), \ \ldots, \theta_m(X', a_m(X'')))$	$X' \cup X'' = X$ $X' \cap X'' = \varnothing$
Алгебраическая	$f(X) = \sum_{i=1}^{m} \theta_i(X') \times a_i(X'')$	$X' \cup X'' = X$ $X' \cap X'' = \varnothing$
Спектральная	$f(X) = \sum_{i=1}^{m} \theta_i(X') \times a_i(X'') = \sum_{i=1}^{m} \theta_i(X) \times a_i$	$X' = X$ $X'' = \varnothing$

Нерегулярные формы получают в результате декомпозиции дискретной функции общего вида. Так, функция f от n переменных $X = \{x_0, x_1, \ldots, x_{n-1}\}$ на каждом шаге декомпозиции выражается как функция от других функций θ и a, $f(X) = \theta(X', a(X''))$, где X', X'' – подмножества X [338].

Различают однократную и многократную, итеративную и рекурсивную декомпозиции. Декомпозиция завершается, когда получено представление, реализуемое на уровне операционных возможностей вычислительного средства.

Попытки найти декомпозицию функции общего вида (нерегулярную декомпозицию) столкнулись с серьезными трудностями, связанными с предельно общей постановкой задачи [256]. Декомпозицию общего вида принято рассматривать как комбинаторную задачу, связанную с перебором большого числа решений, и при больших размерностях практически нереализуемую [20].

Нерегулярная декомпозиция сводится к NP-полной задаче раскраски графа, для которой не найдено эффективных решений [275]. В связи с этим нерегулярную обработку данных представляют, в основном, в алгоритмических формах, для чего применяются методологии и технологии, основанные на использовании содержательных представлений о предметной области. Теоретической базой при этом служат известные парадигмы анализа предметной области и методологии проектирования (синтеза) программных средств, а именно: логическая, функциональная, структурная и объектно-ориентированная и др.

Поэтому поиск формальных методов дискретной декомпозиции и дискретной минимизации пошел в направлении исследования регулярных аналитических конструкций, предельным случаем которых является спектральные разложения.

Промежуточное положение между нерегулярными и регулярными формами занимают решающие диаграммы [268], являющиеся, по своей сути, графической формой представления логических программ [138]. Этот вид представления основан на разложении дискретной функции в алгебраической системе, образованной двумя операциями, которые условно назовем сложением и умножением.

Решающее разложение осуществляется по функциям θ_i одной переменной X': $f(X) = \sum \theta_i(X') \times a_i(X'')$, где a_i – функции (коэффициенты разложения), зависящие от оставшейся части переменных X'' [168]. На следующем шаге разложению подлежат функции a_i. Процесс повторятся до тех пор, пока в качестве коэффициентов не будут получены константы.

Регулярные формы основаны на крайнем случае разложения – спектральном представлении дискретной функции, когда $X' = X$ и $X'' = \varnothing$,

$$f(X) = \sum_{i=0}^{m-1} \theta_i(X) \times a_i \,, \tag{1.1}$$

где a_i – константы, или коэффициенты разложения в базисе $\{+, \times, \theta_i \ (i = \overline{0, m-1})\}$, θ_i – спектральные функции.

Пересекающаяся [101], разделительная [262, 190] и кратная [279] декомпозиции относятся к классу неалгебраических. Решающая [191] и спектральная [117, 308] – выполняются в некоторой алгебре с двумя бинарными операциями и относятся к классу алгеб-

раических. В свою очередь промежуточная декомпозиция [154] является прообразом алгебраической и расположена между алгебраическими и неалгебраическими видами.

При спектральном представлении тип базиса Ω определяется образующими его алгебраическими операциями $\{+,\times\}$. Для булевых функций используется классический базис Буля [265] с операциями конъюнкции и дизъюнкции, базис Жегалкина [75] с операциями неэквиваленции и конъюнкции, арифметический базис [167, 161].

В k-значной логике – одном из расширений традиционной логики – применяются базисы с операциями максимума и минимума [255], сложением по модулю k и минимума [281], арифметическим сложением и поразрядными логическими операциями [71], арифметическими операциями на кольце целых чисел [355], операциями полей Галуа [272, 324].

Просматривается определенная методологическая связь между логической обработкой данных и цифровой обработкой сигналов [143, с. 45]. Отличие цифровой обработки сигналов от логической обработки данных заключается в различных постановках задачи (табл. 1.2). В последнем случае основной задачей является минимизация дискретной функции с целью ее эффективного вычисления (реализации), в то время как при цифровой обработке сигналов основная задача – получение спектра сигнала для его эффективной обработки в спектральной области.

Таблица 1.2. Соответствие терминов и процедур

Цифровая обработка сигналов	Логическая обработка данных
Дискретный сигнал	Дискретная функция
Многомерный сигнал	Система дискретных функций
Вектор отсчетов сигнала	Характеристический вектор функции
Спектр сигнала	Вектор коэффициентов формы
Выбор базиса	Синтез аналитической конструкции формы
Оптимизация базиса	Минимизация формы
Сжатие сигнала	Минимизация функции
Генерация сигналов	Синтез форм
Представление сигнала спектром	Вычисление вектора коэффициентов
Восстановление сигнала по спектру	Кратные вычисления формы
Взаимное преобразование спектров	Взаимное преобразование форм
Обработка сигнала	Операции над формами

Основные достижения в области цифровой обработки по оптимальному представлению сигналов связывают с преобразованием Карунена-Лоэва [8, с. 182], в котором раз-

ложение функции осуществляется по собственным векторам ее ковариационной матрицы. Тем самым обеспечивается наилучшее (оптимальное) приближение функции в среднеквадратическом смысле.

Хотя при разложении по Карунену-Лоэву находится оптимальный спектр заданной функции, имеется проблема эффективности вычисления спектральных представлений. Последнее связано с тем, что вместо исходной функции приходится вычислять множество функций, имеющих ту же размерность, что и исходная. По этой причине необходимо потребовать, чтобы сложность вычисления функции по ее спектральному представлению была меньше, чем сложность вычисления той же функции при использовании других ее форм.

Однако критерии, позволяющие оценить эффективность имеющегося представления, не прибегая к синтезу функции в других формах, не разработаны. Это приводит к перебору множества вариантов декомпозиции функции, что, в конечном итоге, практически не реализуемо, так как связано непреодолимыми трудностями вычислительного характера. В итоге, задача определения абсолютной эффективности математических моделей дискретной обработки данных является на настоящий момент актуальной.

1.4. Дискретные преобразователи

Данные при дискретной обработке представляются в виде конечных последовательностей знаков (цифр), взятых из некоторого конечного алфавита N, а сама обработка осуществляется путем преобразования входных данных X в выходные Y посредством разделения X на части (переменные) x_0, …, x_{n-1} и выполнения над ними некоторых последовательностей операций $F = \{f_0,\ldots,f_{s-1}\}$ для получения результатов y_0, …, y_{n-1}, из которых потом формируются выходные данные Y (рис. 1.1).

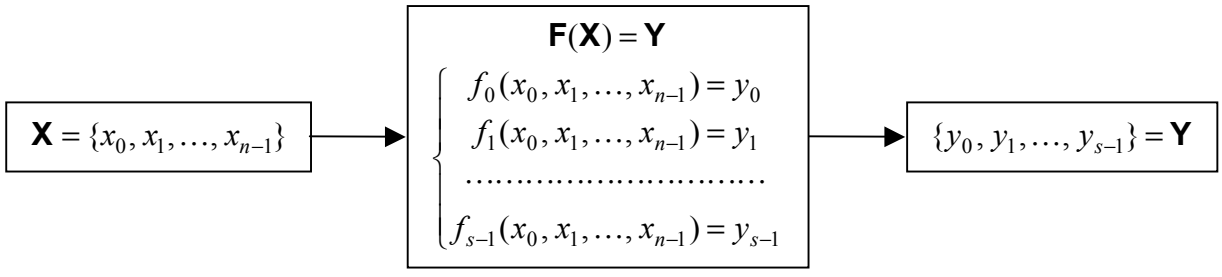

Рис. 1.1. Дискретный преобразователь

1.4.1. Дискретные модели без памяти

Традиционно первой формой получения компактного представления дискретной модели стала индивидуальная минимизация функций, которая заключается в поиске минимальных или близких к минимальным форм для каждой функции системы в отдельно-

сти. Очевидно, при этом общее количество формул остается неизменным; представления функций упрощаются, но приходится жертвовать единим базисом, так как каждая функция имеет свой – в некотором смысле оптимальный – набор операций для своего наиболее простого выражения.

Другим методом получения и упрощения дискретной модели является совместная минимизация функций системы, при которой выделяются некоторые общие подвыражения, участвующие при вычислении двух и более функций. В итоге, количество функций системы увеличивается, но уменьшается требуемое количество логических элементов (при аппаратурной реализации) и общее время вычисления модели (при программной).

Как для индивидуально, так и для совместной минимизации дискретных функций не найдено эффективных алгоритмов, позволяющих за приемлемое время выполнить минимизацию достаточно сложной модели. Как правило, используются переборные алгоритмы, имеющие высокую трудоемкость. Так как индивидуальная и совместная минимизация функций сохраняет их общее количество неизменным или даже увеличивает, то для повышения эффективности модели ставится задача поиска форм совместного описания функций, например, в виде единого выражения, позволяющего вычислять все функции сразу, не вычисляя каждую в отдельности.

Известно, что произвольная функция может быть представлена со сложностью, не превышающей некоторую максимальную, причем эта максимальная сложность не зависит от используемого базиса. Поэтому целесообразно решать задачу поиска конечных форм совместного описания всей системы функций, например, в виде единого выражения в некотором общем для всех функций базисе. Сложность этого выражения может быть максимальной, зато будет описываться вся система сразу, а функции системы будут вычисляться одновременно (параллельно). Это позволяет не только уменьшить объем памяти, необходимый для хранения модели, но и сократить время ее вычисления.

Одной из первых и самой распространенной формой совместного описания системы дискретных функций является арифметический полином [161]. Предельным случаем упрощения дискретной модели является ее представление в виде линейного выражения – суммы переменных, умноженных на некоторые коэффициенты, причем операции сложения и умножения не обязательно должны быть традиционными арифметическими. Доказано [162], что за счет увеличения разрядности коэффициентов произвольная система булевых функций может быть представлена в виде композиции двух линейных арифметических полиномов ограниченной сложности. Если даже выражение нелинейное, то специальное увеличение разрядности коэффициентов позволяет реализовать кратные вычисле-

ния, при которых система функций вычисляется сразу (одновременно, параллельно) на нескольких наборах переменных [71].

Все перечисленные выше методы основаны на спектральной декомпозиции. При спектральной декомпозиции функция разлагается в линейную комбинацию фиксированного множества других функций, называемых спектральными и имеющими такую же длину характеристических векторов, что и исходная. С технической точки зрения такой подход оправдан, когда эффективность реализации спектрального базиса достаточно высока. Однородность спектральных разложений не позволяет гарантировать эффективное формульное представление наперед неизвестной функции. Поэтому развитие спектральных методов связано с уменьшением регулярности синтезируемых формул, в частности: применением многоступенчатой декомпозиции (решающих диаграмм) [309, 260], объединением спектральных составляющих в различных арифметиках [264, 272, 167, 355], совместным описанием множества функций в виде одного спектра [161], использованием новых и расширения имеющихся спектральных базисов [143, 291].

Интерес к спектральным методам представления дискретных систем и распараллеливанию вычислений вызван появлением особо сложных задач, в которых многочисленные данные связаны логической зависимостью. При этом реализация дискретных алгоритмов обычно мыслится как программная. Принятое ныне описание схем и алгоритмов, основанное на языке булевой алгебры, лишь частично соответствует программным и техническим возможностям современных ЭВМ. Заметим, что в англоязычных публикациях структурированное объединение логических выражений называется представлением функций в форме слов (word-level).

Так как в основе описанных подходов лежат способы совместного описания системы дискретных функций посредством какой-нибудь единой формулы, то появляется надежда, что сведение системы функций к одному выражению упрощает и процедуру проверки устройства, реализующего эту формулу. Иначе говоря, упрощается тестирование, диагностирование устройств и верификация соответствующих программ.

1.4.2. Дискретные модели с памятью

Задачи дискретной обработки данных возникают при использовании дискретных систем управления, которые используют дискретные сигналы и функционируют в дискретном времени (рис. 1.2).

В отличие от других областей, постановка задачи дискретной обработки данных в управлении базируется на автоматных моделях с памятью: конечном автомате, магазинном автомате, сети Петри, машине Тьюринга и др.

До сих пор нами рассматривались дискретные модели без памяти, соответствующие в теории автоматов самому их простому виду – комбинационному автомату. Комбинационный автомат описывается дискретной функцией $Y = f(X)$, задающей для каждого состояния входа X состояние его выхода Y, то есть число его внутренних состояний равно одному.

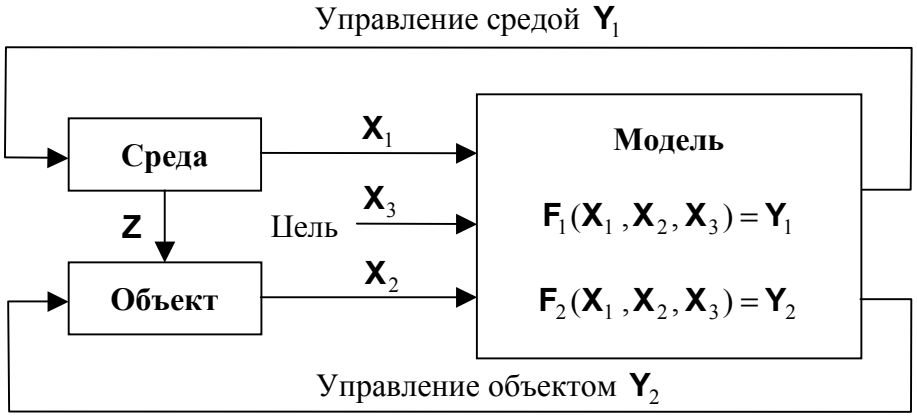

Рис. 1.2. Дискретная обработка данных в управлении

В отличие от комбинационного, конечный автомат имеет конечное число внутренних состояний $\{q_0, q_1, ..., q_{n-1}\}$ и описывается двумя дискретными функциями: функцией выходов $Y = f(X, Q)$ и функцией переходов $Q^+ = g(X, Q)$, где Q и Q^+ – соответственно текущее и следующее состояния автомата (рис. 1.3). Внутренняя память автомата представляется повторителем $Q = Q^+$ (элементом задержки на один такт).

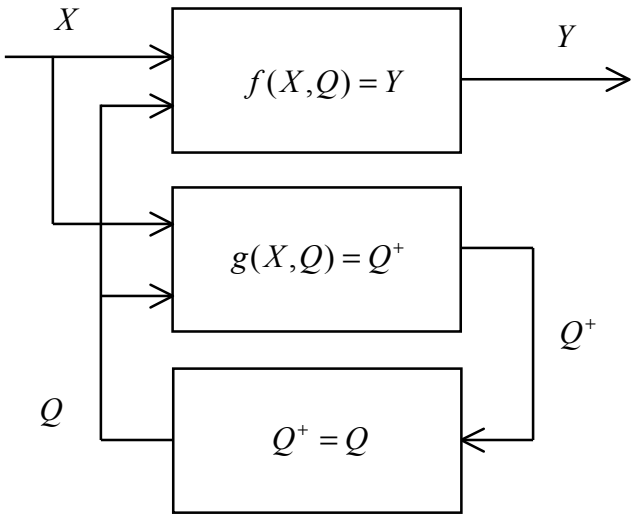

Рис. 1.3. Конечно-автоматная модель

Более сложные автоматы представляются как состоящие из конечного автомата и запоминающего устройства бесконечного объема, но с различными дисциплинами досту-

па: магазинный автомат строится на запоминающем устройстве со стековой организацией, машина Тьюринга – на памяти с последовательным доступом и т.п.

Несмотря на кажущуюся сложность автоматных моделей, в основе их описания лежит все то же аппарат дискретных функций, но интерпретируемый во времени – описание функционирования автомата представляется в виде распределенных во времени элементарных действий, каждое из которых описывается дискретной функцией.

Таким образом, система управления дискретного действия может быть описана совокупностью дискретных функций (системой функций). В результате представления этих функций в одной из форм их описания, например, в виде выражений в некотором функционально полном базисе операций, получают дискретную модель объекта (среды), состоящую из отдельных описаний каждой функции.

1.4.3. Алгоритмические модели

Накоплен большой опыт в обработке данных с использованием информационных систем. В этом случае преобразование данных осуществляется в виде процесса смены состояния информационной среды, сама информационная среда рассматривается как совокупность носителей данных, а программа представляется формализованным описанием этого процесса (рис. 1.4). При создании программ используется аппарат формализации, основанный на записи алгоритмов на универсальных языках программирования.

Рис. 1.4. Информационная система

Становление теории алгоритмов начинается с доказательства К. Гёделем теорем о неполноте формальных систем, включающих арифметику [166]. Возникшее в связи с этими теоремами предположение о невозможности алгоритмического разрешения многих математических проблем (в частности, проблемы выводимости в исчислении предикатов) вызвало необходимость уточнения понятия алгоритма.

Строгое определение этого понятия было дано в работах А. Тьюринга [250], А. Чёрча [242] и Э. Поста [224]. Предложенные ими машина Тьюринга, формализм Поста и лямбда-исчисление Чёрча оказались эквивалентными друг другу. Основываясь на работах А. Гёделя, С. Клини ввел понятие рекурсивной функции, также оказавшееся эквивалентным вышеперечисленным [125]. Одним из наиболее удачных вариантов определения алгоритма является введённое А. А. Марковым понятие нормального алгорифма [164]. Оно было разработано позже в связи с доказательством алгоритмической неразрешимости ряда алгебраических проблем.

В основе формализации понятия алгоритма лежит утверждение, известное как тезис Чёрча-Тьюринга, и заключающееся в том, что любой алгоритм в интуитивном смысле может быть представлен эквивалентной математической моделью, например машиной Тьюринга.

На рис. 1.5 показаны те алгоритмические модели, которые используются в теории алгоритмов для теоретического исследования процесса обработки данных. Как видно из рисунка, все модели представляют собой конечный автомат, дополненный различными типами запоминающих устройств, отличающихся дисциплинами доступа к данным и объемом предоставляемой памяти.

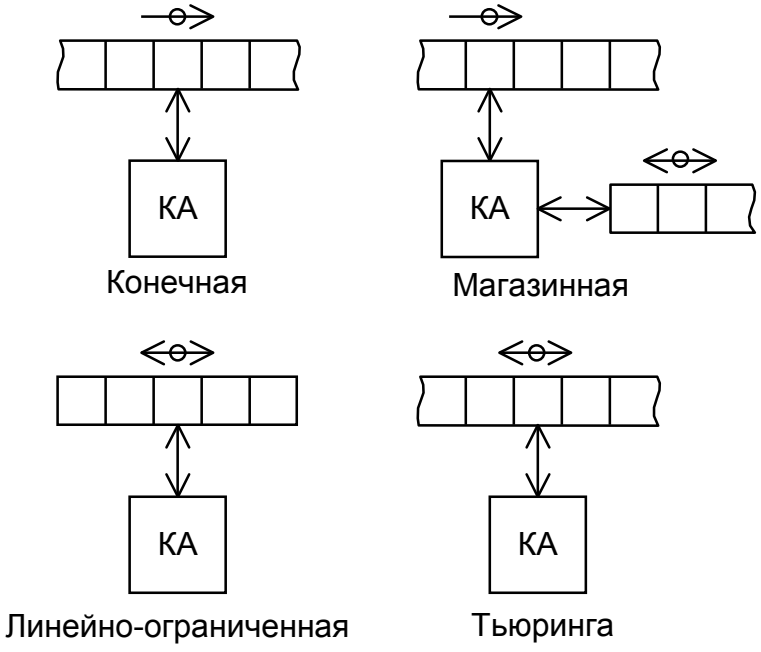

Рис. 1.5. Вычислительные модели

Так, конечно-автоматная модель имеет входной и выходной поток данных с доступом только к текущей ячейке (показано в виде ленты, которая перемещается вправо на одну ячейку за каждый такт). Более мощной является магазинная (стековая) модель, которая дополнена магазинной памятью с организацией «последний вошел, первый вышел» и

доступом к последней ячейке. Следующей моделью по мощности является линейно-ограниченная модель, где реализован произвольный доступ к ячейкам памяти, но количество таких ячеек конечно. Очевидно, самой мощной является машина Тьюринга, имеющая бесконечную память и произвольный доступ к ее ячейкам.

Во всех перечисленных моделях программой является описание работы конечного автомата, однако линейно-ограниченная модель и машина Тьюринга уже в состоянии хранить программу в запоминающем устройстве.

Основной результат общей теории алгоритмов – наличие неразрешимых массовых проблем, являющихся следствием предельно общей постановки задачи [139]. Под массовой проблемой понимается нахождение единого метода решения некоторого класса задач, условия которых могут варьироваться в известных пределах. Уточнение представления о вычислимости на основе формализации понятия алгоритма (машина Тьюринга и другие эквивалентные ей формы) открыло возможности для строгого доказательства алгоритмической неразрешимости различных массовых проблем.

Таким образом, существуют классы корректно поставленных задач (массовых проблем), для которых, тем не менее, доказано отсутствие каких-либо алгоритмов их решения. Алгоритмическая неразрешимость массовой проблемы не означает неразрешимости той или иной единичной задачи данного класса. Та или иная конкретная задача может иметь решение, причем даже вполне очевидное, а для другой задачи может существовать простое доказательство отсутствия решения. Но в целом, данный класс задач не имеет общего универсального алгоритма решения, применимого ко всем задачам. Для решения отдельных подклассов задач нужно разрабатывать свои алгоритмы; для некоторых отдельных задач требуется разработка уникальных методов решения.

Алгоритмически неразрешимыми является, например, проблема остановки – по описанию алгоритма в виде системы команд машины Тьюринга распознать с помощью другой машины Тьюринга, остановится или нет анализируемая машина. Другими неразрешимыми проблемами являются проблемы эквивалентности алгоритмов, тождества двух математических выражений, распознавания того, можно ли или нет из имеющихся автоматов собрать заданный автомат, а также множество проблем, относящихся к топологии, теории групп и другим прикладным областям.

Конкретизация массовой проблемы позволяет найти решение той или иной задачи, но каждый раз по-своему. Например, в общей постановке задача распознавания применимости алгоритма неразрешима, однако, если рассматривать конкретный алгоритм, то решение может быть найдено, и это решение будет вытекать из специфики алгоритма и следовать из его описания. Заметим, что с таким явлением мы столкнулись при поиске де-

композиции дискретной функции, когда в общем виде нахождение оптимальной декомпозиции сопряжено с практически непреодолимыми трудностями, в то время как для конкретной функции может быть найдена ее минимальная форма.

Трудности, связанные с использованием общих методик и универсальных декомпозиционных схем для синтеза формального описания дискретной обработки данных, в значительной мере объясняются непредсказуемостью поведения дискретных объектов, что, в основном, связано с отсутствием ограничений на степени свободы такого рода объектов.

Если предположить, что в процессе изучения предметной области формируются декомпозиционные схемы, приспособленные для эффективного решения прикладных задач, то их отражение в формальном аппарате, используемом для синтеза математических моделей дискретной обработки данных позволит получить эффективные решения. В итоге получаем, что основное направление поиска эффективных математических моделей дискретной обработки данных следует осуществлять не в общей постановке, а с учетом некоторых содержательных представлений, вытекающих из специфики решаемой задачи.

В связи с тем, что общая теория дискретной обработки данных абстрагируется от содержательных представлений о предметной области и решаемых в ней прикладных задачах, нахождение частного вида декомпозиционных схем там несостоятелен. Очевидно, поиск путей решения задач большой размерности следует искать в области, где сохранены эти содержательные представления. Одной из таких областей, где не только накоплен большой практический опыт в решении такого рода задач, но и имеются существенные теоретические результаты, является область программирования – создание формализованных описаний процессов изменения состояний информационных сред.

1.5. Программная инженерия

Попытки использовать различные математические подходы для конструирования программ предпринимались с момента появления программирования. Относительный успех был достигнут в теории компиляторов, реляционных баз данных, функциональном программировании и в нескольких других узкоспециальных областях. Однако серьезных результатов в большинстве практических областей достичь не удалось [219].

В практическом программировании часто не хватает понимания важности теоретических моделей и исследований в области разработки языков и технологий программирования. Степень абстрагирования от реальной задачи, которая необходима на первой стадии построения модели, довольно часто приводит к такому упрощению этой задачи, что теряется весь ее смысл [200].

Современный этап программирования характеризуется тенденцией смешанного использования разных парадигм. Анализ существующих технологий и использование их для реализации информационных систем, а также сравнение тенденций развития языков программирования, позволяют сделать вывод, что наиболее быстрыми темпами в разработку систем, основанных на знаниях, будет входит объектно-ориентированная парадигма и, как следствие, языки типа Smalltalk [302]. Однако следование объектно-ориентированной традиции уже начинает сдерживать развитие информационных систем. Последнее связано с ограниченностью объектного подхода для представления и обработки знаний и объясняется не объектной природой этих знаний.

Методы программной инженерии условно можно разделить на две большие группы: прямые и обратные [228]. Прямые методы необходимы для создания и развития программных средств, а обратные – для их анализа, в частности, для определения показателей качества. Здесь под программным средством понимается совокупность программы, представленной на носителе данных, и документации, специфицирующей требования к этой программе. Программа, в свою очередь, рассматривается как описание процесса обработки данных, выполненное на некотором формальном языке.

1.5.1. Прямые методы инженерии

В прямых методах в качестве идеала рассматривалась следующая схема:

– на языке формальных спецификаций описываются функциональные требования к программному средству;

– путем аналитического исследования устанавливается корректность спецификации – спецификация верифицируется;

– при помощи некоторого инструмента на основе формальных спецификаций генерируется программная реализации.

Несколько более реалистичный сценарий дополняет описанную выше схему процессом постепенного уточнения спецификаций. Каждый шаг уточнения проводится человеком, который направляет процесс уточнения. При этом соответствующие инструменты следят за тем, чтобы очередное уточнение спецификации не пришло в противоречие с исходными.

В обоих сценариях в качестве итогового результата должна появиться программная реализация, удовлетворяющая всем специфицированным требованиям и не содержащая ошибок.

Для решения прямой задачи используются языки формальных спецификаций, которые, с одной стороны, имеют много общего с языками программирования, а с другой

стороны, предоставляют специальные средства, сближающие их с математической нотацией и облегчающие рассуждения о свойствах текстов на формальных языках.

Однако большая часть исследований по прямым формальным методам оказалась малопригодной на практике, за единственным исключением – работы по конечным автоматам, которые нашли самое широкое применение в проектировании и тестировании средств автоматики, связи и вычислительной техники.

Опыт использования конечных автоматов в разработке аппаратурных средств переносился на разработку программ, хотя в существенно меньших масштабах [244]. Вместе с тем, на отдельных направлениях формальные методы, в частности, языки формальных спецификаций, достигли значимых успехов. Эти успехи, с одной стороны, были обусловлены удачным сочетанием представлений относительно предметной области и возможностей формальных методов (языки спецификации телекоммуникационных протоколов SDL, LOTOS) [285], и, с другой стороны, приближением языков спецификации к формам, привычным в традиционном подходе к программированию (Венский метод – Vienna Development Method, и его развитие – язык RAISE) [185, 346]. Из результатов советской школы наибольшую известность получили работы А.А. Маркова [164] и работы А.А. Ляпунова и его учеников [257].

1.5.2. Обратные методы инженерии

Основная проблема обратных методов состоит в том, что на сегодняшний момент результатом изучения программ является знание отдельного индивида [228]. Это знание не отчуждается от индивида и легко им теряется.

Обратные методы в основном развиваются в направлении автоматизации верификации и тестирования программных средств на основе использования формальных спецификаций, но не предметной области, а самого процесса тестирования [4, 286].

Поэтому применение формальных методов в верификации и тестировании программных средств сталкивается с серьезной трудностью извлечения требований из имеющихся описаний программного средства (документация, тексты программ) и их формализации. Этот процесс, по своей сути, является построением некоторой предварительной модели, необходимой для анализа свойств программного средства, выявления противоречий в требованиях, уточнения характеристик. И пока эти свойства, требования и характеристики не будут явным образом определены, задача получения формальных спецификаций программного средства не может быть автоматизирована.

В настоящее время попытки решения проблемы обеспечения качества программных средств выполняются путем стандартизации процессов разработки:

– вводятся единообразные требования к этапам работ, документации;

– организовываются регулярные совещания;

– проводится инспекция кода и прочее.

Одним из основных результатов в этом направлении стало введение понятия жизненного цикла программной системы, четко определявшее необходимость рассмотрения множества задач, без решения которых нельзя рассчитывать на успех программного проекта [150], а основным методом обеспечения качества стала сертификация программных продуктов и организаций, их выпускающих [130].

1.5.3. Методологии и технологии программирования

В программной инженерии наблюдается некоторое несоответствие терминологии общепринятой. То, что обычно называется способом, здесь называется методом; то, что называется методом или методикой, называется методологией; а то, что называется методологией, возводится в ранг парадигмы [174].

Программирование как практическая деятельность основывается на стилях, где под ***стилем*** понимается внутренне согласованная совокупность базовых конструкций программ и способов их композиции, обладающая общими фундаментальными особенностями. Стиль программирования реализуется через ***методологию***, заключающуюся в совокупности соглашений о том, какие базовые концепции языков программирования и какие их сочетания считаются приемлемыми для данного стиля. В свою очередь методология реализуется через ***методики***, которые выражаются в требованиях и рекомендациях по созданию программ, а для производства программ используются ***технологии*** программирования, определяемые как совокупность инструментальных и организационных средств, поддерживающих некоторую методологию.

В практике программирования сложились различные подходы, вызванные совершенствованием и распространением новых вычислительных архитектур и математических теорий, описывающих соответствующий формальный аппарат.

В программной инженерии для декомпозиции предметной области используются методологии функционального, структурного и объектного анализа, лежащие в основе одноименных технологий программирования (табл. 1.3).

Таблица 1.3. Методологии и технологии программирования

Методология	Декомпозиция	Технология
Функциональная	Функциональная (задача-подзадача-формула)	Функциональное программирование
Структурная	Структурная (целое-часть-связи)	Процедурное (модульное) программирование
Объектная	Объектная (объект-состояние-взаимодействие)	Объектно-ориентированное программирование

Функциональная декомпозиция. Функциональная методология была исторически первой методологией анализа, которая позволила с единых позиций подойти к задаче структурно-функциональной декомпозиции предметной области (рис. 1.6).

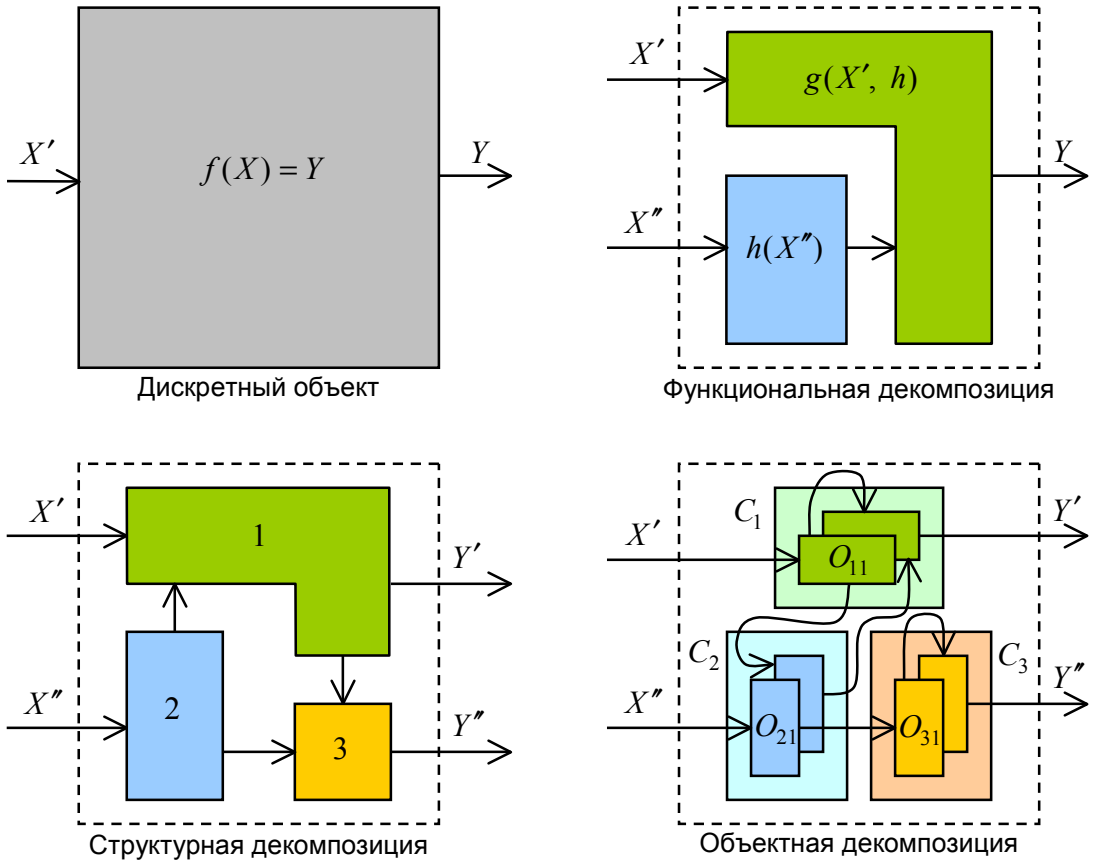

Рис. 1.6. Декомпозиции дискретных объектов

Функциональная декомпозиция определяется как разбиение задачи на более простые функциональные единицы, где каждая единица выполняет завершенную, легко идентифицируемую подзадачу [226]. При функциональном подходе предметная область представляется в виде задачи, которая разбивается на подзадачи, объединяемые в виде формулы, выражающей решение общей задачи путем решения частных подзадач. В свою очередь

частные подзадачи далее декомпозируются на более мелкие, и так до достижения нужной степени подробности. Декомпозиция завершается, когда получены подзадачи, непосредственно решаемые моделирующим устройством.

В итоге, при функциональном подходе, сложные задачи решаются посредством композиции функций. При этом текст программы представляет собой функцию, аргументы которой также рассматриваются как функции, возможно тривиальные.

Для формализации понятия «функция» была построена математическая теория, известная под названием лямбда-исчисления [14] и основанная на преобразовании объектов исчисления из одной формы в другую. Близость функционального подхода к математической формализации позволяет упростить тестирование и верификацию программ, вплоть до доказательства их правильности. Функциональному подходу соответствуют интуиционистская логика предикатов (типизированный вариант) [174] и комбинаторная логика (не типизированный вариант) [47].

Однако функциональный подход не лишен недостатков. Если при анализе и проектировании используется функциональная декомпозиция, то структура программы будет вытекать из исходного понимания разработчиками главной функции. При этом добавление всякой новой функции, даже если оно кажется простым, разрушает структуру программы и приводит к ее повторному проектированию. Развитие функционального подхода сдерживается рядом проблем, среди которых проблемы совместимости, мобильности, сложности обучения и т.п. [360].

Структурная декомпозиция. Структурная методология (рис. 1.6) предоставила в распоряжение разработчиков методы структурной декомпозиции предметной области [163, 241, 114, 129]. Основу структурного анализа составляет структурная декомпозиция предметной области, при которой последняя разделяется на части, между которыми устанавливаются связи. Наглядность и строгость структурного анализа позволяет разработчикам и пользователям системы формализовать представления, необходимые для адекватной реализации информационной системы.

В структурном анализе и проектировании используются различные подходы, структурирующие предметную область и моделирующие эту область в виде информационной системы, описываемой функциональной структурой (модель SADT) [163], последовательностью выполняемых действий (модель IDEF3) [241], передачей данных между структурыми блоками (диаграммы потоков данных DFD) [114], отношениями между данными (модель «сущность-связь», или ERM) [129], и др.

Однако широкое применение структурного анализа и следование его рекомендациям связано с трудностями создания детальных формальных спецификаций информацион-

ной системы, проверки их на полноту и непротиворечивость. Еще большие трудности ожидают разработчиков при попытке изменения этих спецификаций.

Объектная декомпозиция. На смену структурному анализу в виде естественного его развития пришла методология объектно-ориентированного анализа и одноименная технология создания программных средств. В основе объектно-ориентированного анализа лежит создание объектной модели предметной области (рис. 1.6). Ее основные принципы (децентрализация, контракты, самодостаточность, классификация, бесшовность) [315] реализуются методами абстрагирования, инкапсуляции и полиморфизма при структурной декомпозиции предметной области [32]. При этом используются такие понятия, как объект, класс, атрибут, метод, интерфейс, и др., следующие из объектной декомпозиции предметной области, выполняемой в терминах «объект», «состояние» и «взаимодействие».

Спецификация результатов объектно-ориентированного анализа осуществляется на языке моделирования UML (Unified Modeling Language) [203], который содержит средства для определения, представления, проектирования и документирования результатов объектого, структурного и функционального анализа предметной области.

Язык поддерживает стандартный набор декомпозиционных схем, реализуемых в виде всевозможных диаграмм и нотаций. В UML отражена как методология структурного анализа в виде компонентов и их размещения, так и объектно-ориентированного, представленная в виде диаграмм классов и различных моделей поведения объектов: вариантов использования, взаимодействия, состояния и деятельности.

Объектная модель задается в виде диаграмм классов и показывает статическую структуру проблемной области. Сначала определяются классы объектов, затем зависимости между объектами. Динамическая модель показывает поведение объектов и задается последовательностями их взаимодействия. Сначала готовятся сценарии типичных сеансов взаимодействия, затем определяются внешние события, отражающие взаимодействие с внешним миром; после этого строится диаграмма состояний для каждого объекта, на которой показываются образцы событий, получаемых системой и порождаемых ею, а также действий, выполняемых объектами.

Отличительным свойством UML от других средств моделирования (IDEF3, DFD, ERM) является возможность расширения языка, основанная на таких сущностях, как стереотипы, теги и ограничения.

Главным достоинством объектно-ориентированного подхода является то, что создаваемые объектно-ориентированные модели более открыты и легче поддаются внесению изменений, поскольку их конструкция базируется на устойчивых формах отражения ре-

зультатов декомпозиции. Это дает возможность модели развиваться постепенно и не приводит к полной ее переработке даже в случае существенных изменений требований к проектируемой информационной системе.

Однако достижения в области объектно-ориентированного подхода лишь обещают разрешение трудностей, а на самом деле реализуют противоречие, заключающееся в том, что «объекты» являются содержанием, не имеющим ясной формы и говорить о «классах», как о строительных блоках для создания конкретных прикладных моделей, не приходится: они суть лишь строительные блоки программных продуктов [91].

Дело в том, что методы, положенные в основу DFD , STD , SADT/IDEF0, IDEF3, а также UML, методами, по сути дела, не являются. Это только нотации, нормативные системы, языки. А системы правил, используемые в рамках перечисленных нотаций, никак не исходят из знания закономерностей исследуемого предмета и никак не помогают выбрать существенное [170].

Общий недостаток всех методологий программирования – практическое игнорирование того, что программу не опишешь понятиями, которые заданы внутри ее самой. Известно, что для полного описания свойств программы требуются понятия, в программе явно не выражаемые, но необходимые для корректного описания системы в целом. Например, характерным примером таких понятий являются «ординалы» и «число шагов цикла общего вида» [174].

1.5.4. Семантический разрыв

Накопленный опыт проектирования информационных систем показывает сложность и трудоемкость этого процесса, длительного во времени и требующего высокой квалификации участвующих в нем специалистов.

Какая бы методология ни использовалась, в любом случае предметная область представляется в виде иерархии моделей различных видов. Модель вышележащего уровня строится как композиция более простых моделей, составляющих нижний уровень. В итоге осуществляется иерархическая декомпозиция предметной области в соответствии с накопленными о ней знаниями. После построения и верификации модели верхнего уровня выполняется ее описание на нижележащем уровне, вплоть до «принципиального», реализуемого в виде программного средства для некоторой аппаратной платформы.

При необходимости осуществления доработки модели, связанной с изменением представлений о предметной области или решаемых задач, происходит перестройка всей модельной иерархии, вплоть до программной. Тут возникает проблема преобразования изменений в одной форме описания предметной области в изменения других форм, являющихся порождением исходной и выполненных с другой степенью детализации.

Недостатком известных методологий и технологий декомпозиции предметной области, затрудняющим решение этой проблемы, остается сохранение семантического (смыслового) разрыва между моделями различных уровней.[1] Например, между высокоуровневой моделью предметной области и языком программирования, с помощью которого эта модель реализуется на этом языке; или между высокоуровневой моделью (спецификацией) и ее описанием на естественном языке, отражающем содержательные представления пользователей о предметной области, и др.

Последствиями семантического разрыва являются высокая стоимость разработки программных средств, их низкая эффективность и надежность, объективная трудоемкость, интеллектуальная и технологическая сложность самого процесса программирования. Для сокращения семантического разрыва используется повышение уровня абстракций языков моделирования в рамках объектно-ориентированной технологии [3]. Последнее снимает только часть проблем, не затрагивая существенным образом систему понятий языка программирования.

При объектно-ориентированном подходе представления о мире реализуются как множество объектов, характеризуемых поведением и состоянием, выраженным некоторым набором свойств (атрибутов). Объекты состоят между собой в отношениях агрегации и объединяются по своим свойствам и ограничениям в группы (классы), находящиеся в отношениях наследования [310]. Однако выразительные способности объектно-ориентированных языков, в конечном итоге, так и остались привязанными к встроенной в эти языки примитивной системе операций над простыми типами данных.

В итоге, проектирование информационных систем до сих пор выполняется в основном на интуитивном уровне с применением неформализованных методов, основанных на искусстве разработчиков, их практическом опыте, экспертных оценках и дорогостоящих экспериментальных проверках получаемых результатов. Кроме того, в процессе жизненного цикла информационная система подлежит постоянному изменению в соответствии с изменяющимися потребностями пользователей и развитием их представлений о предметной области, что еще более усложняет разработку и сопровождение таких систем.

Самым эффективным методом сокращения семантического разрыва видится приближение выразительных средств описания предметной области на различных уровнях

[1] Термин «семантический разрыв» в научный обиход ввел Г. Майерс [157]. Семантический разрыв был определен как мера различия принципов, лежащих в основе языков программирования высокого уровня, и тех принципов, которые определяют архитектуру ЭВМ. Однако мы будем придерживаться более общего взгляда на проблему семантического разрыва, трактуя ее как мера различие принципов, лежащих в основе различных уровней архитектурной иерархии вычислительных систем.

детализации к парадигмам, лежащим в основе деятельности разработчиков и пользователей информационной системы. Здесь под ***парадигмой*** понимается некоторая концептуальная схема или методология постановки проблем и их решения [142]. Функциональная, структурная и объектно-ориентированная парадигмы, очевидно, относятся к частным методологиям, ограниченным той системой понятий и типовых решений, на которых они базируются.

Таким образом, кардинальное сокращение семантического разрыва видится в том случае, когда понятия языка программирования совпадают или очень близки понятиям, используемым для описания результатов декомпозиции предметной области на значимые сущности. Однако ни одна из известных систем программирования эффективными средствами для описания прикладных понятий не располагает.

1.6. Концептуальный подход

Согласно господствующей в информационном подходе парадигме для формализации любой предметной области первичными признаются категории объектов и отношения между ними. Отсюда формально эмпирическая система отношений есть двойка $< O, R >$, где O – множество объектов предметной области, R – множество отношений между ними, причем во многих случаях ограничиваются исследованием только бинарных отношений.

Принимается, что схожие объекты образуют классы, возможно пересекающиеся. Далее полагают, что межобъектные отношения проявляются вследствие наличия у них определяющих свойств. Закономерным итогом такого подхода становится вывод, что исходным эмпирическим материалом для абстрактного представления предметной области служит перечисление объектов и их свойств.

Более естественным способом декомпозиции предметной области видится соотнесение с ней совокупности понятий, образующих ее понятийную структуру [88]. Это нашло выражение в образцах логико-понятийного анализа, рассматриваемых Платоном, эмпирико-научного – у Аристотеля; историко-политического анализа Полибия [231].

В последнее время получил свое развитие концептуальный подход и соответствующая ему парадигма, основанная на формализации понятийного аппарата. Появление концептуального подхода также связывают с проблемами формализации семантики знаковых систем [40]

Просматривается два основных направления в моделировании понятий [137]. Первое направление изучает понятия как некоторые самодостаточные сущности, независимые от систем знания. При этом обнаруживаются проблемы смысла, значения и интенсио-

нальности понятий, моделирования понятий в качестве предикатов, конструирования различных исчислений понятий, соотношений между понятиями и свойствами и т.п.

В частности, понятие в рамках первого направления рассматривается, по крайней мере, в пяти различных аспектах: как источник, как средство, как цель, как объект и как субъект познания; а концептуальный анализ в самом широком понимании отождествляется со знаковым анализом, т.е. относится к области семиотики. Понятие описывается в виде семиозиса, как целостная (полная) его реализация [169]. Семиозис определяется как пяти-членное отношение (V, W, X, Y, Z), в котором V (знак) вызывает в W (интерпретаторе) предрасположенность к определенной реакции X (интерпретанте) на определенный вид объекта Y (значение) при определенных условиях Z (в контексте). Иными словами, понятия – это отображения, которые ставят в соответствие сущностям из мира смыслов их реализации в мире знаков (в мире, отражаемом в сознании как знаковая система) с учетом познавательных целей воспринимающего субъекта. При этом, в зависимости от силе воздействия на интерпретатора, выделяется четыре типа знаков: знаки-десигнаторы (называющие), знаки-аппарайзеры (оценочные), знаки-прескрипторы (предписывающие) и знаки-форматоры (организующие).

Второе направление исходит из того, что анализ понятий предполагает рассмотрение тех или иных конкретных систем знания. Понятия рассматриваются как сложные и многоаспектные когнитивные феномены, изучаемые с помощью метода моделирования. Предложено, по крайней мере, два класса таких моделей: генерализующие и ассоционистские.

Первый имеет дело с изолированными понятиями, а также с процессами их построения, распознания и понимания. При этом свойства отдельного понятия определяются его внутренними структурами. Характерным примером этого класса моделей может служить теория формального анализа понятий [288].

В частности, в формальном концептуальном анализе [288] понятие отождествляется с определенной формой упорядочения сведений о подпадающих под него объектах и свойствах этих объектов. При этом упускается из виду, что столь же существенной характеристикой понятий являются средства представления и обработки сведений об этих объектах. Несмотря на то, что эти средства специфичны для каждой системы знания, предпринимаются попытки найти единый формальный аппарат определения семантики понятий, без учета особенностей рассматриваемой системы знания.

Второй класс моделей трактует понятия как компоненты ассоционистских (коннекционистских) систем. При этом свойства отдельного понятия выражаются через его

связи с другими понятиями. Классическими примерами этого класса моделей являются семантические [273], ассоциативные [238] и нейронные сети [331, 298, 304].

В рамках второго направления установлено, что понятия являются относительными. Соотнесение понятий с соответствующими им системами знания обнаруживает различие одноименных понятий из разных систем и требует использования специфичного аппарата формализации.

В итоге, в рамках концептуального подхода оказались не решенными проблемы типологии понятий по содержанию (содержательная эквивалентность и синонимия понятий); несогласованности коллективных и индивидуальных понятий; учета познающего субъекта, его целей и задач при формировании и использовании понятий; описания различия между понятием как категорией познания и его языковым выражением, выявление и выражение специфики понятий в различных областях знаний [136, 149, 199].

По мнению Л.Г. Кравцова [131] такое положение дел объясняется тем, что при обсуждении строения и функционирования понятия используются традиции, заимствованные из логики и философии. Практически все ключевые методологические положения, задающие специфику подхода к изучению мышления в рамках современной психологии познания, конфликтуют с устоявшимися логико-философскими представлениями о понятии и остро требуют радикального переосмысления.

Во-первых, мышление человека предметно, в том смысле, что ориентировано на открытие еще не известных существенных связей и отношений в познаваемом предмете. Понятие же, в соответствии с традиционными взглядами, принадлежит, скорее, области всего того, что нам уже известно о предмете познания, поэтому остается неясным, как оно может направлять сам мыслительный поиск.

Во-вторых, мышление человека рефлексивно, то есть так или иначе связано с саморегуляцией познавательных действий. Понятие, однако, обычно представляется как совокупность предметных признаков, что не позволяет даже поставить вопрос о том, способно ли понятийное мышление к самоуправлению, и если да, то что именно в понятии позволяет это сделать.

В-третьих, мышление человека субъектно, то есть сами мыслительные механизмы и средства активно перестраиваются человеком в процессе решения задачи. Однако ни разу понятийное мышление не осмыслялось как процесс работы над понятиями, как активное преодоление субъектом мышления готовых понятийных обобщений. Скорее, понятие рассматривается как такая единица мышления, которая своим устройством предопределяет весь ход решения мыслительной задачи.

В-четвертых, мышление человека продуктивно, то есть, связано с постоянным производством новых содержательных обобщений. Понятие же, как правило, понимается как устойчивое, фактически статичное образование, фиксирующее необходимые и достаточные признаки предмета, поэтому остается неясным, может ли оно вообще изменяться в процессе решения задачи, и если да, то за счет чего.

Однако понимание недостатков понятийного подхода отнюдь не умаляет полученных в это направлении прикладных результатов. Скорее всего, эти недостатки, будучи сформулированными, дают возможность по новому взглянуть на суть используемых средств словесно-логического представления знаний, выработать новые подходы к решению возникших при этом проблем. Тем более, что теоретическая и прикладная психология на настоящий момент не могут предложить никаких других способов репрезентации субъективных знаний, кроме как представления этих знаний и, связанных с этими знаниями, понятий в словесно-логической форме, в виде знаковых систем.

1.6.1. Концептуальный анализ

Считается, что методология точного понятийного анализа была разработана С. П. Никаноровым и его сотрудниками [176]. На ее основе созданы методы проектирования, применяемые для концептуализации различных предметных областей и решения задач организационного управления [175].

Методология представляет собой прикладную версию метода «восхождения от абстрактного к конкретному», основанную на синтезе родов структур – математическом аппарате, разработанном Н. Бурбаки [31]. Однако семантический разрыв между результатами формализации, базирующимися на выделении подмножеств декартового произведения или булеана других множеств, и их содержательной интерпретацией, делает эту методологию практически нереализуемой для областей с большим числом понятий. Другой недостаток концептуального проектирования – его плохая согласованность с методами абстракции, лежащими в основе самого процесса анализа предметной области.

Подход, наиболее близкий к концептуальному моделированию, реализован в модели данных «сущность-связь» [240, 212]. Модель ориентирована на описание таких предметных областей, которые могут быть представлены в виде изменяющихся данных с фиксированной структурой. При построении расширенной модели [267] используется абстрагирование понятий, однако элементами модели, в конечном счете, являются не понятия, а типы данных, причем используется трудно формализуемая семантическая разметка в виде дополнительных нотаций и ограничений. Другой недостаток метода – необходимость создания новой модели при выявлении в предметной области сущностей, не вписывающихся в существующую структуру типов.

В другой модели данных, названной «миварной», предпринята попытка сделать структуру данных изменяющейся (дополняемой) [37]. Модель предлагается строить на трех равнозначных и равнозависимых понятиях: вещь (объект, сущность), свойство (атрибут, характеристика) и отношение (связь, взаимодействие). Считается, что этих понятий достаточно для представления любой предметной области. Особо отмечается, что такие понятия как, например: элемент, объект, система, функция, структура, организация и подобные им, являются вторичными, производными по отношению к трем основным, а, следовательно, могут быть выражены через них.

Очевидно, что «миварной» подход близок к объектно-ориентированному, также основанному на описании объектов (вещей), имеющих состояние (совокупность свойств) и проявляющих поведение (взаимодействие с другими объектами).

1.6.2. Онтология

В технологиях, основанных на обработке знаний, используется термин «онтология», который заимствован из философии, но определяемый как общее описание (эксплицитная спецификация) некоторой области знания, в частности базовых понятий этой области и связей между ними [1, 292].

С одной стороны, несмотря на наличие множества вариантов определения этого термина, во всех таких определениях онтология рассматривается как спецификация некоторой концептуализации. С другой стороны, концептуализация рассматривается как продукт концептуального анализа, а содержание последнего интерпретирует как «абстрагирование существа физических объектов и их взаимосвязи в виде некоторой модели или теории, которая соответственно называется концептуальной моделью изучаемого объекта» [217]. Поэтому имеются основания отождествлять онтологию с концептуальной моделью предметной области.

Следуя подходу Т. Гавриловой и В. Хорошевского [80], дадим формальное определение онтологии O как тройки $O =< N, R, F >$, где N – конечное множество понятий (концептов) предметной области, R – конечное множество отношений между понятиями, F – конечное множество функций интерпретации вида $N^i \times R^j \to N$, i и j произвольные степени множеств N и R.

Между понятиями N устанавливаются отношения R, имеющие семантическую нагрузку. В качестве общих (базовых) отношений выделяют такие, как ***is a***, ***part of***, ***kind of***, ***contained in***, ***member of*** , ***see also*** и некоторые другие. Иными словами, множества N и R задают концептуальную схему онтологии или ее семантическую сеть $S =< N, R >$ [152].

Функции интерпретации определяют более сложные взаимосвязи между понятиями и их отношениями и, тем самым, выражают более сложную семантику. На данный момент существует несколько подходов к описанию онтологии и достаточно большое количество языков, реализующих эти подходы. Так в работах, посвященных глобальной семантической сети (Semantic Web) [280], онтология $O =< S, F >$ рассматривается как явное описание на некотором языке смысла F для терминов N, неявно определенных концептуальной схемой $S =< N, R >$.

Для поддержки построения онтологии разработан стандарт онтологического исследования IDEF5 [214], являющийся некоторой конкретизацией объектной методологии. В IDEF5 определяются специальные онтологические языки: язык доработок и уточнений EL (Elaboration Language) и схематический язык SL (Schematic Language). EL представляет собой формальный язык, предназначенный для описания элементов онтологии, а зависимости между элементами задаются в графической форме на языке SL четырьмя видами схем: диаграммами классификаций, композиционными схемами, схемами взаимосвязей и диаграммами состояния объектов.

С онтологическим подходом связывают ожидания формализации процессов спецификации, повышения надежности и обеспечения многократности использования моделей областей знаний. Язык онтологии выступает в этом случае как средство спецификации предметных областей и является декларативным. Появление новых качеств у систем моделирования основывается на анализе декларативного описания на языке онтологии. При этом говорят о формальном анализе, полагая, что онтология допускает формальный вывод (доказательство) наличия тех или иных свойств предметной области.

Однако из-за не решенной проблемы описания семантики формальных языков, результаты концептуализации и онтологического исследования до сих пор не являются точными формальными спецификациями предметной области. Как следствие этого, семантический разрыв между этими результатами и языковыми средствами, применяемыми для последующего детального описания решения задачи, остается труднопреодолимым.

Другим недостатком онтологического подхода в том виде, как это реализовано во множестве информационных систем, является отсутствие средств в рамках одной онтологии для явного указания проблематики описываемой предметной области и накопления проблемно-ориентированных знаний. Онтология предположительно должна быть концептуальной системой, открытой для пополнения новыми знаниями. Подобная специфика, означающая возможность ее модификации непосредственно перед использованием, для концептуальных моделей предметной области изучена недостаточно.

Считается [80], что онтология имеет сложную иерархическую структуру и разделяется на:

– онтологии верхнего уровня, описывающие такие общие понятия как пространство, время, материя, объект, событие и др.;

– предметные онтологии, описывающие понятия конкретной предметной области, например медицины, торговли, и т.п.;

– онтологии задач, конкретизирующие предметные онтологии за счет специализации понятий, введенных в онтологии вышележащего уровня;

– прикладные онтологии, зависящие как от конкретной предметной области, так и от задач, которые в них решаются.

В итоге, при определении онтологии нижнего уровня, происходит некоторая переоценка понятий вышележащей онтологии. Иными словами онтология верхнего уровня, в лучшем случае, предоставляет следующей онтологии только множество понятий (тезаурус) и их некоторую взаимосвязь (таксономию), а содержательная интерпретация понятий почти всегда изменяется, вплоть до полного переопределения.

В литературе нередко встречается расширенные трактовки форм представления онтологии. Например, из практических соображений В.В. Девятков [93] считает целесообразным неформальное представление онтологии на одном из естественных языков. Другое характерное расширение предложено В.А. Виттихом в виде «инженерной теории», основанной на группировке разнородных знаний в рамках единой концептуальной системы [45]. Для этого разработана методология построения таких теорий с применением средств компьютерного представления и обработки знаний.

Однако основная проблема онтологического подхода проистекает из попыток его универсализации. По мнению К. Айдукевича «Все суждения, которые мы принимаем и которые образуют картину мира, не определяются однозначно данными опыта, но зависят от выбора понятийного аппарата, с помощью которого мы интерпретируем эти данные. При этом мы можем выбирать тот или иной понятийный аппарат, изменяя тем самым всю картину мира. Это означает, что в той мере, в какой кто-либо пользуется определенной понятийной структурой, данные опыта заставляют его признавать определенные суждения. Однако сами по себе эти данные не вынуждают к безоговорочному признанию этих суждений. Мы можем выбрать иной понятийный аппарат, на основании которого те же самые опытные данные не требуют признания этих суждений, ибо в новом понятийном аппарате эти суждения вообще не фигурируют» [2, с. 231].

1.6.3. Концептуальное программирование

Концептуальное программирование основано на синтезе программы решения задачи, которое осуществляется по ее описанию на некотором языке и при заданных ограничениях. Эти ограничения требуют, во-первых, точного указания предметной области, к которой относится решаемая задача, и, во-вторых, фиксации класса решаемых задач. В общем случае описание решения имеет вид:

Зная M, вычислить $y_1, y_2, ..., y_n$ по $x_1, x_2, ..., x_m$.

где M – идентификатор предметной области, y_i $(i = \overline{1,n})$ – идентификаторы переменных, значения которых необходимо вычислить, x_i $(i = \overline{1,m})$ – идентификаторы переменных с известными значениями [16].

Для представления предметной области используются семантические сети специального вида, называемые вычислительными моделями. Семантическая сеть S определяется следующим образом:

$$S =< O, R >,$$

где O – множество объектов предметной области, $O = \{o_i \mid i = \overline{1,u}\}$; R – множество отношений между объектами, $R = \{r_j \mid j = \overline{1,v}\}$. Вычислительная модель предметной области определяется как упорядоченная тройка множеств (кортеж):

$$< \{p_i \mid i = \overline{1,n}\}, \{f_j \mid j = \overline{1,m}\}, \{u_k \mid k = \overline{1,q}\} >,$$

где p_i – имя понятия предметной области; f_j – функциональное отношение между понятиями; u_k – управляющая структура. Функциональное отношение f_j (плекс-элемент) задается тройкой

$$f_j =< X_j, F_j, Y_j >,$$

где $X_j = (x_{j1}, \ldots, x_{jm_j})$ – набор входных переменных для f_j; F_j – ссылка на процедуру (программный модуль), реализующую вычисление $Y_j = F_j(X_j)$; $Y_j = (y_{j1}, \ldots, y_{jn_j})$ – набор выходных переменных для f_j. Управляющие структуры u_k является отображением X_j и Y_j в множество разрешенных типов данных. Кроме того, управляющие структуры содержать элементарные фрагменты программ, которые необходимы для присваивания переменным известных и вычисленных значений, а также для соотнесения переменных с понятиями предметной области.

При концептуальной программировании концептуальная модель предметной области, представленная в виде семантической сети, транслируется в некоторую систему ак-

сиом, управляющую функционированием инструментальной системы и позволяющей пользователю решать задачи без программирования – путем описания задачи и исходных данных. Программирование осуществляется автоматически планировщиком из набора готовых программных модулей, относящихся к конкретной предметной области. В этом случае для решения задачи ищутся не длинные выводы в системах с малым числом аксиом, а сравнительно короткие выводы с большим числом аксиом, что становится возможным вследствие известной схемы решения задачи, которая вводится заранее, на этапе формализации предметной области. Технология концептуального программирования реализована в серии программных решателей ПРИЗ (Микро-Приз, Эксперт-Приз). Общим для них является язык УТОПИСТ (Универсальный Транслятор описаний Теорий) [223].

В концептуальных программах, в которых каждое отношение между понятиями интерпретируется несколькими функциями, отсутствует априорное разделение связанных отношением переменных на аргументы и результаты. Такое разделение в этих моделях реализуется динамически, поскольку к моменту начала вычислений часть переменных должна быть полностью определена, – тем самым образуется множество потенциальных аргументов для функций, интерпретирующих отношения. Те из функций, у которых все аргументы полностью определены, выполняются, добавляя свои результаты ко множеству полностью определенных переменных. Это позволяет вычислить очередные функции, вплоть до получения значений всех или требуемых переменных. В такой форме вычислительные модели являются удобным средством организации автоматического синтеза программы вычислении [172].

Для синтеза программ используется два метода:

– дедуктивный – построение программы выполняется на основе доказательства, что решение задачи существует;

– индуктивный – программа строится по примерам, каждый из которых определяет ответ для некоторого подкласса исходных данных.

В итоге, при концептуальном программировании используется низкоуровневый язык логического вывода в интуиционисткой логике, где неприменимы законы двойного отрицания и исключения третьего, причем для каждого правила вывода формируются программные конструкции, задающие программуню реализацию формул, выводимых по этому правилу.

1.6.4. Концептное программирование

Известно, что структурные отношения в предметной области проявляются при наличии у объектов свойств ссылочного типа [124]. Значениями таких свойств являются ссылки на другие эмпирические объекты, а не на элементы некоторого априори заданного

множества – классы. Более того, поскольку эмпирические объекты неоднородны, то существует априори неизвестная типология необходимых объектных ссылок. И, наконец, у самих объектов следует ожидать различий в способности вступать в структурные отношения [210].

Концептно-ориентированное программирование позиционируется как новый подход, который реализует концепцию ссылочной целостности предметной области и, тем самым, обобщает объектно-ориентированный подход и другие существующие технологии [334].[2]

Таким образом, концептно-ориентированное программирование следует рассматривать как некоторую модификацию объектно-ориентированного, где используется новая программная конструкция, названная концептом. Концепт определяется как структура, состоящая из объектов двух классов: объектов класса объекта и объектов класса ссылки. В противоположность традиционным классам объектно-ориентированного подхода, концепт отражает две стороны любой программы: явно вызываемые целевые методы, и неявно выполняемые методы представления и доступа. Неформально отношение между концептом и классом аналогично отношению между комплексными и действительными числами в математике.

Концепты, как и классы, образуют иерархию, основанную на наследовании. Главная роль родительского класса состоит в предоставлении окружения (контейнера, пространства) для дочернего концепта. Так как каждый концепт состоит из двух классов, то имеется и два типа реализации концептов. Реализация класса объектов рассматривается как объект, который передаются методам по ссылке. В свою очередь реализация класса ссылок рассматривается как ссылка и передается по значению. В итоге каждый объект имеет одну ссылку, которая его представляет, а каждая ссылка определяет его контекст. Контекст, как и любой объект, в свою очередь имеет свой собственный контекст.

Таким образом, концептно-ориентированный подход, кроме обычных возможностей по моделированию формата объектов и их функций, которые используются в объектно-ориентированном программировании, имеет средства для реализации связей между объектами. Для этого водится программная конструкция, которая, в явном виде, позволяет выразить взаимосвязи между объектами и, в неявном виде, моделирует взаимосвязи между классами. Однако семантика такой взаимосвязи является в каком-то смысле недоопределенной, так как зависит от состояния и поведения объекта ссылочного типа.

---

[2] В оригинале используется термин «концептуальное программирование». Для отличия от достаточно устоявшейся области концептуального программирования [223], будем использовать термин «концептное программирование», тем более, что описываемый подход основнывается на понятии «концепта».

1.7. Представление и обработка знаний

Данные, информация и знания есть основные понятия, используемые в информационных системах, основанных на обработке знаний.

Данные будем рассматривать как совокупность состояний некоторого объекта (процесса, явления), которые используются для представления знаний (в системах искусственного происхождения) или являющихся их источником (естественные объекты, явления, процессы).

Знания постигаются субъектом из данных, воспринимаемых как содержащие некоторую информацию. В зависимости от эмоционального состояния субъекта, особенностей его восприятия и имеющихся у него представлений, а также ранее полученных знаний, одни и те же данные воспринимаются как содержащие разную информацию.[3] Следовательно, ***информацию*** можно определить как некоторую содержательную интерпретацию данных, осуществляемую при определенном, возможно неполном знании их структуры и правил использования.

Как правило, для передачи знаний используется синтаксически и семантически разомкнутые формы представления, основанные на существовании некоторой подразумеваемой модели как предметной области, относительно которой эти знания выражаются, так и формы представления передаваемых знаний. Только согласовав эти модели у источника и получателя знания, а именно, базовые (подразумеваемые) знания о предметной области и знание структуры и правил содержательно интерпретации используемой формы представления знаний, появляется возможность адекватной их передачи.

Некоторое, иногда существенное искажение в этот процесс, вносит проблемная ориентация получателя знания, отличающаяся от подразумеваемой источником. Отсюда получаем, что данные – это форма выражения (объективации) знаний, а информация – это форма их восприятия[4].

---

[3] Близкие представления на природу информации содержатся в [246], где для получения информации требуется наличие информационного среза (уровня рассмотрения) и априорной информационной модели у субъекта. Вводится понятие семантической информации, рассматриваемое как активное субъективное отражение материального мира, используемое для построения субъективной модели проблемной среды и себя в этой среде.

[4] В [178] восприятие рассматривается как интегральная форма чувственного познания, представляемая в виде комплексов ощущений, которые организованы в пространстве и времени. Восприятие предполагает участие в своем формировании других форм чувственного и рационального познания, которые отражаются в сознании воспринимающего субъекта в виде некоторой целостной модели окружающей его среды.

В самом общем виде ***знание***[5] определяется как проверенный на практике результат отражения объективной действительности, представленный в сознании субъекта в виде понятий и суждений, утвержденных некоторой последовательностью умозаключений [92]. Необходимость в утверждении перечисленных содержаний познания диктуется природой знания – его обоснованной личностной истинностью.[6]

С гносеологической точки зрения знание определяется как субъективно обоснованное истинное убеждение [34]. Тем самым признается субъективная (внутренняя) природа знания. В объективном смысле знание рассматривается как представленный во внешней форме результат субъективного познания, признаваемый объективно истинным в некоторый исторический момент [178]. Считается, что идеальность знаний является адекватным следствием тех свойств внешнего мира, которую они отражают.

Знания формируются в результате целенаправленного педагогического процесса, самообразования и жизненного опыта. Отсюда, в частности, следует, что знания нуждаются в своей объективации,[7] т.е. отчуждении от носителя в некоторой внешней объективной форме. Так как во внутренней (идеальной) форме передачу знаний осуществить нельзя, то используются внешние формы в виде специальным образом обустроенных данных. Историческим примером такого обустройства является естественный язык (язык-речь и язык-письмо).

Язык представляет собой одну из форм выражения знания, отличающуюся высокой степенью латентности. Знание в языковой форме оторвано от гносеологической основы своего появления и не предполагает осмысленного своего использования без знания самого языка. Овладевая языком, субъект получает некоторую совокупность знаний (метазнаний), на основе которой у него и появляется возможность извлечения знаний языковыми средствами (предметных знаний).

Уточнение понятия «истины» с помощью технических средств логической семантики осуществлено А. Тарским [351], где автор исходит из классического представления об истине, согласно которому термин «истинно» выражает свойство нашего знания, в частности свойство высказываний, а не объективной действительности. Для естественного

---

[5] Знание – сложное психическое явление. Психика рассматривается как способность субъекта к активному отражению объективной реальности. Психические явления составляют необходимое внутреннее содержание предметной деятельности человека и проявляются через его способность к воспроизведению в своем сознании с различной степенью адекватности объектов внешнего мира [23].

[6] Субъективное знание, в отличие от объективированного в тексте или речи, предполагает хотя бы минимальную рефлексию. Такая рефлексия есть тоже знание, а именно знание о знании или метазнание [97].

[7] Под объективацией понимается признание знания, выраженного в некоторой своей внешней форме, объективно истинным и социально значимым [19].

языка задача построения общего определения истины не может быть решена по причине «семантической замкнутости» естественного языка. Для устранения подобных парадоксов Тарский считает необходимым разделить язык на две части: объективный (предметный) язык и метаязык. Определение истины по Тарскому должно формулироваться в метаязыке.

Обычно под метаязыком понимается язык, средствами которого проводится описание структурных, дедуктивных или семантических свойств какого-либо другого (предметного) языка, являющегося предметом изучения соответствующей метатеории, где под метатеорией понимается исчисление, призванное судить о предметном языке [24].

Таким образом, в любой замкнутой семиотической системе,[8] каковой является естественный язык, по определению присутствует два языка: язык для выражения знания (предметный язык) и язык для выражения знаний о знании (метаязык). Заметим, что метаязык может рассматриваться как состоящий из двух языков: синтаксического языка – для определения синтаксиса форм выражения знания, и семантического языка – для описания семантики таких форм.[9]

Последовательный способ передачи языковых единиц ограничивает возможности передачи знания во всей его совокупной целостности. Множественные ассоциативные связи и связи обобщения между элементами знания, выраженные соответствующими языковыми единицами, передаются и воспринимаются только последовательно. Однако линейная последовательность языковых единиц, в конечном итоге, представляется в виде структурированного знания, и является выражением взаимосвязей между понятиями описываемой предметной области.

В общем виде ***данные*** рассматриваются как факты, характеризующие объекты (явления, процессы) предметной области, полученные на основе фиксации (спецификации) их свойств. В информационных системах данные представляются в форме, пригодной для постоянного хранения, передачи и обработки, а получаются путем измерения, наблюдения, выполнения операций над другими данными и т.п. Данные записываются и хранятся на носители данных в кодированном виде. В процессе функционирования происходит изменение данных, благодаря чему информационная система изменяется свое состояние.

Под ***знаниями*** в информационных системах понимается специальным образом организованная совокупность единиц данных, обладающая следующими свойствами [128]:

---

[8] Семиотика изучает производство, строение и функционирование различных знаковых (семиотических) систем, предназначенных для представления, хранения и передачи знания.

[9] В семиотических системах естественного происхождения все три языка (предметный, синтаксический и семантический) объединяются в один, причем граница между языками достаточна условна.

– внутренняя интерпретируемость (уникальность идентификации единиц);

– структурированность (включение одних единиц состав других);

– связность (связывание единиц различными типами отношений);

– шкалирование (соотнесение единиц между собой с помощью различных шкал);

– семантическая метрика (специальная шкала ситуационной близости);

– активность (взаимосвязь единиц по изменениям).

Такие черты, как внутренняя интерпретируемость, структурированность, связность, семантическая метрика и активность присущи любым более или менее крупным блокам человеческих знаний и, в этом смысле, знания в информационной системе можно рассматривать как модель или образ (в широком понимании данного слова) того или иного фрагмента человеческого знания.

Таким образом, под представлением знаний понимается их структурирование с целью формализации процессов решения задач в определенной проблемной области. Основные проблемы, возникающие при этом, вытекают из субъективного характера знания. В частности, известна проблема извлечения знаний, заключающаяся в трудностях объективации знаний эксперта при построении экспертных систем [80]. Другие проблемы связаны с выбором модели представления знаний и методов их обработки.

1.7.1. Математическая логика

Исследования в области представления и обработки знаний находятся под сильным влиянием современной математической логики. Общим методологическим основанием данного направления является «компьютерная метафора», которая предполагает, с одной стороны, наличие некоторых фиксированных структурных этапов или уровней переработки данных, а с другой – существование единого формального в своей основе метаязыка описания знания, аналогичного машинному коду вычислительных устройств [41].

Математическая логика – наука об общих структурах и законах правильного мышления, образования и сочетания понятий и высказываний, о правилах умозаключений независимо от их конкретного содержания [127]. Если изучение функциональных систем является прерогативой дискретной математики, то изучение логики как исчисления относится к сфере математической логики, которая занимается обоснованием суждений, доказательств и логическим выводом.

Фактически современную математическую логику основал Г. Фреге. В 1879 г. он предложил Begriffsschrift («исчисление понятий» или «систему обозначения понятий») [287]. Это был самый значительный шаг в логике со времён Аристотеля. Формальна система Фреге позволила логикам разработать строгое определение доказательства. Кроме того, Фреге был первым, кто серьёзно отнёсся к идее применения функций к сущностям

отличным от чисел и, в частности, к другим функциям. Большое значение в свое время имела работа Ж. Эрбрана [296], в которой он предложил несколько версий процедуры доказательства теорем в логике первого порядка.

Влияние математической логики проявляется в том, что большинство моделей представления знаний задуманы и построены как машинные программы. При этом во всякой программе, реализующей некоторой алгоритм, выделяется две составляющих: логику, описывающую собственно решение проблемы и управление процессом вычислений, сама программа рассматривается как теория, а вычисления представляют собой вывод в этой теории [305].

Как правило, общая постановка задачи основывается на глобальных моделях понимания текста. Теоретически каждая такая модель включает четыре компонента. Первый – это так называемый «парсер», который осуществляет преобразование лингвистических или невербальных данных к виду, соответствующему внутренней репрезентации знания. Второй компонент – база знаний, где фиксируются результаты анализа. Третий компонент – структуры управления, которые определяют алгоритмы распознавания, поиска, логического вывода и т. д. Четвертый компонент полностью симметричен первому, обеспечивая переход от внутренней репрезентации знания к целостным «квазицелесообразным» ответам.

Общим недостатком глобальных моделей является то, что при их построении структура репрезентации знания выбирается более или менее произвольно. Исходным мотивом для введения пропозиционального описания знания было желание свести множество поверхностно различных высказываний к более простому набору базовых семантических элементов (компонентов, примитивов, ролей и т. д.). Этот подход представляется крайне проблематичным, ибо даже формула исчисления предикатов может быть по разному интерпретирована и, более того, бесконечным числом способов выражена с помощью средств других систем математической логики. В итоге, упрощение и гомогенизация обернулись в этой области редкой путаницей в терминологии и бесконтрольным размножением моделей, проверка которых исключительно трудна [41].

Хотя результаты эмпирических исследований и накладывают иногда некоторые ограничения на глобальные модели понимания текста, представления и обработки знания, выбор того или иного формального аппарата и интуиция автора имеют несоизмеримо большее значение, чем объективные закономерности, присущие рассматриваемой проблеме.

1.7.2. Модели знаний в информационных системах

Модель представления знаний определяется как формализм, предназначенный для отображения знаний о некоторой предметной области и специфицирующий ее статические и динамические свойства. Различие подходов для представления знаний определяется теми формальными средствами, которые используются для описания [112]:

– в семантических сетях и онтологии – это понятия и их взаимосвязи;

– в фреймовой модели – описание фактов в виде сети фреймов;

– в логической модели – предикаты и логические формулы;

– в продукционной модели – секвенции или предложения вида «если-то»;

– в логико-лингвистической модели – аксиомы и правила вывода;

– в объектной модели – объекты, классы и сообщения.

Характерным примером декларативной модели представления знаний является сетевая модель. ***Сетевая модель*** представляется в виде совокупности взаимосвязанных понятий, образующих семантическую сеть. Связь между понятиями сетевой модели выражает минимальный объем знаний, простейший факт, относящийся к двум понятиям. Предметная область в любой момент времени может быть представлена в виде совокупности сущностей, понятий и ситуаций, называемой ее состоянием. Каждой ситуации можно поставить в соответствие некоторое утверждение или суждение об ее истинности или ложности.

Основа семантической сети – события, атрибуты, комплексы признаков и процедуры. События – это суждения, факты, результаты наблюдений, рекомендации, которые могут представляться словосочетаниями и числами и группируются тематически или функционально в разделы. Атрибут – это характеризующее событие, имеющее несколько значений. Несколько атрибутов могут объединяться в комплекс, характеризующий событие в большей степени, чем отдельный атрибут. Процедура – это специфический компонент сети, выполняющий преобразование данных. Она позволяет вычислять значения одних атрибутов на основании других. В сетевой модели для логического вывода события делятся на исходные (признаки) и целевые (гипотезы).

Логико-лингвистическая модель относится к классу продукционных и основана на семантическом структурировании данных. Вместо алгоритма, предписывающего на каждом шаге его реализации некоторое однозначное решение, в логико-лингвистической модели используется совокупность указаний, представленных в виде некоторого исчисления. Аксиомами исчисления служат описания объекта моделирования, среды и начальных состояний. Правила вывода – это правила перехода из одного состояния объекта и среды в

другое. Теоремы – промежуточные и конечные состояния системы. В логико-лингвистических моделировании используется:

– интерпретатор, который отражает изменение блока знаний о среде, содержимое которого меняется в процессе функционирования: обновляется, уточняется, пополняется;

– модель знаний, которая отделена от механизма порождения решений и благодаря которой существенно упрощается описание информационной системы и ее функционирования.

В ***фреймовой модели*** фиксируется жесткая структура единиц данных (фреймов), являющихся, по своей сути, совокупностью пар «имя–значение». Имена как единицы данных используются для именования фреймов и их структурных составляющих (слотов). Значения, в свою очередь, задаются произвольными единицами данных, в том числе и другими фреймами.

В ***объектной модели*** знания представляются как множество объектов, характеризуемых поведением и некоторым набором свойств (атрибутов). Объекты создаются и уничтожаются, а во время своего существования состоят в отношениях агрегации и объединяются по своим свойствам и ограничениям в группы (классы), находящиеся в отношениях наследования.

В настоящее время нет ни одной модели знаний, которая обладала бы всеми необходимыми свойствами, а именно интерпретируемостью, структурированностью, связностью, наличием шкал и семантической метрики, активностью [34].

С другой стороны, несмотря на разнообразие предложенных решений (продукционные системы, фреймы, семантические сети, логика предикатов, нечеткие множества, и т.д.), все они основаны на идее логически организованной структуры знаний. Но если в современной когнитивной психологии очевидно, что логическая форма репрезентации знания отнюдь не единственная [103], то в инженерии знаний других форм пока не существует.

Все это требует нового переосмысления понимания познавательной деятельности и ставит задачу поиска таких подходов к представлению и обработки знаний, в которых философский, психологический и технический аспекты были бы представлены не как разные предметы различных научных дисциплин, а как различные явления одной и той же сущности.

1.7.3. Обработка знаний

Нынешнее состояние дел в области обработки знаний отнюдь не предвещает решения проблемы содержательной декомпозиции предметной области и использования полученных результатов для создания ее точных математических моделей. Имеется разрознен-

ный набор решений, направленных на создание специализированных устройств и технологий [94]. При этом используются два противоположных по своей сути подхода: подход, основанный на моделировании отдельных функций интеллекта; и подход, ставящий своей целью получение того же результата, но другими средствами, не основываясь на копировании имеющихся представлений об устройстве интеллектуальных функций [153].

К наиболее масштабным проектам первого типа можно отнести деятельность Artificial Development (AD) по созданию системы CCortex, которая должна стать самой крупной в мире нейронной сетью [276]. Примером второго подхода может служить распределенная система поддержки принятия решений семиотического типа [33] и ее реализация в инструментальной среде G2-GDA [99].

При обработке знаний используются два основных метода:

– логический (формальный) метод, при котором осуществляется формальный вывод на знаниях, представленных в рамках одной из моделей [151];

– эвристический (когнитивный) метод, при котором ставится задача поиска и реализации различных эвристик [202].

Помимо всего прочего, узким местом как первого, так второго методов является использование неадекватных постановке задачи технологий обработки данных, основанных на создании математических моделей, описывающих этот процесс в виде программы, написанной на универсальном языке программирования с достаточно бедными выразительными средствами.

При создании информационных систем было установлено, что существуют прикладные задачи, существенно отличающиеся от расчетных. Они связаны с обработкой символов, плохо представляются в виде единственного жестко заданного алгоритма. Поэтому в основу языков программирования предложено положить два новых принципа [113]: ориентация на символьную обработку и декларативный подход к программированию.

Наиболее часто для представления и обработки знаний используются парадигмы функционального и логического программирования. От структурного и объектно-ориентированного подходов они отличаются нелинейным выводом решений и низкоуровневыми средствами поддержки анализа и синтеза структур данных.

Среди языков обработки знаний ведущее место занимают LISP [319], реализующий функциональный подход к обработке строк, и PROLOG [27] – основанный на декларативном подходе. В процессе исследований возникали новые подходы и языки. К настоящему времени лишь немногие из них получили хотя бы ограниченное распространение – сюда можно отнести POP, Tablog, LOGO, LogLisp, OPS, SNOBOL, Рефал.

Например, в языке обработки строк SNOBOL [282] сопоставление с образцом осуществляется на основе грамматик в нормальной форме Бэкуса-Наура. Для реализации языка используются виртуальные макрокоманды, настраиваемые для каждой целевой платформы. В основу языка Рефал [221, 222] положено понятие рекурсивной функции, определенной на множестве строк. При обработке строк активно используется концепция поиска по образцу, характерная для языка SNOBOL. Рефал вобрал в себя лучшие черты наиболее интересных языков обработки текстовых данных. В настоящее время язык Рефал-5 используется для автоматизации построения трансляторов, систем автоматических преобразований, а также, подобно языку LISP, в качестве инструментальной среды для реализации языков представления знаний.

Наиболее мощные средства для обработки знаний предоставляют продукционные системы. В продукционных языках программа задается совокупностью правил без явного указания порядка их применения. Помимо этого, правила содержат либо условие и действия, которые должны быть выполнены в случае истинности этого условия, либо условие и совокупность других условий, достаточных для истинности этого условия [90].

Продукционные системы, например OPS [284], имеют хранилище правил, рабочую памяти и интерпретатор, реализующий некоторую стратегию управления. Фактически правило является парой «предпосылка–действие»; где предпосылка – это совокупность условий, а условия – это предположения о наличии некоторых свойств, которые принимают логические значения и выражаются, например, на многосортном языке исчисления предикатов первого порядка [171].

Правила в продукционных моделях имеют следующую обобщенную форму:

$$P_1, P_2, \ldots, P_n \rightarrow Q_1, Q_2, \ldots, Q_m$$

которая читается следующим образом: если предпосылки P_i верны, то выполнить действия Q_j. Основная функция рабочей памяти – хранить данные в формате векторов «объект-атрибут-значение». Эти данные используются интерпретатором, который в случае присутствия (или отсутствия) определенного элемента данных в рабочей памяти активизирует те правила, предпосылки в которых удовлетворяются наличными данными.

Эффективность продукционных систем достаточно низка, так как много времени тратится на проверку условий применимости правил. Преодоление недостатков программирования в порождающих правилах видится не на пути усложнения существующих языков программирования, а скорее на пути объединения их с другими парадигмами программирования, позволяющими использовать рекурсивные структуры данных и управления [96].

Однако большинство реальных задач не сводится к набору аксиом и эвристик, и человек, решая эти задачи, не использует классическую логику и фиксированный набор правил, поэтому модели, реализуемые на этих языках, при всех своих преимуществах, имеют существенные ограничения по классам решаемых задач и эффективности их решения.

1.7.4. Ситуационное управление

Использование общих методик и универсальных декомпозиционных схем для решения прикладных задач большой размерности в рамках формально-логического подхода практически неосуществимо. Последнее объясняется предельной абстракцией предметной области. В этом случае математическая модель представляется в формах с большим числом степеней свободы, например, в виде таблиц дискретных функций. Понятно, что при отсутствии содержательных представлений все варианты синтеза модели равноправны и нет оснований заранее, до проведения синтеза, предпочесть один вариант другому (вернее, эти основания были опущены, вынесены за рамки решаемой задачи). Поэтому для синтеза эффективных математических моделей приходится выполнять полный перебор вариантов.

Попытки решить проблему перебора путем разработки соответствующего формализма не увенчались успехом. Если формальный аппарат прост, то описание модели становится необозримым, так как осуществляется через конечное множество простых примитивов. Если формальный аппарат достаточно развит, то и описание упрощается. По мере расширения предметных областей, попавших в область изучения и формализации, количество базовых примитивов приходится постоянно увеличивать. В этом случае приходим к необозримости формального аппарата, что следует из принципиальной непополнимости формальных систем, открытых К. Геделем [166].

В 60-х годах был выполнен ряд пионерских исследований, возглавленных В.Н. Пушкиным и Д.А. Поспеловым [198], целью которых было выяснение, как же, в действительности, человек решает переборные задачи? Выяснилось, что решение таких задач осуществляется на основе преобразования текущей ситуации, таким образом, чтобы результат преобразования оказывался сопоставимым с некоторой обобщенной ситуацией, решение которой известно или очевидно. Эти механизмы и составили содержание области исследований, которую стали называть ситуационным управлением [193].

Известно, что большинство кардинальных научных открытий произошло путём неожиданных «прозрений», т.е. интуитивно, а не путём формальных логических построений. Современная формальная логика с ее аксиоматическими построениями довольно далеко отошла от естественного мышления и не сделала своим непосредственным предме-

том структуру теоретического мышления[10]. На это обстоятельство обратил в свое время А. Эйнштейн: «... чисто логическое мышление само по себе не может дать никаких знаний о мире фактов; все познание реального мира исходит из опыта и завершается им. Полученные чисто логическим путем положения ничего не говорят о действительности...» [252]. Последнее подтверждается большим числом самых разных программ, например, для автоматического доказательства теорем, которые не дали ожидаемых результатов и, как теперь стало ясно, не могли их дать ввиду чрезвычайной сложности задачи, заведомо недоступной для сколь угодно мощной вычислительной техники. По мнению Ю.И. Янова [258] этот факт свидетельствует о наличии у человека особой способности решать задачи комбинаторной сложности, причем эта способность является не формализуемой в рамках известных подходов и, следовательно, не может быть реализована автоматами.

Интересен взгляд А.Л. Шамиса [246, с. 39], который характеризует мышление человека как попытку свести многоэкстремальную задачу, которую можно решать только методом перебора, к градиентной, одноэкстремальной, которую вообще не надо решать, «так как в каждой точке нужно оценить и сравнить очень ограниченное число вариантов. То есть, человек создает новое информационное отображение среды: он решает задачу с помощью обобщения и укрупнения[11]. За счет этого в знакомой среде можно действовать почти автоматически – по прогнозируемому градиенту эмоциональной оценки ситуации. Таким образом, мозг нужен для создания одноэкстремальной модели среды, позволяющей поменьше думать, полагаясь в знакомых ситуациях на эмоциональный выбор». И далее. «Важнейшим аспектом восприятия является предвидение на основе иерархичной модели мира и многоуровневого процесса восприятия. В знакомой среде и знакомых ситуациях восприятие идет на уровнях обобщений (общее – частное) и укрупнений (целое – часть) и состоит в подтверждении предвидения на этих уровнях. Обращение к уровню детального восприятия происходит только по мере поведенческой необходимости или при рассогласовании предвидения и реального выхода» [там же, с. 49].

Однако в области информационных технологий описанные подходы лишь обещают устранение трудностей. На деле до сих пор решается многоэкстремальная задача преодо-

---

[10] Мышление – процесс познавательной деятельности субъекта, характеризующийся обобщенным и опосредованным отражением действительности. Различают следующие виды мышления: словесно-логическое, наглядно-образное, наглядно-действенное. Выделяют также мышление теоретическое и практическое, теоретическое и эмпирическое, логическое (аналитическое) и интуитивное, реалистическое и аутистическое, продуктивное и репродуктивное, непроизвольное и произвольное. Мышление является составной частью и особым объектом самосознания личности, в структуру которого входит понимание себя как субъекта мышления, дифференциация "своих" и "чужих" мыслей, осознание еще не решенной проблемы и своего отношения к ней [132].

[11] Под обобщением автор понимает способность переходить от частного к общему, а укрупнение рассматривается как способность соединять части в целое.

ления семантического разрыва между содержательными представлениями относительно предметной области и используемыми для формализации этих представлений математическими моделями, выраженными, например, на некотором универсальном языке моделирования или программирования. Применяемые языки не имеют развитых средств определения новых языковых конструкций, задания их синтаксиса, семантики и прагматики.

Последнее объясняется рядом методологических проблем, среди которых следует выделить:

– отсутствие единого теоретического базиса для структурирования знаний и несовершенство используемых для этого математических моделей (их дескриптивный, а не конструктивный характер);

– невозможность точной формальной спецификации предметной области и решаемых в ней задач (не разработаны средства определения семантики и прагматики формальных языков и высокоуровневых спецификаций предметной области).

1.8. Семантика формальных языков

Основная проблема концептуального подхода – формальное (объективированное) описание семантики понятий, которое устанавливает соответствие между сущностями из мира смыслов их реализацией в мире знаков.

Текстовая форма выражения понятий, как обобщение исторически сложившегося метода, видится наиболее удачной для описания понятий на прикладном и теоретическом уровне. Гносеологическая роль языка, вернее отсутствие у него самостоятельной познавательной функции, исследована Р. Карнапом. «Если кто-либо хочет говорить на своем языке о новом виде объектов, он должен ввести систему новых способов речи, подчиненную новым правилам; мы назовем эту процедуру построением языкового каркаса для рассматриваемых новых объектов» [119].

Несмотря на определенные успехи, достигнутые в области глобального моделирования естественного языка, имеются основания для сомнения в его перспективности. Прежде всего, глобальное моделирование оказывается наиболее успешным на поверхностных уровнях языка и вызывает тем больше затруднений, чем более глубинный уровень рассматривается. «Можно построить модель морфологии с высокой степенью полноты, довольно далеко зайти в формальном описании синтаксического уровня, описать некоторые соответствия между глубинно-синтаксическим уровнем и гипотетической смысловой записью. Говорить же о моделировании языка на семантическом уровне вообще трудно, поскольку неясно не только, как моделировать, но и что моделировать. Трудно в масштабе всего языка указать, какие процессы или преобразования на уровне смысла мы должны

воспроизвести в формальной модели, чтобы можно было считать этот уровень освоенным.» [237]

Таким образом, о синтаксисе языков известно довольно много, менее изучен вопрос корректного определения их семантики.

1.8.1. Проблемы описания семантики

Считается, что задача формального определения семантики естественного языка не может быть решена из-за его «семантической замкнутости». Семантически замкнутым называется язык, который содержит в себе как выражения, относящиеся к некоторым внеязыковым объектам, так и выражения, относящиеся к характеристике самого языка. Всякий естественный язык является семантически замкнутым, что приводит к возникновению в нем противоречий и парадоксов. Например, пытаясь ответить на вопрос, истинно или ложно высказывание «Данное предложение ложно», мы приходим к противоречию.

Однако понятие истинности присуще не объективной действительности, оно выражает не более чем одно из возможных свойств высказываний [350]. Следовательно, смысл «противоречивых» высказываний может быть различен, в зависимости от контекста их применения. Например, ранее использовалось высказывание, которое в одной из своих интерпретаций порождает антиномию. Но это высказывание не разрушило семантическую структуру настоящего текста, так как было применено в особом контексте и для выражения особого смысла. По этому поводу вполне резонно высказывание Д. Гильберта: «…закон исключённого третьего ни в малейшей степени не повинен в появлении известных парадоксов теории множеств; эти парадоксы происходят скорее потому, что пользуются недопустимыми и бессмысленными образованиями понятий…» [258].

Аналогичные результаты получены Б. Расселом в своей теории логических типов [230]. Для разрешения парадоксов теории множеств, связанных с существованием множеств, которые могут быть элементами самих себя, потребовалась классификация множеств как относящихся к различным типам и запрет на их членство в множествах, относящихся к типу меньшего или равного значения.

Возникновение противоречий вызвано не характером исходных понятий и их свойств, а тем, что их язык оказывается настолько широким, что позволяет определять внутренне противоречивые ситуации и понятия, например такое, как «множество всех множеств». Подобное явление присуще естественному языку в силу его универсальности, что выражается в существовании различных «реальных» противоречий вроде «парадокса лжеца» и др. Следует сказать, что любую формальную теорию, путём введения новых понятий или чрезмерного расширения объёма существующих понятий можно сделать противоречивой. Поэтому введение новых понятий не только в формальную, но и в содержа-

тельную теорию должно сопровождаться тестом на непротиворечивость. Это вселяет определенные надежды на возможность решения проблем, связанных с возможной противоречивостью языков по причине их семантической замкнутости.

Однако всякий естественный язык является семантически замкнутым. Чтобы избежать возникновения противоречий при построении формальных языков, различают предметный язык и метаязык. В частности, для описания семантики А. Тарским предложено разделить язык на две части: объективный (предметный) язык, служащий для выражения высказываний, и метаязык, предназначенный для описания семантики этих высказываний [350, 351]. На предметном языке говорят о той или иной области объектов, а на метаязыке обсуждают свойства предметного языка. Благодаря этому разделению в языке не могут появляться предложения, говорящие о самих себе.

Метаязык как одно из основных понятий современной математической логики и теоретической лингвистики используется при исследовании языков различных логико-математических исчислений, естественных языков, для описания отношений между языками различных «уровней» и для характеристики отношений между языками и описываемыми с их помощью предметными областями [118].

Понятие метаязыка гораздо шире, чем понятие терминологии, так как метаязык включает в себя семантические аспекты в более широком смысле, чем предметный язык. В отличие от предметных языков, метаязык является, как правило, некоторым специальным образом созданным языком, не содержащим всякого рода двусмысленностей, препятствующих использованию его в качестве орудия для создания точных высказываний о предметном языке [7]. Более того, метаязык должен быть «не беднее» своего предметного языка и содержать выражения более высоких «логических типов», нежели язык-объект [218]. При невыполнении этих требований возникают семантические парадоксы (антиномии).

Среди используемых на практике метаязыков наиболее распространенными являются металингвистическая форма Бэкуса-Наура (БНФ) [82], которая была разработана для описания языка ALGOL60 и стала математическим фольклором, и синтаксические диаграммы Вирта [43, 366].

Однако метаязык, как и любой язык, также нуждается в описании. Для определения метаязыка, очевидно, требуется метаязык более высокого порядка, что приводит к некоторой иерархической структуре языковых средств. По этой причине семантика формальных систем в настоящее время задается путем определения некоторого универсального метаязыка, который замыкает иерархию, строится на основе конечного множества первичных

семантических категорий и содержит формальный аппарат, с помощью которого и через первичные категории выражается более сложный смысл.

Так как метаязык по своей природе должен быть более выразительным, чем предметный язык, то при его использовании возникли трудности, связанные с необходимостью динамического расширения категориального аппарата в зависимости от того, какие выражения предметного языка встретились при анализе [211].

В итоге, попытки построения универсальных метаязыковых средств столкнулись с проблемой принципиальной непополнимости множества первичных семантических категорий, ранее обнаруженной К. Геделем в арифметике и заключающейся в неполноте и непополнимости ее аксиоматики [166]. Последнее означает, что какое бы множество первичных семантических категорий выбрано не было, на практике найдутся задачи, для описания семантики которых этого множества категорий будет недостаточно.

1.8.2. Содержательное описание семантики

Содержательное описание правил смысловой интерпретации тестов используют при изучении языков. Как правило, сначала при помощи какой-либо формальной грамматики дается определение синтаксиса, а затем, для пояснения семантики, приводятся несколько примеров и небольшой пояснительный текст.

Однако даже содержательное описание приводит к труднопреодолимым трудностям проверки этого описания на полноту и непротиворечивость. Например, для спецификации результатов объектного анализа используется общецелевой язык визуального моделирования UML [203]. Так как язык непрерывно совершенствуется разработчиками, то описание UML стало нетривиальным, поскольку семантика базовых понятий включает в себя целый ряд перекрестных связей с другими понятиями и конструкциями языка. В связи с этим, линейное или последовательное рассмотрение основных конструкций UML стало практически невозможным. В то же время, каждое из представлений модели обладает собственными семантическими особенностями, которые накладывают отпечаток на семантику базовых понятий языка в целом [147].

1.8.3. Формальное описание семантики

Задача формального определения семантики рассматривается теоретиками так же долго, как и задача определения синтаксиса, но до сих пор удовлетворительного решения не найдено. Традиционно определение семантики основывается на выделении некоторой типологии смыслов – категорий значений (семантических категорий, примитивов), через которые и посредством которых на семантическом языке задается более сложный смысл.

В основе описания семантики формальных языков лежит ***принцип композиционности*** Г. Фреге [26, 95], заключающийся в определении семантики целого через семанти-

ку частей, т.е. между семантическими правилами интерпретации и синтаксическими правилами образования выражений языка может быть устновлено соотвествие. По характеру описания семантики можно выделить два наклонения: повелительное, или императивное, представленное операторами (командами, предписаниями), и изъявительное, или декларативное, представленное только описаниями [9].

В частности ГОСТ 34320-1996 рекомендует следующий метод описания семантики формальных языков: «Чтобы приписать смысл (семантику) различным выражениям в языке, необходимо начать с множества неопределенных понятий – примитивов. Другие понятия тогда получают смысл, выводимый из неформальных примитивных понятий с помощью формальных определений. Соответствующие аспекты смысла каждого примитивного понятия формально вводятся посредством задания аксиом, которые считаются истинными. Правила вывода должны сохранять истинность» [89].

Трудности, возникающие на этом пути, прежде всего связаны с многообразием семантических категорий (примитивов), к которым принадлежат выражения формализованных языков. Так, в работе А. Тарского [350] выделено четыре типа языков, классифицируемых в зависимости от того, к каким категориям принадлежат встречающиеся в этих языках переменные:

– языки, в которых все переменные относятся к одной семантической категории, например язык исчисления высказываний с кванторами по пропозициональным переменным;

– языки, в которых число категорий конечно, например одноместное исчисление предикатов с кванторами по предикатным переменным;

– языки, в которых переменные принадлежат к бесконечному числу различных семантических категорий, но порядок этих категорий конечен и не превосходит некоторого фиксированного числа, например исчисление предикатов 2-го порядка;

– языки, содержащие переменные сколь угодно высокого порядка, например язык простой теории типов [353].

Как оказалось, даже в простых случаях не удается избежать неоднозначности в определении принадлежности высказывания к той или иной семантической категории [211]. Уже в языках 2-го типа переменные могут принадлежать более чем одной категории. Это объясняется зависимостью семантической интерпретации предиката не только от числа его аргументов, но и от семантических категорий каждого из них. Последнее приводит к необходимости динамического расширения категориального аппарата в зависимости от того, какие выражения формального языка встретились при анализе.

На данный момент не существует универсального общепринятого подхода для формального описания семантики. Разработано множество различных моделей и методов:

– грамматические модели, основанные на добавлении расширений к грамматике, определяющей этот язык (атрибутные грамматики [126], императивный или операционный метод [346], венский метод [312], W-грамматики [364]);

– аппликативные модели, в которых непосредственно конструируется определение функции, которую вычисляет каждая программа, написанная на этом языке (аксиоматический метод [297], метод денотационной семантики [348], метод функциональной семантики [311]);

– модели спецификаций, в которых описываются отношения между различными функциями языка, и если программа реализует эти отношения, то она корректна по отношению к спецификации (метод спецификаций [200], алгебраический метод [76, 77]).

1.8.4. Семантика языков программирования

Для описания семантики языков программирования упоминаются три наиболее распространенных метода [207]: операционной, продукционной (дедуктивной, аксиоматической) и денотационной (математической) семантики, которые являются фактически единственными потенциально пригодными. Последнее обстоятельство объясняется гибкостью этих методов, потенциальной возможностью расширения набора семантических категорий (математических объектов, команд виртуальной машины), с использованием которых осуществляется описание семантики.

Продукционный метод базируется на исчислении предикатов, где результат вычисления описывается через взаимосвязь переменных до и после применения конструкций языка. Продукционная семантика фактически построена на основе математической логики, ориентирована на человека и предназначена скорее для доказательства правильности программ, чем для формального определения семантики. В качестве примеров продукционных методов описания семантики можно привести аксиоматический метод Хоара [297] и метод индуктивных утверждений Флойда [283].

Денотационный метод основывается на функциональных вычислениях, в которых встроенные операции языка отображаются в однозначные математические объекты, которые затем применяются для описания семантики языковых конструкций [48]. В частности, данный подход иллюстрируется теорией вычислений Скотта, основанной на семантических доменах [336], где сначала перечисляются стандартные и определяемые (конечные) домены, затем, на основе определения конструкторов, задается семантика конструкций языка в виде формулы над доменами и конструкторами. Заметим, что денотационное опи-

сание языка используется для порождения компиляторов, однако ни одного работоспособного компилятора на основе этого метода до сих пор не построено [232].

В ***операционном*** методе операции языка описываются через команды некоторой абстрактной машины. Смысл выполняемых оператором действий выражается в изменении состояния этой машины. Примерами абстрактных машин является SECD-машина [307] и категориальная машина [278]. Подобный метод применялся в 1972 году для описания семантики языка PL/I. Как показала практика, описания, основанные на низкоуровневом моделировании, практически бесполезны, поскольку слишком громоздки и неудобны для восприятия человеком [207].

Ни один из перечисленных методов определения семантики не оказался полезным ни для пользователя, ни для разработчика языка. Операционный метод достаточно удобный для реализации и может быть полезен разработчику, но для пользователя этот метод не имеет большого значения, так как порождаемые им описания содержат слишком много ненужных ему подробностей. Разработчик вряд ли сможет руководствоваться функциональным и денотационным методами, а для пользователя они, как правило, оказываются слишком сложными, чтобы их можно было использовать непосредственно. Пользователю легче понять аксиоматический метод, но при попытке составить полное определение языка он порождает чрезвычайно сложное описание, а для разработчика этот метод и вовсе непригоден [200].

Таким образом, как в области описания дискретной обработки данных на языках высокого уровня в рамках концептуально-онтологического подхода, так и при низкоуровневом описании задач в рамках формально-логического подхода имеются нерешенные проблемы, связанные с методологией и технологией декомпозиции предметной области и с синтезом эффективных математических моделей, формализующих результаты этой декомпозиции. На сегодняшний день практически отсутствует единая концептуально-методологическая база, включающая достаточно универсальные теоретическое обобщение и математическую модель, на которых возможно было бы основать схему практической реализации, внедрения и сопровождения информационных систем, основанных на обработке данных [105].

Выводы к Главе 1

1. Выявлено два противоположных и дополняющих друг друга подхода к синтезу математических моделей дискретной обработки данных – формально-логический и концептуально-онтологический. Формально-логический подход реализуется при декомпозиции дискретных функций и используется, когда содержательные представления о решаемой задаче отсутствуют или опущены, а размерность задачи невелика. При концептуально-онтологическом подходе, в противоположность формально-логическому, для получения математических моделей дискретной обработки данных используется содержательная интерпретация решаемых задач.

2. Анализ современного состояния формально-логических методов синтеза математических моделей и основанных на них технологий дискретной обработки данных показал, что нерешенной оказалась задача поиска эффективных математических моделей для реализации дискретной обработки данных на современных и перспективных вычислительных средствах и установление критериев этой эффективности.

3. При синтезе математических моделей дискретной обработки данных в рамках формально-логического подхода имеется потребность в разработке теории и методики оценки абсолютной эффективности результатов моделирования, позволяющих по виду модели, не прибегая к синтезу альтернативных вариантов, определить ее эффективность.

4. При использовании концептуально-онтологического подхода для синтеза высокоуровневых моделей дискретной обработки данных на основе языков, технологий и методологий программирования не решена задача получения декомпозиционных схем предметной области и качественных (точных, надежных, обозримых) ее формальных спецификаций, позволяющих эффективно реализовать дискретную обработку данных.

5. Для синтеза эффективных математических моделей в рамках концептуально-онтологического подхода необходимо решить задачу определения семантики формальных языков, заключающуюся в создании методологии анализа предметной области и технологии формализации полученных при анализе результатов, позволяющих создавать полные и семантически прозрачные формальные спецификации.

Глава 2.
Понятийный анализ

Настоящая глава посвящена методологии изучения и описания предметной области – понятийному анализу. Основная цель понятийного анализа состоит в получении таких декомпозиционных схем предметной области, которые хотя и сформулированы в рамках содержательных представлений, однако обладают формальной строгостью и точностью, достаточной для прямого использования полученного высокоуровневого описания для низкоуровневой реализации дискретной обработки данных. Тем самым обеспечивается получение первичных формальных спецификаций предметной области, используемых в рамках контекстной технологии для реализации дискретной обработки данных путем детализации этого описания, возможно иерархического. Подробное описание контекстной технологии приведено в Главе 3.

2.1. Содержательная постановка задачи

Главной содержательной особенностью словесно-логической традиции в описании понятий является его понимание как некоторого мысленного «слепка» множества сущностей, обобщенно репрезентирующим его в сознании человека в виде определенно организованной совокупности существенных признаков. Последнее приводит к тому, что в рамках этой традиции сложную организацию предметного содержания понятия тщетно пытаются описать уже довольно давно. Причины неудач, видимо, кроются в том, что понятие является не просто статической репрезентацией реальности, а сложно устроенным когнитивным феноменом, позволяющим изменять (перестраивать) свои собственные репрезентации в зависимости от познавательных целей субъекта [131].

Однако, как бы то ни было, внешняя репрезентация понятия в словесно-логической форме видится единственной и принципиально необходимой формой представления и обработки знаний. По этой причине описываемый далее проблемный подход и пополняемое множество форм многоаспектного выражения понятий является некоторым неизбежным компромиссом между понятием как когнитивным феноменом и понятием как сложно организованной совокупностью существенных признаков, используемой для внешней репрезентации понятия в словесно-логической форме.

Для формальной спецификации предметной области будем использовать две формальные системы. Первую формальную систему – исчисление понятий, применим для выражения инвариантных свойств предметных областей. К инвариантным свойствам отнесем свойства понятийных структур предметных областей. Вторую формальную систему

– специализированный предметный язык, будем строить для каждой предметной области и использовать для описания специфических ее свойств.

Понятия, выявленные в процессе анализа предметной области, условно разделим на две группы: терминальные, или сигнификативные, выражаемые последовательностью знаков терминального алфавита специализированного предметного языка, и нетерминальные, или денотационные, соответствующие нетерминальным знаками порождающей грамматики этого языка. Разделение понятий на денотационные и сигнификативные осуществим с учетом некоторой фиксированной проблематики, задающей класс задач, для которых определяется специализированный язык.

На основе выявления способов абстрагирования денотационных понятий построим понятийную структуру предметной области, где под абстракцией понимается одно из четырех отображений вида $N^i \to N$, где N^i – декартова степень i множества понятий N, которые соответствуют четырем фундаментальным способам образования понятий: обобщению, типизации, агрегации и ассоциации. Для каждой такой абстракции дадим формальное и семантически прозрачное (инвариантное) определение, не требующее предметной интерпретации, как это имеет место в других концептуальных моделях, где используется множество связей между понятиями, несущими различную семантическую нагрузку.

Выявленные в процессе анализа предметной области денотационные понятия включим в множество понятий специализированного предметного языка, а найденные декомпозиционные схемы преобразуем в его языковые конструкции. Языковые конструкции будем рассматривать как формы выражения денотационных понятий в тексте и задавать последовательностью знаков денотационных и сигнификативных понятий.

В заключении исследуем разработанный формализм высокоуровневой спецификации предметной области и определим его выразительные возможности.

Основные результаты настоящей главы опубликованы в работах [63, 69].

2.2. Основные определения

Проблемную область определим как совокупность предметной области и решаемых в ней задач (проблем).

Представления, следующие их содержательной постановки решаемых задач и определяющие один из возможных аспектов рассмотрения и декомпозиции предметной области, будем называть ***проблематикой***. В отличие от прагматики, определяющей отношение субъекта к тексту [316], проблематику будем рассматривать как целевую установку

(целеполагание) субъекта при декомпозиции предметной области на значимые сущности.[12]

Под ***предметной областью*** будем понимать фрагмент реальной (мыслимой) действительности, представляемый некоторой совокупностью принадлежащих ему сущностей.

При декомпозиции предметной области в качестве одного из основных допущений будем использовать предположение о том, что мир состоит из относительно устойчивых взаимосвязанных сущностей, которые могут быть выделены и представлены (объективированы) в виде знаковой системы.

Сущность определим как устойчивое и уникальное представление о выделенной части предметной области. Сущность воспринимается в виде своих признаков. Известна следующая классификация сущностей [192]: предмет, свойство (атрибут), состояние, процесс, событие, оценка, модификатор, квантификатор, модальность.

Признак – именованная сущность, характеризуемая множеством своих проявлений (значений) и имеющая проблемную интерпретацию (семантическую роль). Признаки по своей сути являются элементарными сущностями, с точностью до которых производится описание предметной области. Выделение признаков в предметной области осуществляется под углом зрения той проблематики, которая зафиксирована проблемной областью. Иными словами, одна и та же сущность, будучи использована для решения различных проблем, может характеризоваться различными признаками.

Понятие – это форма мышления, отражающая совокупность сущностей в их существенных признаках.[13] ***Понятие*** определим как именованное множество сущностей, объ-

---

[12] Например, в [89] проблемная область рассматривается как состоящая из сущностей, классификация которых основывается на сходстве и учитывает характеристики, общие для нескольких сущностей. Выбор характеристик для группировки сущностей в классы произволен и осуществляется прагматически, в зависимости от целей проблемной области.

[13] В рамках логической традиции, занимающейся способами понимания основных структурных единиц словесно-логического мышления, необъясненным остается факт сложной организация предметного содержания понятия, которую тщетно пытаются описать формальными средствами, Последнее является следствием трактовки понятия как мысленного «слепка» предмета, обобщенного репрезентирующего его в сознании человека в виде организованной совокупности существенных признаков. На деле оказывается, что понятие является не просто репрезентацией реальности, а «сложно устроенным культурным средством, позволяющим управлять своими собственными мыслительными операциями» [131]. Таким образом, неизбежный недостаток логического подхода состоит в том, что в нем внутреннее содержание и строение понятия обсуждается на языке свойств сущностей, при полном замалчивании вопроса о том, что именно позволяет субъекту мышления выделять в сущности ее отличительные признаки. Второй недостаток логического подхода состоит в том, что понятие рассматривается в форме законченного знания о предмете при полной невозможности описания понятия как еще не завершенного акта мышления.

единенных на основе общности своих признаков. Понятия будем именовать, и задавать схемой, интенсионалом и экстенсионалом.

Имя, или знаковое представление понятия, будем рассматривать как языковую единицу, отражающую некоторый смысл в семантическом плане и некоторую конкретную сущность в плане синтаксическом.

Схему понятия зададим набором признаков, на которых понятие определено. Признаки в этом случае будем интерпретировать как некоторые элементарные понятия, на которых определяется схема. Последние, возможно, являются составными и строятся на основе других более простых понятий. Таким образом, понятие в общем случае может быть определено как именованное множество других понятий, имеющих подсхему, принадлежащую схеме образуемого понятия.

Интенсионал, или содержание понятия, представим набором значений взаимосвязанных признаков, позволяющим отличать сущности, принадлежащие понятию, от других сущностей предметной области. Тем самым интенсионал понятия косвенно задает тот смысл, который вкладывается в это понятие.

Экстенсионал, или объем понятия, будем задавать множеством сущностей, принадлежащих понятию.

Пример 2.1. Пусть предметная область задана в виде сущностей, обозначенных 1, 2, …, 99, – некоторой последовательности натуральных чисел, заданных в десятичной системе счисления.

Исходя из проблемной ситуации зададим исходные (терминальные) понятия-признаки, существенные для решения некоторого класса прикладных задач, например, для выбора сущностей предметной области по ряду критериев.

Пусть признаками будут «делитель», «количество», «сумма», «произведение». Семантические роли введенных признаков определим соответственно как «делители числа», «количество цифр числа», «сумма цифр числа», «произведение цифр числа», а множества значений – {1, 2, ..., 99}, {1, 2}, {1, 2, ..., 18}, {1, 2, ..., 81}.

Зададим понятие «двузначное кратное 5». Схемой данного понятия будет набор из двух признаков («количество», «делитель»),[14] интенсионал понятия – множество из одного набора значений (2, 5), а экстенсионал – множество сущностей {10, 15, 20, ..., 90, 95}. ♦

2.2.1. Признаки и сущности

Сущности, составляющие экстенсионал понятия различаются с помощью признаков. Так как каждый признак можно рассматривать как некоторое специализированное

[14] Здесь и далее фигурными скобкам будем обозначать множества, а в круглых скобках задавать наборы элементов, которые, в отличие от множеств, могут содержать повторяющиеся элементы.

(первичное) понятие, то множество допустимых значений понятия-признака образует его экстенсионал, или ***домен***, а ***семантическая роль*** понятия-признака представляется его интенсионалом.

В итоге, интенсионал признака – это те минимальные смысловые единицы, на которых строится описание предметной области, а экстенсионал признака задает элементарные синтаксические единицы (термы), которые используются для выражения составных понятий через простые.

Заметим, что деление понятий на простые и составные является относительным и, по большей части, следует из онтологических представлений познающего субъекта. Часть таких представлений определяется прагматикой и выводится из проблематики решаемых задач.

Пример 2.2. Рассмотрим признак «делитель», введенный в примере 2.1. Для понятия-признака «делитель» схема («делитель») тривиальна, интенсионал {(1), (2), (3), (4), …, (99)} задан набором допустимых значений признака с присвоением ему некоторого первичного смысла – «делитель числа», а экстенсионал {1, 2, 3, …, 99} рассматривается как множество сущностей, характеризуемых признаком. ♦

В предельном случае, когда у понятия единичный объем, имеем некоторую именованную сущность. Следовательно, имена могут использоваться не только для обозначения понятий, но и для обозначения единичных сущностей. Таким образом, под сущностью будем понимать понятие, имеющее единичный объем (терминальное понятие).

Пример 2.3. Рассмотрим предметную область из примера 2.1. Для первой сущности, обозначенной как 1, зададим имя «единица» для определяемого ею понятия. Схема понятия – («единица»), как и у признаков, тривиальна и замкнута. Интенсионалу {(1)} присвоим семантическую роль, выраженную, например, словами «начальная сущность». Очевидно, экстенсионал {1} понятия-сущности тривиален, так как равен самой сущности (замыкается на определяемую сущность). ♦

На основе вышеизложенного делаем вывод, что понятие обладает свойством фрактальности: для его определения используются сущности, рассматриваемые как единичные понятия, и признаки, являющиеся простыми понятиями, причем разделение понятий на сущности и признаки задается активной проблематикой. В итоге, понятие следует рассматривать как общую категорию, а сущность и признак – как ее необходимые частные случаи.

2.2.2. Теоретико-множественный формализм

Используем введенные ранее определения для теоретико-множественной формализации понятий в рамках аппарата, описанного в [158].

Определение 2.1. Формальным понятием *N называется тройка*

$$N = \begin{cases} \text{shm}\, N = (N_0, N_1, \ldots, N_{n-1}); \\ \text{int}\, N = \{(V_0^j, V_1^j, \ldots, V_{n-1}^j) \mid j = \overline{0, m-1}\}; \\ \text{ext}\, N = \{E_0, E_1, \ldots, E_{u-1}\}. \end{cases} \quad (2.2)$$

состоящая из схемы $\text{shm}\, N$*, интенсионала* $\text{int}\, N$ *и экстенсионала* $\text{ext}\, N$*, где* N *– имя понятия,* N_i *– понятия-признаки,* $i = \overline{0, n-1}$*,* $(V_0^j, V_1^j, \ldots, V_{n-1}^j)$ *– наборы значений признаков, составляющие интенсионал определяемого понятия,* $j = \overline{0, m-1}$*,* E_k *– сущности, принадлежащие понятию* N*,* $k = \overline{0, u-1}$.

Если некоторые исходные понятия N_i служат признаками для определения другого понятия N, то упорядоченные наборы $(V_0^j, V_1^j, \ldots, V_{n-1}^j)$ следует интерпретировать как состоящие из сущностей исходных понятий, представленных в одной из возможных синтаксических форм своего выражения.

Пример 2.4. Представим понятия «единица», «делитель», «количество», «сумма», «произведение» и «двузначное кратное 5» из примеров 2.1 и 2.3 в теоретико-множественной форме:

$$\text{«единица»} = \begin{cases} \text{shm}\, I = (I); \\ \text{int}\, I = \{(1)\}; \\ \text{ext}\, I = \{1\}, \end{cases} \qquad \text{«количество»} = \begin{cases} \text{shm}\, D = (D); \\ \text{int}\, D = \{(1), (2)\}; \\ \text{ext}\, D = \{1, 2, \ldots, 99\}, \end{cases}$$

$$\text{«делитель»} = \begin{cases} \text{shm}\, R = (R); \\ \text{int}\, R = \{(1), (2), ..., (99)\}; \\ \text{ext}\, R = \{1, 2, 3, \ldots, 99\}, \end{cases} \qquad \text{«сумма»} = \begin{cases} \text{shm}\, S = (S); \\ \text{int}\, S = \{(1), (2), ..., (18)\}; \\ \text{ext}\, S = \{1, 2, \ldots, 99\}, \end{cases}$$

$$\text{«произведение»} = \begin{cases} \text{shm}\, M = (M); \\ \text{int}\, M = \{(1), (2), ..., (81)\}; \\ \text{ext}\, M = \{1, 2, \ldots, 81\}, \end{cases}$$

$$\text{«двузначное кратное 5»} = \begin{cases} \text{shm}\, C = (D, R); \\ \text{int}\, C = \{(2, 5)\}; \\ \text{ext}\, C = \{10, 15, \ldots, 95\}, \end{cases}$$

где I – обозначение понятия-сущности «единица», D, R, S и M – обозначения понятий-признаков «количество», «делитель», «сумма» и «произведение». ♦

2.2.3. Логическая форма

Помимо теоретико-множественной формы для описания интенсионалов понятий может быть использована логическая форма [81]. В логической нотации интенсионал в (2.2) выражается формулой f исчисления высказываний

$$f(E, V_i, P_j),$$

в которую входит свободная предметная переменная E, обозначающая сущность, проверяемую на принадлежность понятию, предметные константы V_i, интерпретируемые как сущности-признаки или их множества, и предикаты P_j, выражающие наличие у сущности значений соответствующих признаков.

Интенсионал простого понятия-признака N представляется логическим выражением

$$N(E, \mathrm{dom}N) \to N(E),$$

или, если домен признака N задан в виде нескольких множеств V_i,

$$\bigvee_{i=0}^{n-1} N(E, V_i) \to N(E),$$

где N и V_i – имя и множества сущностей понятия-признака, а $N(E, V_i)$ – истинный предикат, если сущность E принадлежит множеству V_i, и ложный – в противном случае.

Составные понятия описываются логическим выражение общего вида:

$$\bigvee_{j=0}^{m-1} \&_{i=0}^{n-1} N_i(E, V_{ij}) = \bigvee_{j=0}^{m-1} N_j(E) \to N(E). \qquad (2.3)$$

где N_i и V_{ij} – понятия-признаки и заданные для них множества сущностей, а $N_i(E, V_{ij})$ – предикат принадлежности признака N_i понятия-сущности E множеству V_{ij}.

Фактически выражение (2.3) определяет понятие N через m альтернативных понятий N_j, которые заданы на общем множестве из n признаков и выражаются конъюнкциями своих предикатов. Значения признаков V_{ij} ($j = \overline{0, m-1}$) могут принимать произвольные множества значений из доменов $\mathrm{dom}\, N_i$, соответствующих альтернативным понятиям-признакам N_j.

Пример 2.5. Представим понятие N «двузначное кратное 5 или 6» в логической форме:

$$D(E,2) \,\&\, R(E,5) \vee D(E,2) \,\&\, R(E,6) \to N(E),$$

где D и R – обозначения предикатов, соответствующих понятиям-признакам «количество» и «делитель», E – вхождение сущности, проверяемой на принадлежность понятию N. ♦

Очевидно, для конструктивного определения выполнимости предикатов необходимо, чтобы множества сущностей-признаков V_{ij} были разрешимы.

2.2.4. Грамматическая форма

Для определения интенсионала понятий воспользуется формализмом контекстно-свободных грамматик. ***Контекстно-свободной грамматикой*** называется формальная

система $< T, W, P, I >$, состоящая из терминального T и нетерминального алфавита W, правил вывода P вида $N \to \alpha$ и аксиомы I – начального нетерминального знака, где N – некоторый знак нетерминального алфавита, а α – конечная последовательность терминальных и нетерминальных знаков [205].

Интенсионалы будем задавать множеством правил контекстно-свободной грамматики. Терминальный алфавит предполагается достаточным для определения любой формы выражения понятия в тексте, а нетерминальный алфавит, очевидно, совпадает с множеством понятий предметной области. Правила грамматики запишем в виде $N \to \alpha$, где N – определяемое понятие. В том случае, когда понятие N подразумевается, левую часть правила будем опускать.

Интенсионал понятия N, заданный в грамматической форме, будем использовать как для порождения экстенсионала N, так и для распознавания принадлежности ему некоторой сущности. В итоге получаем, что в грамматической форме использованию подлежит как продукционная, так и редукционная форма правил вывода. Продукционная форма используется при конструировании правил выражения понятий в тексте, а редукционная – при идентификации (распознавания) понятия, выраженного некоторым фрагментом текста.

При записи правил вывода терминальные строки будем включать в одинарные кавычки, а терминальные шаблоны, заданные на языке регулярных выражений [369], – в двойные.

Пример 2.6. Представим понятие «двузначное кратное 5» в грамматической форме:

$$\text{«двузначное кратное 5»} = \begin{cases} \text{int}\, N = \{\text{'5'}, \text{"[1-9][05]"}\}; \\ \text{ext}\, N = \{5, 10, \ldots, 95\}, \end{cases}$$

где интенсионал понятия N задан множеством из двух правил $N \to$'5' и $N \to$"[1–9][05]". Первое правило утверждает, что терминальный знак 5 является выражением понятия «двузначное кратное 5». Второе правило задает терминальный шаблон на языке регулярных выражений. В этом случае определяемое понятие выражается в виде двух знаков, первый из которых от 1 до 9, а второй 0 или 5. ♦

Важной особенностью грамматической формы является возможность использования для определения понятий не только знаков терминального алфавита, но и нетерминальных знаков. В этом случае нетерминальные знаки интерпретируются как некоторые понятия и соответствуют множеству строк в терминальном алфавите, которыми эти понятия могут быть выражены.

Пример 2.7. Представим понятие «отрицательное двузначное» в грамматической форме при условии, что ранее определено понятие N_0 – «десятичная цифра» и понятие N_1 – «десятичная цифра кроме нуля»:

$$\text{«отрицательное двузначное»} = \begin{cases} \text{int}\, N = \{'-' N_1 N_0\}; \\ \text{ext}\, N = \{-10, -11, \ldots, -99\}. \end{cases} \blacklozenge$$

Заметим, что в грамматической форме задается только интенсионал понятия, который используется для конструктивного описания экстенсионала. Очевидно, для полного определения понятия необходимо предусмотреть средства для описания схем.

2.2.5. Модель понятия

Современная лингвистика и семиотика рассматривает «знак» как термин, связывающий «понятие» и его «значение», где под ***знаком*** понимается некий условный объект – чувственно воспринимаемый предмет, действие или явление – представляющий при определенных условиях некоторое смысловое значение, которое выступает в качестве указания, обозначения или представителя другого предмета, действия, явления или абстрактного понятия [178]. Из определения знака вытекает его важнейшее свойство: будучи некоторым материальным объектом, знак служит для обозначения чего-либо другого.

Для раскрытия содержания знака используется модель, выраженная семантическим треугольником, или треугольник Фреге (рис. 2.1), где знак рассматривается как имеющий, по крайней мере, два типа значений. Термины обозначающие нижние вершины треугольника разными исследователями называются по разному [144]: Ч. Пирс [187] различает «объект» и «интерпретанту»; Г. Фреге [229] – «значение» и «смысл»; Ч. Моррис [169] и А. Черч [242] – «денотат» и «десигнат», Р. Карнап [119] – «экстенсионал» и «интенсионал» и т.д.

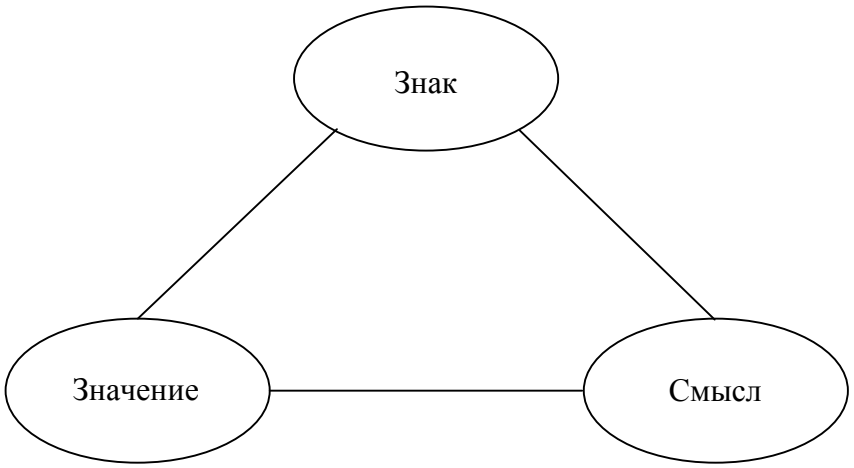

Рис. 2.1. Треугольник Фреге

В логике и семиотике вершина Значение, или обозначаемое, раскрывается как единичная сущность, обозначаемая Знаком-именем (десигнатом), или класс сущностей, обозначаемых общим для них Знаком-понятием (денотатом, метазнаком) [179]. Другой характеристикой Знака является его Смысл, соответствующий некоторому мысленному содержанию, возникающему у интерпретатора при восприятии Знака.

В прикладной семиотике для раскрытия различных способов представления Знака и тех последствий, к которым приводит его восприятие интерпретатором, используются еще две вершины, называемые Синтаксисом и Прагматикой (рис. 2.2) [196].

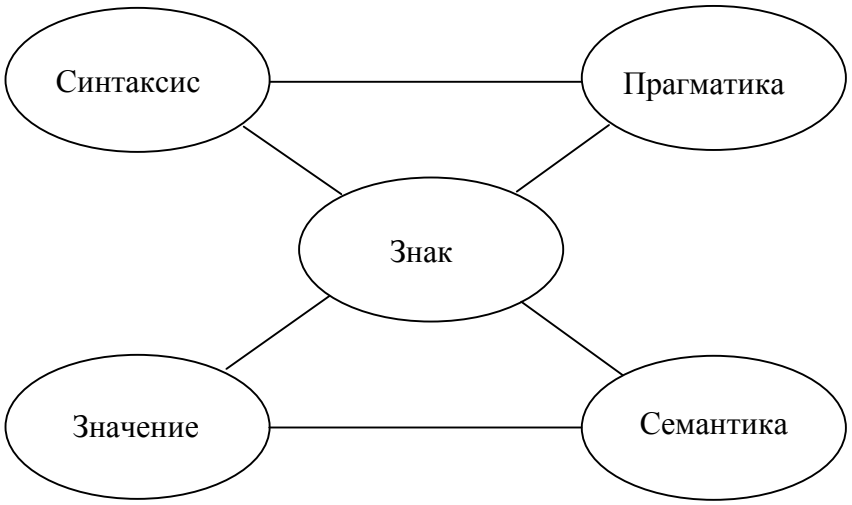

Рис. 2.2. Модель знака в прикладной семиотике

Введение Синтаксиса и Прагматики оказалось взаимно обусловлено, так как позволяет представлять Знак различным образом, в зависимости от выражаемой им Прагматики. В итоге Знак раскрывается через Синтаксис – способ выражения или кодирования Знака, Значение – денотат или десигнат Знака, Семантику – соответствия между формой выражения Знака и его Смыслом, и Прагматику – действия, связанные со Знаком.

Общим недостатком известных моделей организации знания (когнитивных моделей), является то обстоятельство, что при их построении структура репрезентации знания выбирается более или менее произвольно, без должного учета конкретной обстановки. Исходным мотивом для введения пропозиционального описания было желание свести множество поверхностно различных высказываний к более простому набору базовых семантических элементов. Однако всякое высказывание может быть множеством способов интерпретировано и представлено с помощью других высказываний. Поэтому опыт создания современных когнитивных моделей показывает, что их адекватность ситуации определяется не столько мощностью используемого формального аппарата, сколько учетом особенностей организации повседневных форм деятельности человека [41, т. 2, с. 69].

Исходя из необходимости учета проблемной ориентации любой формы репрезентации знаний разработаем адекватную решаемым задачам модель понятия. Для этого будем предполагать, что образование или выявление (специализация, детализация) уже существующих понятий происходит в процессе изучения предметной области. При этом под углом зрения некоторой проблематики выделяются сущности, которые имеют или которым приписываются некоторые имена, т.е. происходит их ***означивание***. Далее множество выявленных сущностей подвергается анализу на предмет установления их сходства и различия. Сходные сущности группируются, в результате чего происходит образование понятий, или наполнение уже имеющихся понятия проблемным содержанием.

Понятие, задаваемое как некоторый знак, расположим в верхней вершине треугольника, а нижние вершины треугольника обозначим соответственно как Имя и Сущность (рис. 2.3). Вершины Имя и Сущность являются крайними полюсами в процессе выявления (узнавания) понятий в предметной области. Сущность в указанном контексте может интерпретироваться как естественный знак, наиболее близкий к обозначаемому понятию. В свою очередь Имя, как противоположный полюс, интерпретируется как знак, который наиболее отдаленный от обозначаемого, но все еще связанный с ним хотя бы тем, что отражает некоторые его черты.

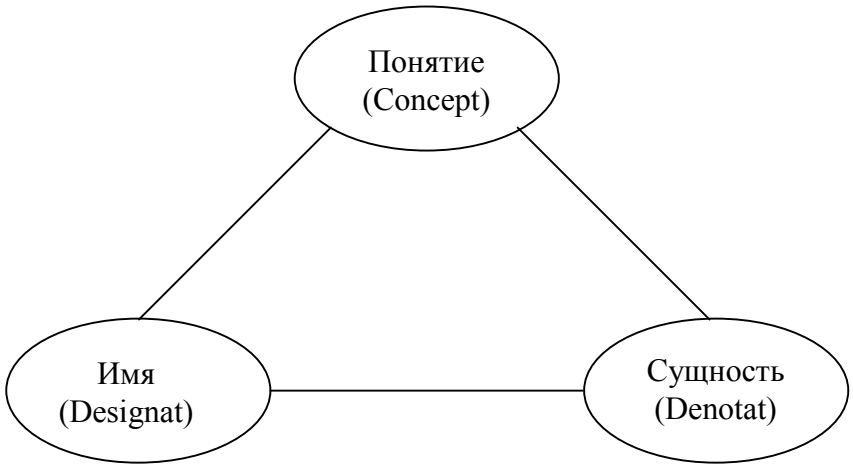

Рис. 2.3. Образование понятий

Таким образом, вершина Сущность определяет денотационную форму выражения понятия (Denotat), в то время как вершина Имя задает сигнификативную форму (Designat). Под ***денотационной*** формой означивания здесь понимается соотнесение сущности со знаком уже имеющегося понятия, а под ***сигнификативной*** – с присваиванием сущности некоторого индивидуального имени. С точки зрения синтаксиса между указанными крайними точками могут существовать множество промежуточных форм, сочетающих в себе различны степени денотационного и сигнификативного выражения. Заметим, что в чистом виде сигнификативная форма находит свою реализацию в терминальных понятиях фор-

мальных языков, в то время как денотационная – в нетерминальных. Однако другие понятия, для выражения которых создан формальный язык, требуют сочетания этих форм.

В итоге получаем треугольник «Имя-Сущность-Понятие», объясняющий образование нового или выражение (детализация, конкретизация) уже существующего понятия в некоторой проблемной области.

С другой стороны, ранее образованное понятие используется в проблемной области не в абсолютном, а в некотором относительном смысле, выявляемом при учете активной проблематики. Поэтому имеется часть признаков понятия, которая подвергается изменению. Совокупность таких признаков традиционно называется ***прагматикой*** [197].

Однако выявление прагматики происходит под влиянием некоторой точки зрения на предметную область: одна и та же сущность, будучи всеобъемлющей объективной реальностью, представляется различным образом в зависимости от класса решаемых задач, объединенных единой проблематикой. Если предположить, что понятие с точки зрения своего содержания есть вместилище всех своих смыслов (сложно устроенный когнитивный феномен, позволяющий в зависимости от решаемой задачи видоизменять свою репрезентацию), то задание проблематики конкретизирует семантику понятия до его прагматики, рассматриваемой как частный случай интерпретации этого понятия.

Следовательно, Прагматика характеризуется тем, что определяет смысл приписываемый понятию наиболее конкретно. В то время как Семантика выражает предельно общий (абстрактный) смысл, связанный с понятием вообще. Между указанными полюсами имеется множество промежуточных форм выражения содержания понятия, различающихся долей абстрактного и конкретного (общего и частного). В итоге получаем второй треугольник «Прагматика-Семантика-Проблематика», объясняющий интерпретацию уже имеющегося понятия в некоторой проблемной области (рис. 2.4).

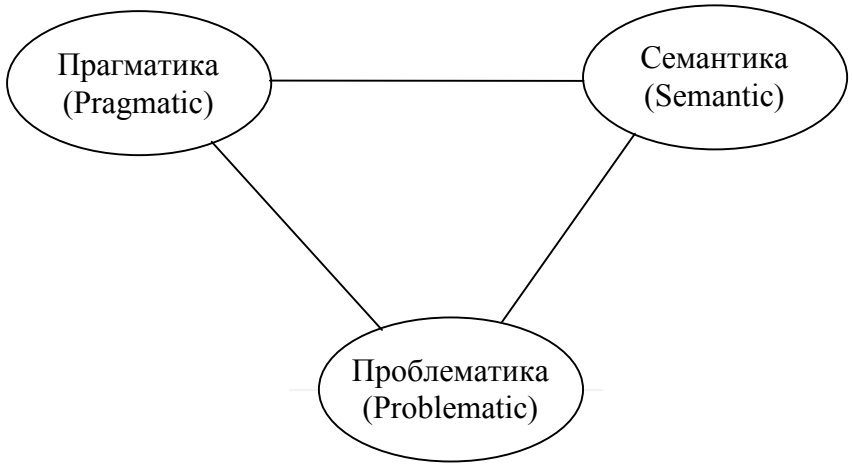

Рис. 2.4. Интерпретация понятия

Соединяя полученные треугольники на одном рисунке, имеем шестиугольник (рис. 2.5), состоящий из верхней триады, объясняющей образование понятия, и нижней триады, определяющей интерпретацию уже образованного понятия.

Так как Проблематика является началом, позволяющим выявить в проблемной области значимые сущности, которые во всей своей совокупности (при различных проблематиках) образуют полный объем некоторого понятия, то переупорядочим взаимосвязи модели на рис. 2.5 и установим между вершинами Проблематика, Имя и Сущность связь в виде нижнего треугольника.

С другой стороны, уже существующее понятие раскрывается через свою Семантику и Прагматику, где Семантика есть общее содержание понятия, а Прагматика – его возможная конкретизация. Следовательно, между вершинами Понятие, Прагматика и Семантика имеется связь, аналогичная рассмотренной выше, которая представляется вторым треугольником.

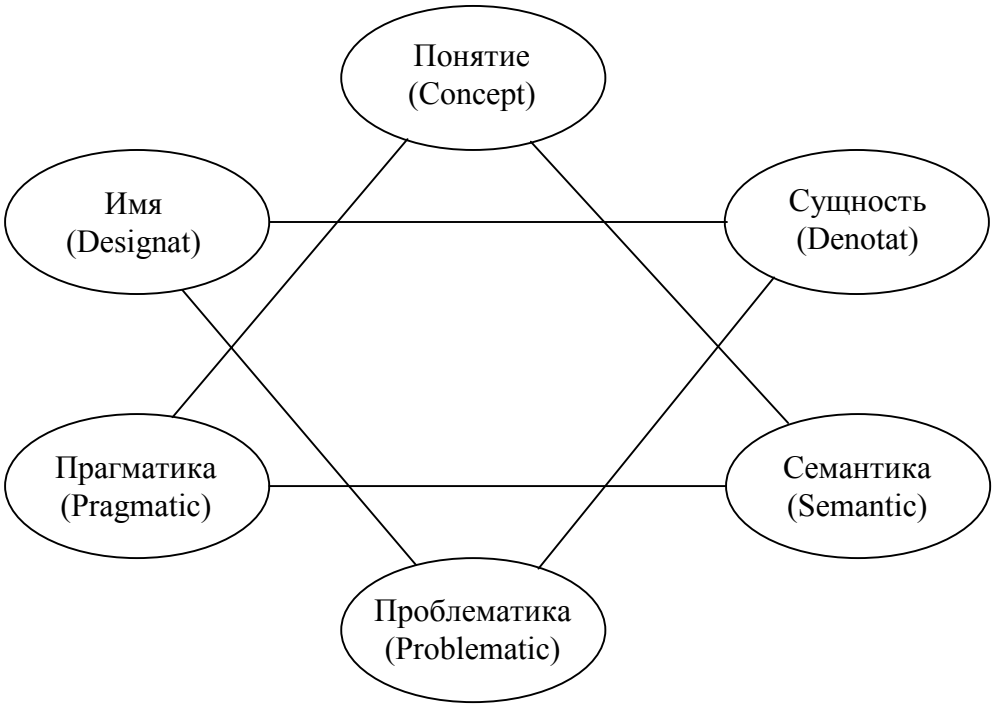

Рис. 2.5. Структура понятия

Таким образом, Имя-Сущность являются крайними полюсами в процессе выявления значимых сущностей в проблемной области, причем форма этого выявления полностью определяется активной Проблематикой. В другой паре Прагматика-Семантика, Прагматика, как полюс, соответствующий Имени, характеризуется тем, что определяет смысл Понятия наиболее конкретно, а Семантика как полюс, соответствующий Сущности, выражает общий (неотчуждаемый) его смысл.

Может оказаться, что описанная и представленная на рис. 2.5 модель понятия в иной проблемной области (при другой постановке задачи) будет непригодной. Однако поиск новых моделей осуществим в рамках описанной модели – путем фиксации новой проблематики, выделения сущностей в проблемной области и их означивания, образования новых или проблематизации уже сформированных понятий. Завершается построение новой модели установлением взаимосвязи найденных понятий и определения содержательной интерпретации этих связей.

2.3. Абстрагирование понятий

Понятия образуются при абстрагировании. ***Абстракция*** (лат. abstractio – отвлечение) определяется как один из основных процессов умственной деятельности человека, позволяющий мысленно вычленить и превратить в самостоятельный объект рассмотрения отдельные свойства, стороны, элементы или состояния предмета [23].

Следует различать уровни абстрагирования: абстрагирование, приводящее к образование простых понятий, и абстрагирование, результатом которого является образование составных понятий.[15]

При образовании простых понятий абстракция рассматривается как некоторое сознательное неведение, позволяющее сосредоточиться на одной стороне сущности и игнорировать другие ее стороны. В этом смысле одним из проявлений абстракции является соотнесение сущности с уже имеющимся понятием, что соответствует денотационной форме ее выражения. Вторым проявлением абстракции есть обозначение понятия некоторым знаком (именем), без раскрытия признаков, на которых понятие определено. Этот вид абстракции используется при сигнификативном выражении понятия.

Более сложные формы абстракции используется при образовании составных, или абстрактных, понятий [158]. В этом случае ***абстрагирование*** рассматривается как форма мышления, при которой образуются новые понятия на основе выделения существенных и несущественных, общих и различающихся признаков абстрагируемых понятий.

---

[15] В логике понятие определяется как мысль, отражающая в обобщенной форме предметы и явления действительности, а также существенные связи между ними посредством фиксации общих и специфических признаков [213]. Такие понятия называются первичными, соотносятся с чувственно воспринимаемыми предметами и имеют наглядно-образный характер. С умножением потребностей человека и усложнением видов его деятельности появились более отвлеченные понятия, непосредственно не связанные с чувственным отражением, но, вместе с тем, являющиеся более близкими к реальности в смысле отражения ее сущности. В итоге считается, что понятия образуются не только через сравнение наглядных образов, но и путем применения логических приемов: анализа, синтеза, абстрагирования, индукции, дедукции, аналогии, идеализации и т.д.

Выделение ***существенных*** и ***несущественных*** признаков основано на способности различать и сравнивать. Последнее осуществляется с учетом индивидуальности познающего субъекта и под углом зрения той проблемы, которая послужила стимулом для процесса познания. Заметим, что при выявлении существенных признаков у сущностей происходит ***взаимная переориентировка*** уже сложившейся системы понятий путем их разделения на сущности и признаки, через которые возможно определение уже имеющихся понятий или образование новых. Более того, проблематика позволяет разделить признаки и взаимосвязи сущностей на существенные и несущественные. Очевидно, при изменении проблематики и это разделение будет другим.

После выявления существенных признаков наступает следующий этап, заключающийся в соотнесении сущностей между собой путем выявления их ***общих*** и ***различающихся*** признаков и установлению между сущностями отношений независимости, дифференциации и интеграции. Используя графическую форму, предложенную в [342], взаимосвязь понятий представим диаграммами (рис. 2.6).

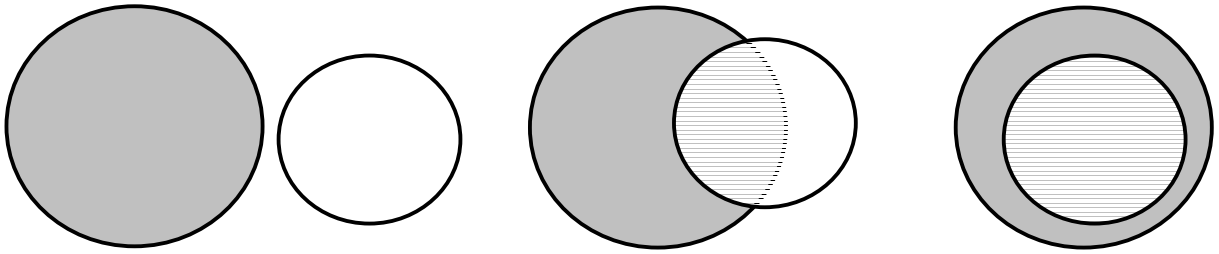

Рис. 2.6. Пространство признаков

Понятия ***независимы***, если признаки, на которых они определены, не пересекаются (а). Если у двух понятий имеется общее подмножество признаков, то наблюдается ***дифференциация*** (б). Если все признаки одного понятий являются признаками другой, то происходит их ***интеграция*** (в).

Известны следующие способы абстрагирования понятий [81]:

– обобщение-специализация;

– типизация-конкретизация;

– агрегация-декомпозиция;

– ассоциация-индивидуализация.

Обобщение и типизация и обратные им абстракции специализации и конкретизации выражают общность понятий, проявляющуюся при дифференциации. Ассоциация и агрегация и противоположные им индивидуализация и декомпозиция раскрывают интеграцию понятий.

В математике абстракция используется для определений понятий через отношение равенства. Отношение равенства, заданное на некотором исходном множестве элементов, разбивает это множество на попарно непересекающиеся классы равных (в данном отношении) элементов. Указанные классы называются классами абстракции данного отношения, а множество этих классов – фактормножеством исходного множества по данному отношению. В этом случае, если выделен класс в каком-либо смысле равных элементов, то тем самым определён и «абстрактный» элемент этого класса, поскольку с точки зрения целей, определяющих данное отношение равенства, каждый «конкретный» элемент исходного множества понимается в качестве «абстрактного» элемента – носителя свойства, общего всем элементам данного класса абстракции. Например, понятие непрерывной функции есть один из классов эквивалентности, порождаемый разбиением множества всех функций отношением непрерывности. С другой стороны, попарно непересекающиеся классы рассматриваются как результат разбиение исходного множества на подмножества и, тем самым, выявляется структура исходного множества, которая порождается заданным отношением равенства. Именно так в математике формируются любые абстрактные понятия. Отсюда следует, что основными видами абстракции в математике являются типизация и агрегация.

В свою очередь в информатике считается, что фундаментальными абстракциями являются обобщение и агрегация, для которых найден соответствующий теоретико-множественный формализм [158]. Однако в психологии к фундаментальным абстракциям относят обобщение и ассоциацию. Так в [23] ассоциация (от лат. associatio – соединение) определена как возникающая в опыте индивида закономерная связь между двумя содержаниями сознания (ощущениями, представлениями, мыслями, чувствами и т. п.), которая выражается в том, что появление в сознании одного из содержаний влечет за собой и появление другого. Ассоциации различают по типу своего образования, а именно: образованные по сходству, по контрасту, по смежности в пространстве или во времени, причинно-следственные и др. В свою очередь, обобщение (от англ. generalization) определено как одна из основных характеристик познавательного процесса, состоящая в выделении и фиксации в сознании относительно устойчивых, инвариантных свойств предметов и их отношений.

В связи с наличием двух фундаментальных абстракций в философской логике различают и два способа образования понятий: генерализирующее и индивидуализирующее [178, ст. Образование понятий]. ***Генерализирующее*** образование, свойственное естествознанию, возникает из интереса фиксировать повторяющиеся явления и процессы. В данном случае сущности предметной области превращаются в экземпляры соответствующего

понятия, причем в такие, которые взаимозаменяемы без ущерба в отношении содержания этого понятия, несмотря на то, что различные сущности сами по себе никогда не могут быть равными.

Индивидуализирующее образование понятий, свойственное историческому познанию, возникает, когда интерес к окружающему миру проявляет себя при ином понимании действительности. В этом случае в любой сущности выделяется нечто особенное, ее отличительный признак. Именно поэтому, укрупняя единичные понятия, например события, мы получаем все те же единичные понятия различного уровня укрупнения. Если в генерализируемых понятиях объем и содержание обратно пропорциональны друг другу, то в индивидуализируемых понятиях они являются прямо пропорциональными. Последнее обусловлено тем, исторический подход дает не просто изображения индивидуального содержания своих сущностей, а конструирует систему скоординированных понятий с индивидуальным содержанием, где на любом уровне укрупнения будет иметь место все та же единичность, но уже с большим единством. В сравнении с содержанием своих частей понятие целого оказывается богаче содержанием.[16]

Таким образом, материал, данный в представлениях, может преобразовываться посредством двух различных способов его абстрагирования и логико-теоретической переработки. Это приводит к появлению двух разновидностей логической формы знания и детерминирует две альтернативные логики: общую, или генерализирующую, и трансцендентальную, или индивидуализирующую.[17] Вместе с тем генерализирующая и индивидуализирующая логики носят сугубо описательный характер и отличаются не логическими законами (они признаются общими), а принципами соотнесения понятий.

Предпримем попытку объединить два перечисленных подхода к образованию и выражению понятий. Для этого генерализирующее образование будем рассматривать как основанное на абстракции обобщения, а индивидуализирующее – на абстракции ассоциа-

---

[16] В формальной логике отсутствие явных средств для выражения индивидуализирующего образования понятий приводит, в частности, к проблемам отражения времени и пространства [128].

[17] Различение общей и трансцендентальной логики восходит к И. Канту [115, с. 120]. Оно состоит в том, что общая логика отвлекается от всякого содержания познания и изучает одну лишь форму познания в понятиях, суждениях и умозаключениях, т.е. исследует формальные правила рассуждений, а трансцендентальная логика имеет дело с определенным содержанием познания, т.е. исследует онтологическую структуру универсума. При этом общая логика, как и математика, тяготеет к работе с однородным универсумом, отвлекаясь от его качественной неоднородности и сложности. Например, силлогистика работает в рамках в родо-видового мира универсалий, а современная логика предикатов – в номиналистическом универсуме индивидов и отношений. Поэтому кантовская идея об учете семантики универсума при разработке синтаксических формализмов (в частности, учет его качественной разнородности) является до сих пор актуальной [123].

ции. В отличие от других известных формализмов, абстракции ассоциации и обобщения будем выражать совместно и независимо друг от друга.[18]

2.3.1. Абстракция обобщения

При обобщении происходит образование понятия на основе одного или нескольких подобных понятий, когда новое понятие сохраняет общие признаки исходных понятий и игнорирует их более тонкие различия.

Обобщение – порождение понятия-обобщения на основе пересечения схем обобщаемых понятий и расширенного объединения их экстенсионалов. При ***специализации***, наоборот, – для заданного понятия-обобщения выделяются похожие на него понятия.

Определение 2.2. *Понятия* N_j, $j = \overline{0, m-1}$, *использованные для образования нового понятия* N_G *путем их обобщения, будем называть обобщаемыми понятиями. Для интенсионала, экстенсионала и схемы понятия-обобщения* N_G *справедливы следующие выражения:*

$$\begin{cases} \operatorname{shm} N_G = \bigcap\limits_{j=0}^{m-1} \operatorname{shm} N_j; \\ \operatorname{int} N_G \supseteq \bigcup\limits_{j=0}^{m-1} \operatorname{int} N_j; \\ \operatorname{ext} N_G \supseteq \bigcup\limits_{j=0}^{m-1} \operatorname{ext} N_j. \end{cases} \tag{2.4}$$

Выражения (2.4) непосредственно следуют из определения обобщения, так как пересечение схем обобщаемых понятий N_j обеспечивает выделение их общих признаков, а объединение интенсионалов позволяет рассматривать сущности обобщаемых понятий как принадлежащие и понятию-обобщению. В свою очередь, выражение для экстенсионала N_G следует из того, что любая сущность $E \in \operatorname{ext} N_G$ является хотя бы одной из сущностей $E \in \operatorname{ext} N_j$.

Пример 2.8. Пусть имеются два понятия: N_1 «двузначное кратное 5» и N_2 «двузначное с суммой 5»:

---

[18] В таких теориях и формализмах как концептуальный анализ (Никаноров, 1972), семантическая сеть (Коллинз и Квилиан, 1969; Цейтин, 1985), исчисление предикатов (Кольмероэ, 1975), теория концептуальной зависимости (Шенк и Ригер, 1974), концептуальный вывод (Плесневич, 2004), формальный анализ понятий (Вилли и Гантер, 1999), концептуальные графы (Сова, 1984), EER-модель (Чен, 1976; Броди и Мулополос, 1984) абстракция ассоциации выражается связями между обобщенными понятиями и не используется для образования отдельных (независимых) понятий-ассоциаций. Подробный анализ перечисленных теорий и формализмов приведен в 2.6.

$$\text{«двузначное кратное 5»} = \begin{cases} \text{shm}\, N_1 = (D, R); \\ \text{int}\, N_1 = \{(2,5)\}; \\ \text{ext}\, N_1 = \{10, 15, \ldots, 95\}, \end{cases}$$

$$\text{«двузначное с суммой 5»} = \begin{cases} \text{shm}\, N_2 = (D, S); \\ \text{int}\, N_2 = \{(2,5)\}; \\ \text{ext}\, N_2 = \{14, 23, 32, 50\}, \end{cases}$$

Определим понятие N_G «двузначное» как обобщение понятий N_1 и N_2:

$$\text{«двузначное»} = \begin{cases} \text{shm}\, N_G = (D); \\ \text{int}\, N_G = \{(2)\}; \\ \text{ext}\, N_G = \{10, 11, ..., 99\}, \end{cases}$$

Заметим, что при обобщении произошло расширение понятия, и его экстенсионал пополнился новыми сущностями, которые ранее не принадлежали экстенсионалам обобщаемых понятий. ♦

При обобщении подобные видовые понятия соотносятся с родовым, а при специализации, наоборот, – родовые понятия делятся на видовые. При специализации схема понятия расширяется, и в нее включаются новые признаки, которые ранее были несущественными, или схема остается прежней, но происходит некоторое ограничение интенсионала исходного понятия.

Пример 2.9. Рассмотрим специализацию понятия «двузначное кратное 5» из примера 2.8. Получим понятие N_1 путем ограничения интенсионала:

$$\text{«двузначное кратное 10»} = \begin{cases} \text{shm}\, N_1 = (D, R); \\ \text{int}\, N_1 = \{(2,10)\}; \\ \text{ext}\, N_1 = \{10, 20, \ldots, 90\}, \end{cases}$$

а понятие N_2 зададим путем добавления нового признака «сумма»:

$$\text{«двузначное кратное 5 с суммой 5 или 6»} = \begin{cases} \text{shm}\, N_2 = (D, R, S); \\ \text{int}\, N_2 = \{(2, 5, \{5, 6\})\}; \\ \text{ext}\, N_2 = \{15, 50, 60\}. \end{cases} \blacklozenge$$

2.3.2. Абстракция типизации

Типизация может быть определена как один из видов абстракции, когда множество полезных для достижения некоторой цели и сходных по функциям понятий объединяются и обозначаются одним для всего множества понятием. Типизация лежит в основе образования простых понятий, когда множество сущностей объединяется и обозначается как новое понятие.

При типизации происходит группировка понятий на основе соответствия их интенсионалов некоторому эталону. Понятие-тип определяет то общее, что присуще некоторой

совокупности понятий. Причем это общее выражает однородность, однотипность понятий. Обратным по отношению к типизации является конкретизация, при которой происходит выделение из понятия-типа типизированных в нем понятий.

Типизация является частным случаем обобщения. В отличие от обобщения, при типизации имеется возможность определить для сущности из экстенсионала понятия-типа исходное ее понятие. Для этого задается множество признаков, называемое ключом.

Типизация – порождение понятия-типа на основе пересечения схем типизируемых понятий и строгого объединения их экстенсионалов. При ***конкретизации*** из понятия-типа извлекаются типизированные в нем понятия. Для идентификации исходного понятия у сущностей понятия-типа используется подмножество признаков, называемое ***ключом***.

Определение 2.3. *Понятия* N_j *(* $j = \overline{0, m-1}$*), использованные для образования нового понятия* N_T *путем их типизации, будем называть типизируемыми. Для интенсионала, экстенсионала и схемы понятия-типа* N_T *справедливы следующие выражения:*

$$\begin{cases} \operatorname{shm} N_T = \bigcap\limits_{j=0}^{m-1} \operatorname{shm} N_j, \quad \operatorname{key} N_T \subseteq \operatorname{shm} N_T; \\ \operatorname{int} N_T = \bigcup\limits_{j=0}^{m-1} \operatorname{int} N_j; \\ \operatorname{ext} N_T = \bigcup\limits_{j=0}^{m-1} \operatorname{ext} N_j. \end{cases} \tag{2.5}$$

Пример 2.10. Пусть заданы понятие N_1 «двузначное кратное 5 и суммой 8» и понятие N_2 «кратное 9 суммой 9 и произведением 14»:

$$\text{«двузначное кратное 5 и суммой 8»} = \begin{cases} \operatorname{shm} N_1 = (D, R, S); \\ \operatorname{int} N_1 = \{(2,5,8)\}; \\ \operatorname{ext} N_1 = \{35, 80\}, \end{cases}$$

$$\text{«кратное 9 суммой 9 и произведением 14»} = \begin{cases} \operatorname{shm} N_2 = (R, S, M); \\ \operatorname{int} N_2 = \{(9,9,14)\}; \\ \operatorname{ext} N_2 = \{27, 72\}. \end{cases}$$

Найдем понятие N_T как результат типизации понятий N_1 и N_2:

$$\text{«кратное 5 суммой 8 или кратное 9 суммой 9»} = \begin{cases} \operatorname{shm} N_T = (R, S), \quad \operatorname{key} N_T = (S); \\ \operatorname{int} N_T = \{(5,8), (9,9)\}; \\ \operatorname{ext} N_T = \{27, 35, 72, 80\}. \end{cases}$$

Для однозначного соотнесения произвольной сущности из $\operatorname{ext} N_T$ к одному из типизированных понятий в качестве ключа необходимо выбрать набор признаков: $\operatorname{key} N_T = (S)$. Если $S(8, E)$ истина, то сущность E принадлежит N_1, а если $S(9, E)$ истина, то N_2. ♦

В общем случае возможна ситуация, когда одно и то же понятие из экстенсионала понятия-типа одновременно принадлежит разным типизированным понятиям, т.е. ключ не является уникальным, а его значение идентифицирует не одно, а несколько понятий. Если потребовать уникальность ключа, при которой любое его значение задает единственное понятие, то пересечение экстенсионалов типизируемых понятий должно быть пустым,

$$\bigcap_{j=0}^{m-1} \operatorname{ext} N_j = \varnothing .$$

Заметим, что абстракция типизации лежит в основе образования объемных понятий, когда в результате сопоставления сущностей предметной области происходит их группировка на основе эквивалентности схем. На этом основании абстракцию типизации можно рассматривать как ***элементарную*** абстракцию, так как она используется для образования объемных понятий независимо оттого, что будет получено в результате такого образования: понятие-обобщение или понятие-ассоциация.

2.3.3. Абстракция ассоциации

При ассоциации устанавливается взаимосвязь между сущностями, принадлежащими одному и тому же или разным понятиям. Ассоциация выражает специфическое соединение сущностей ассоциируемых понятий в единое целое. Это соединение сохраняет индивидуальные свойства сущностей абстрагируемых понятий, благодаря чему имеется возможность от сущности одного понятия перейти к одной или нескольким сущностям других понятий.

Ассоциация – порождение понятия-ассоциации на основе объединения схем ассоциируемых понятий и ограничения декартова произведения их экстенсионалов. При ***индивидуализации*** понятие-ассоциация разъединяется на ассоциируемые понятия. Для установления взаимосвязи сущностей, принадлежащих ассоциируемым понятиям, используется набор признаков, называемый ***связью***.

Определение 2.4. *Понятия* N_j *(*$j = \overline{0, m-1}$*), использованные для образования нового понятия* N_A *путем их ассоциации, будем называть ассоциируемыми. Для интенсионала, экстенсионала и схемы понятия-ассоциации* N_A *справедливы следующие выражения:*

$$\begin{cases} \operatorname{shm} N_A = \bigcup\limits_{j=0}^{m-1} \operatorname{shm} N_j, \quad \operatorname{lnk} N_A \subseteq \operatorname{shm} N_A; \\ \operatorname{int} N_A \subseteq \bigtimes\limits_{j=0}^{m-1} \operatorname{int} N_j; \\ \operatorname{ext} N_A \subseteq \bigtimes\limits_{j=0}^{m-1} \operatorname{ext} N_j, \end{cases} \tag{2.6}$$

где объединение признаков осуществляется с повторением элементов и сохранением их порядка, $\ln N_A$ *– подсхема понятия-ассоциации, задающая ассоциативную связь.*

Пример 2.11. Рассмотрим понятие N_1 «однозначное кратное 3 или 5» и понятие N_2 «двузначное с суммой 3 или 5»:

$$\text{«однозначное кратное 3 или 5»} = \begin{cases} \text{shm}\, N_1 = (D, R); \\ \text{int}\, N_1 = \{(1,3),(1,5)\}; \\ \text{ext}\, N_1 = \{3,5,6,9\}, \end{cases}$$

$$\text{«двузначное с суммой 3 или 5»} = \begin{cases} \text{shm}\, N_2 = (D, S); \\ \text{int}\, N_2 = \{(2,3),(2,5)\}; \\ \text{ext}\, N_2 = \{12,14,21,23,30,32,50\}, \end{cases}$$

Определим понятие N_A как ассоциацию N_1 и N_2 следующим образом:

$$\text{«ассоциация } N_1 \text{ и } N_2\text{»} = \begin{cases} \text{shm}\, N_A = (D, R : D, S); \\ \text{int}\, N_A = \{(1,3:2,3),(1,5:2,5)\}; \\ \text{ext}\, N_A = \{3:12, 3:21, ..., 5:50\}, \end{cases}$$

где установлены связи между сущностями N_1 кратностью 3 и 5 и сущностями N_2 с суммой 3 и 5 соответственно. Признаки, позволяющие переходит от одной сущности ассоциированного понятия к сущности другого, образуют подсхему $\ln N_A = (R : S)$ (рис. 2.7).

♦

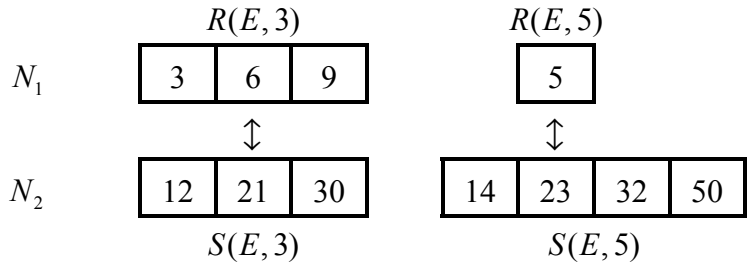

Рис. 2.7.Связь сущностей понятий N_1 и N_2

При ассоциации могут образовываться различные типы связей между сущностями, а именно: $1:1$ (одна к одной), $1:\infty$ (одна ко многим) и $\infty:\infty$ (многие ко многим). Заметим, что в примере 2.11 показана связь типа $\infty:\infty$.

Используемый формализм позволяет рассматривать ассоциацию, определяемую на трех и более понятиях. В этом случае поучаем ассоциативный ряд, а связь обеспечивает навигацию между сущностями ассоциируемых понятий.

Пример 2.12. Рассмотрим ассоциативный ряд «градации серого». Построим на его основе понятие-ассоциацию «градации серого», которая состоит из понятий-сущностей «черный», «серый» и «белый». Тогда, имея сущность «белый», из схемы понятия-

ассоциации находим связанные этой ассоциацией другие две сущности: «серый» и «черный». Очевидно, что в рассматриваемом случае связью является вся схема понятия-ассоциации. ♦

В рассмотренном выше примере индивидуализирующее образование понятий использовано в частном виде, так как полученное в результате понятие имеет единичный объем. В общем случае возможно индивидуализирующее образование и объемных понятий, при котором происходит объединение схожих сущностей на основе абстракции типизации.

Пример 2.13. Рассмотрим тот же ассоциативный ряд «градации серого», но состоящий уже из объемных понятий «черный», «серый» и «белый». В этом случае экстенсионалы ассоциируемых понятий содержат сущности, соответствующие различным формам выражения в тексте черного, серого и белого цветов, а сама форма выражения задается общим признаком у этих понятий, входящим в ассоциативную связь.

Тогда, имея некоторую сущность понятия «серый» и зная форму ее выражения (например, на одном из естественных языков; интенсивностью красного, зеленого и синего цветов; яркостью и т.п.), в экстенсионалах понятий «белый» и «черный» можно найти ассоциированные с исходной сущности этих понятий и восстановить тройку «черный-серый-белый». ♦

2.3.4. Абстракция агрегации

Агрегация используется в тех случаях, когда вновь порождаемое составное понятие включает исходные понятия в качестве своих составных частей. Абстракция агрегации позволяет выразить внутренние связи, существующие между отдельными сущностями предметной области. При этом структура составного понятия раскрывается путем его разделения на совокупность составляющих его понятий. Процесс, противоположный агрегации, называется декомпозицией.

Агрегация является предельным случаем ассоциации. В отличие от ассоциации, где между сущностями устанавливаются только часть связей, при агрегации устанавливаются все возможные связи, т.е. в агрегацию могут входить произвольные комбинации сущностей агрегируемых понятий, а при ассоциации – только взаимосвязанные. Отсюда следует, что на одном и том же множестве понятий можно задать несколько ассоциаций, в то время как агрегация осуществляется с точностью до агрегируемых понятий.

Агрегация – порождение понятия-агрегата на основе объединения схем агрегируемых понятий и декартова произведения их экстенсионалов. При ***декомпозиции*** понятие-агрегат разделяется на составляющие его агрегированные понятия.

Определение 2.5. *Понятия* N_j *(* $j = \overline{0, m-1}$*), использованные для образования нового понятия* N_C *путем их агрегации, будем называть агрегируемыми. Для интенсионала, экстенсионала и схемы понятия-агрегата* N_C *справедливы следующие выражения:*

$$\begin{cases} \text{shm}\, N_C = \bigcup\limits_{j=0}^{m-1} \text{shm}\, N_j; \\ \text{int}\, N_C = \times\limits_{j=0}^{m-1} \text{int}\, N_j; \\ \text{ext}\, N_C = \times\limits_{j=0}^{m-1} \text{ext}\, N_j, \end{cases} \tag{2.7}$$

где объединение признаков осуществляется с повторением элементов и сохранением их порядка, а знаком $\times$ *обозначено декартово произведение.*

При агрегации вновь образованное понятие приобретает одновременно все признаки агрегируемых понятий, следовательно, у понятия-агрегата все интенсионалы $\text{int}\, N_j$ выполнимы, а экстенсионал $\text{ext}\, N_C$ равен декартову произведению экстенсионалов $\text{ext}\, N_j$. Так как все признаки агрегируемых понятий передаются понятию-агрегату, то схемы понятий объединяются, причем объединение производится с повторением одноименных признаков.

Пример 2.14. Рассмотрим понятие N_1 «однозначное кратное 3» и понятие N_2 «двузначное с суммой 5»:

$$\text{«однозначное кратное 3»} = \begin{cases} \text{shm}\, N_1 = (D, R); \\ \text{int}\, N_1 = \{(1, 3)\}; \\ \text{ext}\, N_1 = \{3, 6, 9\}, \end{cases}$$

$$\text{«двузначное с суммой 5»} = \begin{cases} \text{shm}\, N_2 = (D, S); \\ \text{int}\, N_2 = \{(2, 5)\}; \\ \text{ext}\, N_2 = \{14, 23, 32, 50\}, \end{cases}$$

Определим понятие N_C как агрегацию N_1 и N_2:

$$\text{«агрегация } N_1 \text{ и } N_2\text{»} = \begin{cases} \text{shm}\, N_C = (D, R : D, S); \\ \text{int}\, N_C = \{(1, 3 : 2, 5)\}; \\ \text{ext}\, N_C = \{3 : 14, 3 : 23, ..., 9 : 50\}, \end{cases}$$

где N_C интерпретируется как «упорядоченная пара, состоящая из однозначного кратного 3 и двузначного с суммой 5» и выражается набором из двух сущностей. Для разделения признаков и сущностей, принадлежащим различным агрегируемым понятиям, использовано двоеточие. ♦

Заметим, что абстракция агрегации, как и абстракция типизации, является ***элементарной***. В отличие от абстракции типизация, служащей для образования объемных поня-

тий, абстракция агрегации лежит в основе образования составных понятий, независимо оттого, получается ли в результате такого образования понятие-обобщение или понятие-ассоциация. Это проявляется при формировании схем составных понятий как объединения схем понятий-признаков, на которых эти понятия определяются.

2.4. Семантическая теория понятий

Прикладная ценность любой теории определяется ее способностью описывать некоторое множество предметных областей. Отличительной особенностью формальных теорий (исчислений, или синтаксических теорий) является то, что в последовательность знаков, получаемых в этих теориях, не вкладывается никакого смысла, пока не введена их явная интерпретация. Но введение интерпретации не относится к зоне ответственности самой синтаксической теории. Для этого требуется привлечение другой, более общей и более мощной теории (метатеории, или семантической теории), в которую в качестве фрагмента входит интерпретируемая синтаксическая теория. Только в этом случае, на языке метятеории, становится возможных формулировка свойств синтаксической теории. По этой причине любая синтаксическая теория подразумевает наличие некоторой стандартной области интерпретации, относительно которой делаются все содержательные утверждения (метаутверждения).

Так, в математической логике различают исчисление высказываний (исчислении предикатов) и алгебру высказываний (алгебру предикатов), причем алгебра высказываний (алегбра предикатов) рассматривается как семантическая теория, поставляющая исчислению высказываний (исчислению предикатов) стандартную область интерпретации [251, с. 150, 157]. В свою очередь исчисление высказываний (исчислений предикатов) является синтаксической теорией и используется как логическое средство в других математических теориях. Однако все высказывания о полноте и непротиворечивости исчисления высказываний (исчисления предикатов) формулируются не ностиельно прикладной математической теории, для описания которой оно предназначено, а относительно алгебры высказываний (алгебры предикатов), которая, в этом случае, является семантической теорией и непременно должна быть включена в синтаксические средства прикладной теории.[19]

Поставим целью разработать семантическую теорию понятий, которую в дальнейшем будем использовать для доказательства содержательных утверждений о синтаксической теории понятий, применяемой для формализации понятий и связей между ними. По

[19] Замечательным в этом плане является учебное пособие [84], где сначала вводится формальный язык (символический язык) и рассматриваются семантические теории высказываний и предикатов (семантика логики предложений и семантика логики предикатов), а затем определяются соответсвующие синтаксические теории (синтаксис логики предложений и синтаксис логики предикатов).

своей сути построение семантической теории понятий уже начато выше, где дано формальное определение понятий и их абстракций.

2.4.1. Понятийная структура

В отличие от известных формализмов (семантических сетей, концептуальных схем, и др.), где на понятиях задается большое множество отношений различной природы, введем в использование новый формализм – понятийную структуру, которую определим множеством понятий с четырьмя видами отображений, единственное назначение которых – показать способы образования понятий.

Определение 2.6. Понятийной структурой[20] $S = \langle N, G, T, A, C \rangle$ *будем называть конечное множество понятий* $N = \{N_0, N_1, \ldots, N_{n-1}\}$, *на которых заданы четыре конечные множества отображений абстрагирования: обобщения* $G = \{g_0, g_1, \ldots, g_{n_g-1}\}$, *типизации* $T = \{t_0, t_1, \ldots, t_{n_t-1}\}$, *ассоциации* $A = \{a_0, a_1, \ldots, a_{n_a-1}\}$ *и агрегации* $C = \{c_0, c_1, \ldots, c_{n_c-1}\}$, *где*

$$g(N_{g_o}, N_{g_1}, \ldots, N_{g_{i-1}}) \to N_g,$$

$$t(N_{t_o}, N_{t_1}, \ldots, N_{t_{j-1}}) \to N_t,$$

$$a(N_{a_o}, N_{a_1}, \ldots, N_{a_{k-1}}) \to N_a,$$

$$c(N_{c_o}, N_{c_1}, \ldots, M_{c_{l-1}}) \to N_c,$$

для всех $g \in G$, $t \in T$, $a \in A$ *и* $c \in C$.

Если по условиям решаемой задачи не требуется уточнение способа абстрагирования понятий, то понятийную структуру будем рассматривать как множество понятий, на котором установлены два типа отображений: отображения дифференциации и отображения интеграции:

$$S = \langle N, D, I \rangle,$$

где $D = \{d_0, d_1, \ldots, d_{n_d-1}\}$ и $I = \{i_0, i_1, \ldots, i_{n_i-1}\}$ множества отображений дифференциации и интеграции, заданные на понятиях N.

Пример 2.15. На рис. 2.8 приведен фрагмент понятийной структуры, где показано, что для понятия N_3 определены два отображения обобщения: $g_0(N_1, N_4)$ и $g_1(N_4)$, а для понятия N_1 – отображения агрегации $c_0(N_2)$ и $c_1(N_4)$. Отображения интеграции понятий

[20] Понятийная структура наиболее близка расширенной модели данных «сущность-связь» (EER-модель) [267], однако в EER-модели элементами являются не понятия, а типы данных, причем для детализации модели используется трудно формализуемая семантическая разметка в виде дополнительных нотаций и ограничений.

помечены квадратом, а дифференциации – кругом, причем строгие или элементарные отображения (типизацию и агрегацию) будем изображать сплошной линией, а нестрогие (обобщение и ассоциацию) – пунктирной. ♦

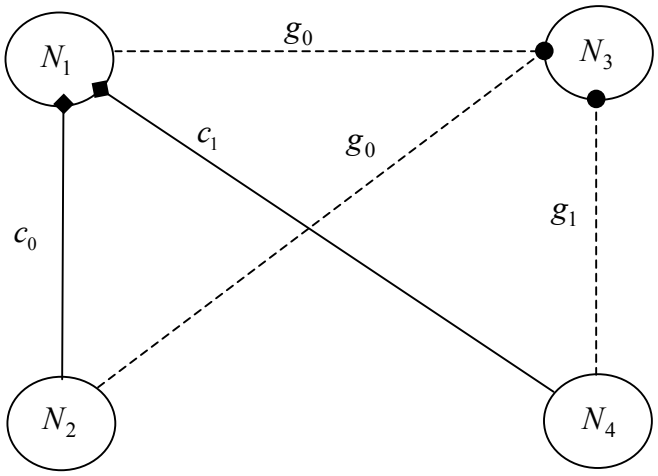

Рис. 2.8. Фрагмент понятийной структуры

Пример 2.16. На рис. 2.9 приведена одна из возможных понятийных структур предметной области «Числа», полученная для проблематики, связанной с реализацией чисел на вычислительных средствах дискретного действия.

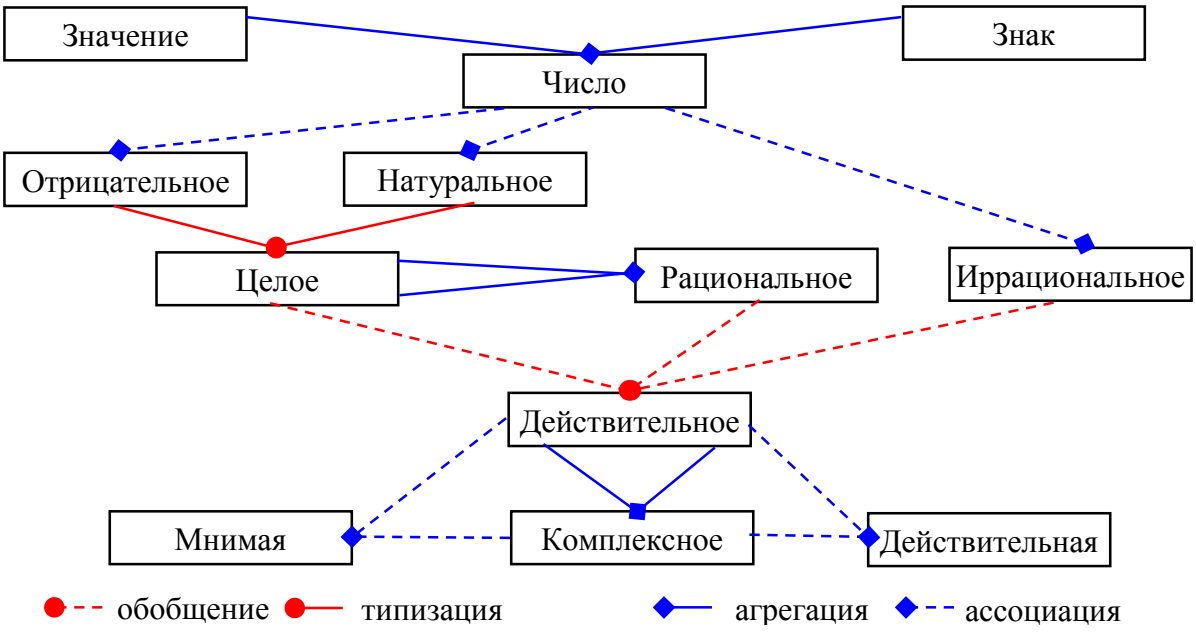

Рис. 2.9. Понятийная структура

Сущности предметной области «Числа» разделим на две группы: базовые сущности, имеющие преимущественно денотационную форму выражения, и вспомогательные, для которой достаточно сигнификативной формы. С учетом рассматриваемой проблематики к базовым (денотационным) сущностям отнесем сами числа, а к вспомогательным (сигнификативным) – операции над ними. Такое разделение диктуется архитектурой наи-

более распространенных вычислительных средств, реализующих концепцию командного управления.[21]

Из понятийной структуры видно, что простыми (первичными) являются только понятия «Знак» и «Значение». Остальные понятия составные и образованы на основе абстрагирования. Понятие «Число» полностью характеризуется первичными понятиями и является их агрегацией. С «Натуральным», «Отрицательным» и «Иррациональным» ассоциируется понятие «Число», которое в каждом случае ограничивается особым образом. «Целое» является типизацией «Натурального» и «Отрицательного» с ключом, равным «Знаку». «Рациональное» агрегирует два «Целых», а «Действительное» определено как обобщение «Целого», «Рационального» и «Иррационального». В свою очередь «Комплексное» – агрегация двух «Действительных», а «Действительная» и «Мнимая» части заданы как ассоциации «Комплексного» и «Действительного» со связью, задаваемой первым или вторым «Действительным» понятия «Комплексное». ♦

2.4.2. Алгебра понятийных структур

Совокупность понятий и четырех множеств отображений одних понятий в другие ранее было определено как понятийная структура (определение 2.6 на с. 85). Учитывая то, что типизация является частным случаем обобщения, которое, в свою очередь выражает дифференциацию понятий, а агрегация является частным случаем ассоциации, выражающей интеграцию понятий, абстрагируемся от частных способов образования понятия и будем далее использовать только дифференциацию и интеграцию.

В алгебре понятийных структур различают понятия и оперции над понятиями. При этом под понятием понимается знак (слово), о котором можно вполне определенно сказать, что он (оно) обозначает некоторое множество других понятий. В алгебре понятий отвлекаются от конкретного содержания понятий и интересуются лишь вопросом, из каких других понятий образовано каждое понятие.

Для обозначения понятий мы будем использовать знаки N_i, где i – натуральное число, включая ноль. Для того, чтобы показать, что понятие N_i образовано из n понятий N_1, N_2, ..., N_n путем их дифференциации будем использовать запись вида $\delta_n(N_1, N_2, \ldots, N_n) \to N_i$, а путем их интеграции – запись вида $\lambda_n(N_1, N_2, \ldots, N_n) \to N_i$.[22]

---

[21] Возможно построение понятийной структуры предметной области «Числа» для другого типа архитектуры, реализующий концепцию управления по данным [98]. Очевидно, что этом случае разделение сущностей на сигнификативные и денотационные будет другим, например, операции над данными объявляются денотационными сущностями, а типы данных – сигнификативными.

[22] Знак δ выбран для обозначения дифференциации понятий по его внешнему сходству со знаком дифференциала, а знак λ – по его сходству со знаком интеграла.

Определение 2.7. *Пусть n – любое натуральное число. Тогда n-местным дифференциалом (интегралом) понятий, определенным на множестве N, называется всякое однозначное отображение n-ой декартовой степени множества N в само множество N,*

$$\delta_n : N^n \to N \ (\lambda_n : N^n \to N).$$

Если имеются некоторые понятия, то из них при помощи операций интеграции и дифференциации можно образовать (выразить) другие понятия. Например, если понятия N_1 и N_2 служат для выражения понятия N_3 путем дифференциации, запишем $\delta_2(N_1, N_2) \to N_3$, а если понятие N_4 образовано при интергации понятия N_1, N_2 и N_3, то $\lambda_3(N_1, N_2, N_3) \to N_4$.

Определение 2.8. Алгеброй понятийных структур *называется тройка $< N, \Delta, \Lambda >$, где N – множество понятий $N = \{N_0, N_1, \ldots\}$, а Δ (Λ) – множество операций дифференциации (интеграции) понятий различной местности:*

$$\Delta = \{\delta_0, \delta_1, \delta_2, \ldots\} \ (\Lambda = \{\lambda_0, \lambda_1, \lambda_2, \ldots\}).$$

Для определенности положим, $\delta_0() \to N_0$, $\lambda_0() \to N_0$, где N_0 – выделенное понятие, которое будем называть пустым. Некоторые другие свойства операций интеграции и дифференциации следуют из определений абстракций (см. 2.3 на с. 73) и приведены в табл. 2.1, где знак ~ обозначает синтаксическую эквивалентность.

Таблица 2.1. Свойства операций алгебры понятийных структур

Дифференциация	Интеграция
$\delta_1(N_i) \to N_i$	$\lambda_1(N_i) \to N_i$
$\delta_k(\ldots, N_i, \ldots, N_i, \ldots) \sim \delta_{k-1}(\ldots, N_i, \ldots)$	$\lambda_k(\ldots, N_i, \ldots, N_i, \ldots) \not\sim \lambda_{k-1}(\ldots, N_i, \ldots)$
$\delta_k(\ldots, N_i, \ldots, N_j, \ldots) \sim \delta_k(\ldots, N_j, \ldots, N_i, \ldots)$	$\lambda_k(\ldots, N_i, \ldots, N_j, \ldots) \not\sim \lambda_k(\ldots, N_j, \ldots, N_i, \ldots)$

Формулой алгебры понятийных структур называется множество отображений дифференциации и интеграции, заданное на некотором конечном подмножестве N.

Пример 2.17. Представим в виде формулы понятийную структуру, изображенную на рис. 2.8:

$$\{\lambda(N_3, N_4) \to N_1, \ \delta(N_1, N_3, N_4) \to N_2\}.$$

Из формулы, описывающей абстрагирование понятий N_1, N_2, N_3 и N_4, видно, что понятие N_1 является интеграцией понятий N_3 и N_4, а понятие N_2 – дифференциацией

понятий N_1, N_3 и N_4. Так как способы абстрагирования понятий N_3 и N_4 не определены, делаем вывод, что эти понятия первичные. ♦

2.4.3. Схемы понятий

Из определения понятия следует, что каждое понятие обладает схемой – набором признаков, на которых понятие определено. Однако при задании понятийной структуры определению подлежат только отображения понятий. В связи с чем возникает задача определения схемы понятий по понятийной структуре. Потребность в решении этой задачи возникает и при абстрагировании, когда над схемами понятий выполняются теоретико-множественные операции пересечения и объединения с учетом повторения элементов.

Рассмотрим задачу определения схем понятий по имеющейся понятийной структуре. Для ее решения схему понятия определим как набор простых понятий, на которых понятие определено. Схему понятий будем вычислять на основе отображений абстрагирования, заданных в понятийной структуре, по следующей рекуррентной процедуре:

– схема простого понятия N равна (N);

– схема понятия, полученного в результате дифференциации, равна пересечению схем дифференцируемых понятий;

– схема понятия, полученного в результате интеграции, равна объединению схем интегрируемых понятий;

– схема понятия, полученного в результате дифференциации и интеграции, равна объединению схем интегрируемых понятий, принадлежащему пересечению схем дифференцируемых понятий.

Операции объединения и пересечения множеств в рассматриваемом случае выполняются с учетом повторения элементов. Одновременная интеграция и дифференциация понятий является выразительным средством, которое позволяет уточнить схему определяемого понятия путем ограничения пересечения схем дифференцируемых понятий указанием некоторого подмножества этого пересечения. Очевидно, если схема, получаемая через интеграцию понятий, не содержится в схеме, вычисленной на основе дифференциации, то последнее говорит о недопустимом описании понятийной структуры.

Пример 2.18. Покажем вычисление схем понятий, заданных понятийной структурой на рис. 2.9.

Простыми являются понятия «Знак» и «Значение», следовательно, их схемы тождественны самим понятиям. Схемы понятий «Число», «Натуральное», «Отрицательное», «Иррациональное» и «Целое» равны («Знак», «Значение»). Однако схема «Рационального» и «Комплексного» числа определяется как («Знак», «Значение» : «Знак», «Значение»).

При обобщении «Действительного» опять получаем схему («Знак», «Значение»). Самую сложную схему («Знак», «Значение» : «Знак», «Значение» : «Знак», «Значение») имеют «Мнимая» и «Действительная» части.

Заметим, что в зависимости от проблематики решаемых задач для такой хорошо формализованной области как «Числа» возможно построение нескольких понятийных структур. ♦

Требование вычислимости схем понятий является распространением свойства регулярности (фундированности)[23] на все понятия, а не только на понятие множества. В этом случае вычислимость схем понятий по понятийной структуре предметной области гарантирует отсутвие определений понятий через самих себя, что в любой формальной или содержательной теории, предендующей на адекватность, признается недопустимым.

Таким образом, из содержательных представлений о понятийной структуре и свойств абстрагирования следует, что в списках дифференцируемых и интегрируемых понятий не может появляться определяемое понятие, задаваемое прямо или косвенно. Поэтому, при построении схемы понятия необходимо осуществлять контроль за наличием циклов в понятийной структуре, выражающих указанное ограничение.

2.4.4. Свойства понятийных структур

С целью исследования формализма понятий и их абстракций рассмотрим предварительно те элементарные следствия, которые могут быть получены из приведенных выше определений.

Свойство 2.1. *Любая понятийная структура содержит не более чем одно обобщение для каждого включенного в нее понятия.*

Доказательство. Пусть имеется понятие N_g, для которого определены два различных обобщения: $g_1(N_{11}\dots N_{1n}) \to N_g$ и $g_2(N_{21}\dots N_{2m}) \to N_g$. Если окажется, что экстенсионалы понятия N_g, полученные при обобщениях g_1 и g_2 различны, то отсюда сле-

[23] Антиномии Рассела и другие антиномии теории множеств привели к необходимости критического рассмотрения способов умозаключений, применяемых при доказательствах. В конечном итоге было признано, что недопустимо определять множество с помощью некоторой совокупности множеств, а затем причислять его к этой совокупности [251, с. 125]. По этой причине Б. Расселом и Уайтхедом была построена теория типов, исключающая рассмотрение множеств, приводящих к антиномии [35, с. 12]. Однако теория типов оказалась существенно бедной, так как вводит чрезмерные ограничения, в силу которых этот формализм становится очень сложным. В итоге, ограничения на допустимые виды умозаключения никак не коснулись самого исчисления предикатов – структура и способы правильного конструирования предикатов так и остались вне рамой теории. Последнее, в частности, привело к включению П. Бернайсом и К. Геделем в аксиоматику современной теории множеств аксиомы фундирования [13, с. 15], ограничивающей предмет теории множеств только регулярными (фундированными) множествами, т.е. множествами, не принадлежащими самим себе.

дует противоречивость понятийной структуры, так как существует сущность понятия N_g, описанная как принадлежащая, так и как не принадлежащая этому понятию.

Остается предположить, что обобщения g_1 и g_2 порождают один и тот же экстенсионал. В таком случае такие обобщения могут быть объединены в одно $g(N_{11}\dots N_{1n}\ N_{21}\dots N_{2m}) \to N_g$. ♦

Таким образом, использование различных обобщений одного и того же понятия ничего, кроме возможности получения противоречивой понятийной структуры не дает и, следовательно, может быть запрещено без ухудшения выразительных качеств формализма.

Пример 2.19. Рассмотрим понятийную структуру, заданную списком отображений дифференциации и интеграции:

$$()N_0() \quad ()N_1() \quad ()N_2(N_1\ N_3) \quad ()N_3(N_0\ N_1) \quad (N_1\ N_2)N_4() \quad (N_0\ N_4)N_5(),$$

где список дифференцируемых понятий располагается слева от знака понятия-дифференциала, а список интегрируемых понятий – справа. В заданной понятийной структуре каждое понятие имеет по одному обобщению. Следовательно, эта понятийная структура не может привести к появлению противоречий, связанных с множественностью обобщения. Однако такая структура неполна. Чтобы показать это, вычислим схемы понятий (см. 2.4.3 на с. 89): $\mathrm{shm}\,N_0 = (N_0)$, $\mathrm{shm}\,N_1 = (N_1)$, $\mathrm{shm}\,N_2 = (N_0\ N_1\ N_1)$, $\mathrm{shm}\,N_3 = (N_0\ N_1)$, $\mathrm{shm}\,N_4 = (N_1)$, $\mathrm{shm}\,N_5 = ()$. Так как схема N_5 оказалось пустой, то это свидетельствует о неполноте рассматриваемой понятийной структуры. ♦

Свойство 2.2. *В любой понятийной структуре типизируемые и типизированное понятия имеют одинаковые схемы.*[24]

Доказательство. Из определения абстракции типизации следует, что общих признаков типизируемых понятий должно быть достаточно для определения интенсионала понятия-типа, причем таких признаков, которые описывают все множество типизируемых сущностей и только их. Последнее означает, что частные признаки этих понятий являются несущественными для своих собственных интенсионалов и могут быть исключены из рассмотрения. Таким образом, типизации подлежат только те понятия, схемы которых равны схеме понятия-типа: $\mathrm{shm}\,N_T = \mathrm{shm}\,N_j$. ♦

Из свойства 2.2 следует, что каким бы образом ни были бы определены типизируемые понятия, их схемы должны быть одинаковыми. Если такие схемы окажутся разли-

[24] В [81] абстракция типизации изначально определяется на множестве понятий, имеющих одинаковые схемы. Однако в общем случае, форма представления понятия-типа может не соответствовать этому требованию, как, например, это имеет место в примере 2.10.

чающимися, то это означает наличие несущественных признаков у рассматриваемых понятий.

Пример 2.20. Из анализа примера 2.10 следует, что типизируемые понятия N_1 и N_2 имеют различные схемы: $\text{shm}\, N_1 = (D, R, S)$, $\text{shm}\, N_2 = (R, S, M)$. Так как из свойства 2.2 следует, что схемы типизированных понятий должны совпадать, то делаем вывод о наличием несущественных признаков в схемах понятий N_1 и N_2.

Так как схемой понятия-типа N является пересечение схем понятий N_1 и N_2, $\text{shm}\, N = (R, S)$, то необходимо показать, что признак D понятия N_1 и признак M понятия N_2 являются несущественными, т.е. должна существовать форма описания этих понятий без использования признаков D и M соответственно. В противном случае понятие N будет выражать не типизацию понятий N_1 и N_2, а их обобщение.

Обращаясь к области интерпретации находим, что понятие N_1, определенное как «двузначное кратное 5 и суммой 8», тождественно понятию «кратное 5 и суммой 8», так как чисел, состоящих из одной десятичной цифры, которые кратны 5 и имеют сумму цифр, равную 8, в используемом универсуме не существует. Следовательно, признак D является несущественным. Аналогично находим, что у понятия N_2 «кратное 9 с суммой 9 и произведением 14» признак M также является несущественным, так как сущности этого понятия со значение признака M, равным 14, не принадлежат экстенсионалу.

В итоге заключаем, что понятийная структура избыточна, так как понятия N_1 и N_2 имеют несущественные признаки. ♦

Свойство 2.2 не запрещает использования одного и того же понятия в образовании различных понятий-типов. Однако, если такие типизируемые понятия имеются, то это означает только эквивалентность схем у результирующих понятий-типов, а не эквивалентность самих понятий.

Свойство 2.3. *Любая понятийная структура содержит не более чем одну агрегацию для каждого включенного в нее понятия.*

Доказательство. В случае явного задания для понятия N_c двух различных его агрегаций $c_1(\ldots, N_1, \ldots) \to N_a$ и $c_2(\ldots, N_2, \ldots) \to N_a$, делаем вывод, что существуют понятия N_1 и N_2, входящие в эти агрегации и имеющие различные экстенсионалы, $\text{ext}\, N_1 \neq \text{ext}\, N_2$. Тогда из (2.7) следует, что $\text{ext}\, N_c \neq \text{ext}\, N_c$. Получено противоречие, которое доказывает свойство в случае явного задания двух агрегаций.

Для случая неявного, или косвенного, выражения агрегаций предположим, что в результате анализа понятийной структуры для некоторого понятия N_c установлено противоречие $\mathrm{ext}\, N_c \neq \mathrm{ext}\, N_c$, т.е. найдена сущность, которая одновременно как принадлежит, так и не принадлежит экстенсионалу N_c. Тогда из (2.7) следует, что для двух экстенсионалов, полученных разными способами, существует хотя бы один отличный элемент, а это возможно только при наличии как минимум двух различных агрегации у некоторого понятия, которые учтены при двух различных способах получения экстенсионала N_c. Следовательно, для непротиворечивости понятийный структуры необходимо обеспечить единственность отображений агрегации и при неявном ее выражении.♦

Заметим, что свойство 2.3 справедливо и для ассоциации, так как ассоциация, также как и агрегация, формируется на основе объединения схем и декартова произведения экстенсионалов стем отличием, что при ассоциации происходит ограничение экстенсионала понятия-ассоциации, которое задается его интенсионалом. Отсюда следует, что рассуждения, использованные при доказательстве свойства 2.3, могут быть применены и для абстракции ассоциации. Таким образом, как и для агрегации, любая понятийная структура содержит не более чем одну ассоциацию для каждого включенного в нее понятия.

Пример 2.21. Пусть задана понятийная структура списками дифференциации и интеграции понятий:

$$()N_0() \quad ()N_1() \quad ()N_2(N_1\ N_3) \quad ()N_3(N_1\ N_4) \quad ()N_4(N_0\ N_2),$$

где, как и ранее, список дифференцируемых понятий располагается слева от знака понятия-дифференциала, а список интегрируемых понятий – справа.

В рассматриваемой понятийной структуре все понятия имеют единственные интеграции, выраженные явно. Однако понятие N_4 интегрирует понятие N_2, которое, в свою очередь, есть интеграл понятий N_1 и N_3, а N_3 интегрирует понятие N_4. Следовательно N_4 косвенно интегрирует само себя и, тем самым, с самим собой не совпадает. В итоге получаем, что существует сущность, которая в одном случае принадлежит понятию N_4, а в другом случае ему не принадлежит. Следовательно, рассматриваемая понятийная структура противоречива, что является следствием двух взаимно исключающих описаний интеграции понятия N_4. ♦

Заметим, что явное выражение нескольких агрегаций у одного понятия приводит к неоднозначности вычисления его схемы, а неявное выражение – к невозможности завершить вычисление схем из-за наличия циклов. Например, в понятийной структуре из примера 2.21 для определения схемы понятия N_2 необходимо знать схему N_3, а вычисление

схемы N_3 требует определения схемы N_4, для вычисления которой необходима схема исходного понятия N_2.

Свойство 2.4. *В любой понятийной структуре у каждого объемного понятия-ассоциации пересечение схем ассоциированных понятий не пусто.*

Доказательство. Пусть ассоциация N_a образована от понятий N_1, N_2, ..., N_n. Так как понятие N_a обеспечивает навигацию между сущностями ассоциированных понятий по значениям их признаков, принадлежащий связи $\ln k N_a$, то при пустом пересечении схем ассоциируемых понятий $\ln k N_a = \varnothing$. Следовательно, каждая сущность некоторого ассоциированного понятия N_x (а их более чем одна, так как понятие N_a объемно) связана со всеми сущностями других ассоциированных понятий N_y ($y \neq x$). Последнее является следствием того, что пустое множество признаков рассматривается как принадлежащее любому их множеству.

Следовательно, экстенсионал N_a равен декартову произведению экстенсионалов ассоциированных понятий. Однако по определению 2.4 экстенсионал N_a не равен декартову произведение экстенсионалов ассоциируемых понятий. Получили противоречие, которое доказывает свойство. ♦

2.5. Синтаксическая теория понятий

Поставим целью разработать формальную теорию понятий, которую будем использовать для достижение максимальной объективности в описании понятий и их абстракций. Универсумом формальной теории понятий является множество понятийных структур, порождаемых соответсвующей алгеброй, а ее предметом – сами понятия, отображения абстрагирования и способы выражения экстенсионалов.

Всякая ***формальная теория*** определяется формальным языком, порождающим формулы, имеющими смысл с точки зрения этой теории, и совокупностью теорем, интерпретируемых в некоторой предметной области как выполнимые (имеющие место).

Для ***конструктивного***[25] построения формальной теории Φ:

– фиксируется конечный алфавит знаков T, называемый терминальным;

– определяется перечислимое[26] множество формул F, каждая из которых является строкой в алфавите T;

[25] Здесь и далее под конструктивностью понимается существование эффективной процедуры (алгоритма), предназначенной для решения той или иной задачи, например, задания множеств, определения принадлежности элемента множеству, и др.

– выделяется разрешимое[27] множество аксиом A, являющееся подмножеством множества формул F, $A \subset F$;

– задается конечное множество правил вывода P, которые рассматриваются как вычислимые отображениями различной конечной местности i на множестве формул F, позволяющие получать новые теоремы на базе имеющихся,

$$P \subset (\bigcup_{i=0}^{\infty} F^i \to F),$$

где F^i – i-я декартова степень множества F.[28]

Выводом формулы f в формальной теории Φ из формул-посылок φ_1, φ_2, ..., φ_n называется последовательность формул f_1, f_2, ..., f_m такая, что $f = f_m$, а любая формула f_j ($j = \overline{1, m}$) есть либо аксиома, либо одна из формул-посылок φ_i ($i = \overline{1, n}$), либо непосредственно выводима из формул f_1, f_2, ..., f_{j-1} по одному из правил вывода.

Собственно формальной теорией называется множество теорем, которое замкнуто относительно правил вывода. Формальная теория называется ***разрешимой***, если существует конструктивная процедура, которая по любой строке в алфавите T позволяет судить, является ли эта строка формулой теории или нет.

2.5.1. Полнота и непротиворечивость

Основными свойствами любой формальной теории являются ее полнота и непротиворечивость.

Полнота формальной теории рассматривается как свойство, характеризующее достаточность для каких либо целей ее выразительных средств. Для любой интерпретации теории необходимо отображение, устанавливающее соответствия между формулами феории и сущностями (объектами) предметной области, которую в этом случае называют ***областью интерпретации*** теории.

---

[26] Множество называется (рекурсивно-)***перечислимым***, если существует алгоритм, порождающий это множество, т.е. такой алгоритм, который последовательно выдаёт элементы данного множества (быть может с повторениями) и только их. Предполагается, что любой элемент множества рано или поздно будет получен. Очевидно, что всякое разрешимое множество перечислимо. Обратное неверно, поскольку существую примеры перечислимых, но не разрешимых множеств.

[27] Множество называется ***разрешимым***, если существует единый способ, позволяющий относительно любого элемента определить, принадлежит он этому множеству, или нет. Формальным уточнением этого понятия является понятие рекурсивного множества – такого множества, для которого существует (формальный) алгоритм, разрешающий это множество, т.е. дающий для любого элемента ответ, принадлежит он этому множеству или не принадлежит.

[28] В абстрактных формальных системах конструктивные средства для перечисления множества формул могут не задаваться. В этом случае предполагается, что произвольная строка в алфавите T является формулой теории. Тем самым неявно предполагается наличие некоторой конструктивной процедуры перечисления или распознавания произвольных строк над этим алфавитом.

В прикладных теориях значения исходных терминов[29] даны с самого начала, т.е. интерпретацию данной теории полагают фиксированной. В рамках таких теорий возможны рассуждения о выводимости всех требуемых формул и о соответствии таких формул свойствам прикладной области интерпретации. Полнота формальной теории в данном случае называется ***семантической полнотой***.

В математических теориях, конструируемых на основаниях формальной аксиоматики, значения исходных терминов остаются неопределенными во время вывода. В этом случае формальная теория называется ***семантически полной относительно заданной интерпретации***, если из нее могут быть выведены все формулы, необходимые и соответствующие этой интерпретации.

Наряду с понятием семантическим полноты определяется и другое ее понятие, которое рассматривается как внутреннее свойством формальной теории, не зависящее ни от одной из ее интерпретаций. Формальную теорию называют ***синтаксически полной***, если порождаемое ею множество формул достаточно для произвольной области интерпретации.[30] Отсутствие четких критериев синтаксической полноты в общем случае привело к поиску такой области интерпретации, которая могла бы стать минимальным фрагментом любой другой области.

Такая область была найдена и названа областью ***логической интерпретации***. Понятие логической истины достаточно определенно сформулировал Лейбниц [146]. Он назвал предложение логически истинным, если оно истинно во всех «мирах», т.е. во всех интерпретациях. Это означает, что логика не содержит никаких фактических истин, относящихся к какому-либо конкретному миру. Несмотря на то, что в прикладных теориях формулы интерпретируются не только как истинные и ложные, но также – неопределённые, бессмысленные, модальные и т.п., при логической интерпретации требуется обойтись только двузначной логикой, т.е. такой логикой, в которой каждая формула либо истинна, либо ложна. Именно такое понимание логики проводит чёткую границу между ло-

[29] Термин – слово (знак), обозначающий точно (однозначно) определенное понятие предметной области.

[30] Определение синтаксической полноты формальной теории в общем смысле возможно только на основе мощностных оценок множеств формул теории и множества сущностей области интерпретации. В частности, могут быть выделены конечные, счетные и континуальные теории и соответствующие им области. Однако в этом случае определение соответствия между сущностями области интерпретации и формулами теории, как правило, является нетривиальной задачей, которая не всегда поддается конструктивному решению. Поэтому в абстрактных формальных теориях основной акцент делается на выявления свойств множеств формул, а именно, определяется их разрешимость и перечислимость.

гическими и содержательными истинами, что существенно необходимо для построения формальных теорий, и в частности, самой математической логики.[31]

Формальную теорию называют ***дедуктивно полной***, если для всякой формулы, выводимой в данной теории, можно показать ее истинность (является в таком случае теоремой).[32] Понятно, что в этом случае подразумевается логическая интерпретация, которая ставит в соответствие каждой формуле значение ее истинности.

Интерпретация становится ***моделью*** для формальной теории, если все ее теоремы истинны в данной интерпретации. По этой причине всякая формальная теория, как правило, содержит в качестве своего фрагмента формальную логику, т.е. логические аксиомы и правила вывода, благодаря которым становится возможным рассматривать формальные ***доказательства***, т.е. конструктивными средствами показывать выводимость требуемых формул (теорем).

С логической интерпретацией тесно связано понятие непротиворечивости. Формула называется ***общезначимой***, если она логически истинна, и ***противоречивой***, если она логически ложна, т.е. ее отрицание общезначимо. Формальная теория логически ***непротиворечива***, если все ее теоремы не являются противоречиями. Однако доказательство непротиворечивости средствами самой теории не означает ее действительной непротиворечивости, поскольку в противоречивой теории всегда доказуема её непротиворечивость. Для доказательства непротиворечивости необходима другая формальная теория, а потому такое доказательство лишь сводит вопрос о непротиворечивости одной теории к непротиворечивости другой.

---

[31] Особую роль в математической логике имеет понятие семантической полноты исчисления предикатов первого порядка, где семантическая полнота понимается в том смысле, что в исчислении доказуемы все общезначимые формулы и только они. Это отличает формальную логику от других теорий. В итоге, основной смысл исчисления предикатов – обеспечение истинности формул при всех возможных интерпретациях. Благодаря этому замечательному свойству логика предикатов широко применяется во всей математике, несмотря на некоторую ограниченность её языка. Для прикладных теорий рассматривается либо синтаксическая полнота, либо полнота относительно каких-либо специальных семантик [258]. Последнее связано с тем, что такие теории основаны на расширении логики предикатов с общей тенденцией приблизить язык формальной теории к естественному языку. Однако, если иметь в виду, что понятие истины присуще всем возможным «мирам», то вводимые в прикладные теории дополнительные понятия таковыми не являются и требуют явного определения своей семантики.

[32] Во многих формальных теориях имеется различие между множеством выводимых формул и множеством теорем. В этом случае формальная теория дополняется конструктивными средствами, позволяющими получить произвольную теорему. При логической интерпретации теоремой называется формула, выводимая из аксиом и интерпретируется как истинная, а конструктивные средства для этого предоставляются логическими правилами вывода. Заметим, что в математической логике распознание в произвольной формуле теоремы исчисления предикатов первого порядка неразрешимо [30].

Заметим, что результаты, полученные Гёделем исключают возможность проверки полноты для таких основополагающих теорий, как арифметика или теория множеств [30]. Отрицательные теоремы Гёделя утверждают, что если формальная арифметика первого порядка непротиворечива, то она неполна и даже принципиально ***непополнима*** в том смысле, что в любом непротиворечивом аксиоматизируемом расширении арифметики существует неразрешимая замкнутая теорема, т.е. недоказуемая формула, отрицание которой также не доказуемо. Поскольку каждая замкнутая формула при логической интерпретации либо логически истинна, либо логически ложна, то это означает, что существует истинная арифметическая формула, не доказуемая в формальной арифметике.

Хотя доказательство непротиворечивости формальной арифметики стало возможным в ее расширении, при котором к аксиомам арифметики добавляется трансфинитная индукция, или конструктивное правило Карнапа [165, т. 2, с. 728], неполнота арифметики так и осталась научным фактом. По этой причине у достаточно сложных формальных теорий, имеющих логическую интерпретацию, различают два вида полноты: синтаксическую и семантическую, что связано с разделением языка таких теорий на синтаксическую и семантическую части и, как следствие этого, наличием двух видов доказательств.[33] К первому виду относятся доказательства, которые касаются только синтаксических свойств теории в логической интерпретации. Второй вид – это семантические доказательства, апеллирующие к содержательным интерпретациям языка теории [258].

Заметим, что семантические доказательства содержат обращение к семантике, которая, как правило, не является аксиоматизируемой теорией. Считается, что для всякого семантического доказательства можно получить его синтаксический аналог, если формализовать соответствующую содержательную теорию (тезис Гильберта [12, с. 49]). В этом случае для каждого семантического доказательства строят его синтаксический эквивалент путём аксиоматизации фрагмента содержательной теории, содержащего посылки доказательства. При этом открытым остается вопрос о непротиворечивости всей совокупности таким образом полученных доказательств, разрешение которого может оказаться эквивалентным доказательству непротиворечивости всей содержательной теории.

2.5.2. Семантические инварианты

Уточнение понятия истины с помощью средств логической семантики осуществлено А. Тарским [218]. Им показано, что термин «истинно» выражает только свойство наше-

[33] В частности, в математической логике различают исчисление предикатов и алгебру предикатов. Исчисление предикатов является синтаксической теорией и используется в синтаксических математических теориях. Множество же общезначимых формул алгебры предикатов является семантической теорией предикатов и определяет логические средства, используемые в математике.

го знания, в частности, свойство высказываний, а не объективной действительности. Следовательно, инвариантность истины в различных областях интерпретации проистекает не из свойств этих областей, а из свойств, присущих мышлению как таковому. После такого уточнения правомерным становится вопрос: существуют ли другие такие инварианты?

Очевидно, поиск ***семантических инвариантов*** следует осуществлять в области получения, переработки и представления знаний. В самом общем виде знание может быть определено как проверенный на практике результат субъективного отражения объективной действительности, представляемый понятиями и суждениями, утвержденными некоторой последовательностью умозаключений. Единственная известная и семантически полная формальная теория – исчисление предикатов первого порядка – акцентирует свое внимание на правилах выражения суждений и построения на их основе умозаключений. Однако столь же общими для всех областей интерпретации видятся не только правила вывода, сохраняющие истинность, но и правила образования и выражения понятий.

Таким образом, на основе формализации способов образования понятий, а именно: абстракций обобщения, типизации, ассоциации и агрегации, может быть построена формальная теория, претендующая, как и исчисление предикатов, на семантическую инвариантность во всех «мыслимых мирах». Однако, в отличие от исчисления предикатов, имеющего в качестве семантического инварианта область логической интерпретации, формальную теорию понятий определим с учетом другого семантического инварианта – ***области понятийной интерпретации***.

2.5.3. Формальная теория понятий

Формальную теории понятий будем строить на основе алгебры понятийных структур, которая предназначена для выражения способов абстрагирования понятий и дополненная описанием интенсионалов понятий, задающих конструктивные процедуры для разрешения их экстенсионалов.

Определение 2.9. *Формулой формальной теории понятий будем называть понятийную структуру, которая дополнена описанием интенсионалов всех входящих в нее понятий.*

Так как в рассмотренном ранее формализме понятийной структуры предусмотрены средства для выражения всех известных абстракции, то определение синтаксической полноты теории будем осуществлять путем выявления условий, при которых такое определение становится неприменимым. Как это имеет место и в исчислении предикатов, будем различать полноту и непротиворечивость самой формальной теории. Однако в противоположность исчислению предикатов, введем в использование понятия полноты и непроти-

воречивости понятийной структуры и ее интенсионалов, выражаемой формулой этой теории.

Определение 2.10. *Формулу формальной теории понятий будем называть* ***синтаксически полной****, если в ней отсутствуют понятия, которые не содержат ни одной сущности, т.е. экстенсионал которых пуст.*

Заметим, что синтаксическая неполнота формулы в указанном смысле делает понятийную структуру неприменимой в любой области интерпретации. Последнее основано на том, что если в некоторой предметной области мыслится некоторое понятие, то это понятие должно иметь как минимум одну сущность, ему принадлежащую. В противном случае такое понятие не может быть образовано при абстрагировании и не может быть использовано для описания предметной области.

В частности, непосредственно из определения 2.10 следует, что если у некоторого понятия схема пуста, то это свидетельствует о синтаксической неполноте понятийной структуры, так как из пустоты схемы следует и невозможность определить хотя бы одну сущность, принадлежащую экстенсионалу этого понятия, кроме разве что пустого понятия, не содержащего ни одной сущности.

Определение 2.11. *Формулу формальной теории понятий будем называть* ***семантически полной****, если в ней определены все понятия, способы их абстрагирования и выражения, необходимые для описания заданной области интерпретации.*

Таким образом, определение семантической полноты формулы теории понятий связано с выявлением достаточности ее понятий для описания некоторой предметной области. Очевидно, что семантическая полнота конкретной формулы не может быть определена средствами самого формализма и требует обращения к области интерпретации. При этом необходимо выполнить содержательный анализ предметной области, решаемой задачи и метода ее решения, а также привести семантическое доказательство ее полноты. Далее будем считать, если не установлено обратное, то в понятийной структуре определены все необходимые понятия, т.е. рассматриваемые понятийные структуры будем считать семантически полными.[34]

Заметим, что понятие непротиворечивости только для понятийной структуры введено быть не может. Если понятийная структура дополнена способами выражения сущностей, принадлежащих каждому из ее понятий, то непротиворечивость понятийной структуры должна быть установлена и на основе выявления противоречий, связанных с принадлежностью сущностей экстенсионалам понятий, рассматриваемых как разрешимые множества, где разрешающая процедура задается описанием интенсионала. Таким обра-

[34] Методика построения семантически полных понятийных структур рассмотрена в 2.7.

зом, понятие противоречивости может быть полностью определено только для формулы формальной теории понятий, которая помимо понятийной структуры содержит и описания интенсионалов.

Определение 2.12. *Формула формальной теории понятий называется* ***противоречивой****, если в ней описана сущность, выраженная как принадлежащая, так и как не принадлежащая экстенсионалу одного и того же понятия.*

Определение непротиворечивости осуществим путем установления противоречий в описании схем и интенсионалов понятий, что возможно финитными средствами по причине конечности и разрешимости этих множеств. Заметим, что разрешающая процедура для экстенсионалов понятия выражается его интенсионалом.

Определение 2.13. Формальную теорию понятий *будем называть:*

– ***непротиворечивой****, если все выводимые в теории формулы непротиворечивы;*

– ***семантически полной****, если для любой области интерпретации существует в теории вывод соответствующей ей формулы;*

– ***синтаксически полной****, если для любой выводимой в теории формулы существует область интерпретации;*

– ***разрешимой****, если существуют конструктивные средства для распознавания формул, выводимых в теории.*

Из данного определения видно, что понятия полноты и непротиворечивости формальной теории понятий несколько отличаются от принятых в математической логике. Специфика определения 2.13 обусловлена тем, что универсум формальной теории понятий имеет большую общность, чем универсум математической логики. Более того, с последним не совпадает и включает последний в качестве своего фрагмента (частного случая).

Для обоснования данного выше определения непротиворечивости, следует заметить, что «в сознании нет отрицательных функций. Не видеть чего-нибудь – это значит видеть что-нибудь другое или это значит слышать, думать, чувствовать что-нибудь определенное... Только если в действительной картине предмета есть признаки, которые исключают ожидаемую картину, я могу сказать, что ожидаемой картины действительно нет. Таким образом, отсутствие только тогда может служить основанием для отрицательного суждения, когда оно сводится к несовместимости» [38]. В итоге, существует некоторый общий принцип, выражающий фундаментальное свойство мышления, частным проявлением которого является, в том числе, и логический закон противоречия – ***принцип несовместимости***. Формулировка этого принципа, заключающаяся в том, что невозможно мыслить некоторую сущность, которая в одной и той же степени принадлежит, и одно-

временно не принадлежит одному и тому же понятию, положена в основу определения непротиворечивости формальной теории понятий.[35] В частности, для понятия логической истины могут существовать сущности (высказывания), которые описаны как принадлежащие, так и как не принадлежащий экстенсионалу этого понятия. В этом случае имеем противоречие, понимаемое в классическом смысле.

В формальной теории понятий, по аналогии с исчислением предикатов, в качестве семантической теории выступает алгебра понятийных структур, поставляющая синтаксической части теории понятий стандартную область интерпретации. Поэтому для оценки состоятельности теории по описанию полных и непротиворечивых понятийных структур введено понятие семантической полноты, а для оценки невозможности получения описаний неполных и противоречивых понятийных структур – понятие синтаксической полноты.[36]

Таким образом, формальная теория понятий синтаксически полна, если не требуется специальным образом устанавливать состоятельность (выполнимость) любой выведенной в ней формулы, т.е. все выводимые формулы описывают понятийные структуры, для которых сущестуют области интерпретации (возможно мыслимые). С другой стороны, если для всех предметных областей существует вывод соотвествующих им формул (понятийных структур), то такая теория понятий семантически полна.

В итоге, область логической интерпретации следует рассматривать как соответствующую конкретному предметному содержанию, связанному с понятием логической истины. С другой стороны, полнота и непротиворечивость теории в области логической интерпретации является частным случаем полноты и непротиворечивости этой теории в области понятийной интерпретации, а через нее – для соотвествующего фрагмента произвольной предметной области (рис. 2.10).

Стандартной интерпретацией формальной теории понятий является понятийная область (семантическая теория понятий), относительно которой известно, что она принадлежит некоторой предметной области, которая, в свою очередь, структурирована на понятийном уровне и включает способы образования и выражения понятий. Таким образом,

---

[35] В исчислении предикатов показано, что позитивно образованные формулы являются более выразительным и эффективным средством представления и обработки знаний, чем используемые в современных интеллектуальных системах логического типа языки хорновских дизъюнктов. Позитивно-образованные формулы представляются строкой, состоящей из типовых кванторов с некоторым заключительным утверждением безкванторного типа. Последнее позволяет строить более эффективные процедуры логического вывода [39].

[36] Несмотря на все старания, достичь в достаточно сложной формальной теории такой идеальной ситуации, когда синтаксическая часть теория описывает все сущности предметной области и только их, как правло, не удается [173, с. 132].

формальная теория понятий, как и исчисление предикатов, не зависит от своих предметных расширений и может быть включена любое из них. В этом случае предполагается, что любая предметная область допускает в качестве своего минимального фрагмента область понятийной интерпретации, а через нее, в случае необходимости, может содержать и подобласть логической интерпретации.

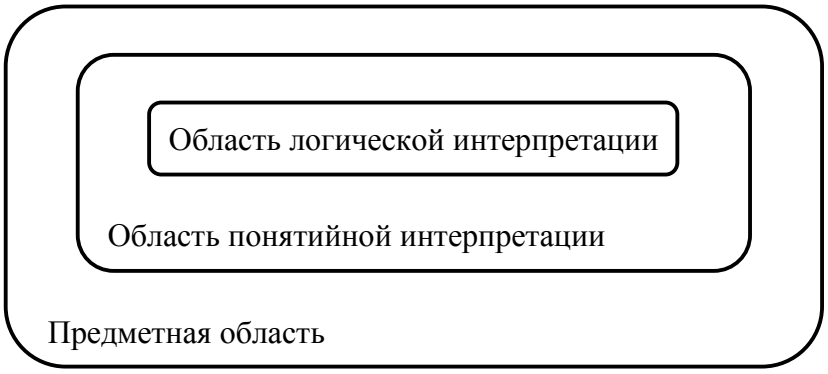

Рис. 2.10. Области интерпретации

В заключении заметим, что введенное понятие синтаксической полноты формальной теории понятий является некоторым обобщением синтаксической полноты исчисления высказываний в узком смысле. Непротиворечивая теория называется ***синтаксически полной в узком смысле***, если добавление к ее аксимомам любой невыводимой в этой теории формулы с сохранением всех правил вывода, приводит к противоречивой теории [251, с. 154].

Утверждение 2.5. *Если формальная теория понятий непротиворечива и синтаксически неполна в узком смысле в области логической интерпретации, то она является синтаксически неполной и в область понятийной интерпретации.*

Доказательство. Действительно, пусть имеется формальная теория понятий T, которая непротиворечива, семантически и синтаксически полна. Пусть f – любая невыводимая в теории T формула. Добавим f множество аксиом этой теории и рассмотрим полученную при этом новую теорию T'. Поскольку T' содержит все аксиомы и правила вывода теории T, то в ней выводимы все формулы теории T. Так как в новой теории формула f – аксиома, то эта формула в ней также выводима. Однако формула f не может иметь области интерпретации, так как все области интерпретации уже описаны в теории T и выражены соотвествующими формулами, т.е. между формулами этой теории и областями интерпретации установлено взаимно-однозначное соотвествие. Остается толь-

ко предположить, что формула f – противоречива, т.к. не имеет области интерпретации. Следовательно, противоречива и теория T'.[37] ♦

2.5.4. Исчисление понятий

Исчисление понятий определим как исчисление понятийных структур, а саму понятийную структуру будем задавать множеством формул, каждая из которых выражает образование одного из ее понятий. Таким образом, универсумом исчисления понятий является множество понятийных структур, а ее предмет – понятия и отображения абстрагирования.

Алфавит. Алфавит исчисления включает знаки понятий N_1, N_2, ...; знак неопределенности $\neg$, знаки операций объединения $\cup$, пересечения $\cap$, строгого $\supseteq$ и нестрогого $\supset$ включения, круглых скобок), (.

Аксиома. Подразумевается, что исчисление понятий строится на базе теории множеств и включает ее аксиоматику.[38] Единственной дополнительной аксиомой, добавляемой в исчислении понятий к аксиоматике теории множеств, является аксиома существования пустого понятия, которое определяется как понятие, имеющее пустое множество признаков, пустой интенсионал и экстенсионал:

$$0)\ \frac{()()}{()},$$

где для обозначения понятия использована дробь: в числителе дроби между парами фигурных скобок задается имя понятия (у пустого понятия имя отсутствует) и способы его образования – списки дифференциации (первая пара скобок) и интеграции (вторая пара скобок), а в знаменателе – схема понятия, которая для пустого понятия также пуста.

---

[37] Может возникнуть возражение, что в рассмотренном доказательстве не следует использовать такое сильное свойство теории, как ее семантическая полнота. Однако последнее не стало препятствием для доказательства синтаксической полноты в узком смысле исчисления высказываний [251, с. 154], где для такого доказательства потребовалось привлечь свойство семантической полноты этого исчисления по отношению к алгебре высказываний, а множество тождественно-истинных формул алгебры высказываний было объявлено как семантическая теория, достаточная для описания произвольных предметных областей.

[38] Так как рассматриваемые далее множества не являются множествами более высоких типов (не содержат себя в качестве своих элементов), то используемый фрагмент теории множеств косвенно гарантирует от проникновения в формализм противоречий. Считается, что парадоксы теории множеств связаны только с понятием множества всех множеств, а также с самоприменимыми предикатами [258]. В частности, понятие множества всех множеств не признается математическим объектом потому, что оно не удовлетворяет такому естественному свойству всякого множества, как возможность его расширения.

Заметим, что самая простая понятийная структура состоит из одной аксиомы. Более того, по определению аксиома содержится в любой понятийной структуре и сохраняет ее полноту и непротиворечивость.

Правила вывода. Понятийную структуру будем представлять формулой, состоящей из подформул. Правила вывода зададим в виде вычислимых отображений подформул-посылок в подформулу-заключение. При этом посылками могут стать только подформулы, которые выведены ранее и принадлежат множеству подформул понятийной структуры. Заключение вывода – это новая подформула, которая включается в формулу понятийной структуры.

Таким образом, выводом в исчислении понятий будем называть получение формулы понятийной структуры S, состоящим из конечной последовательности подформул f_1, f_2, ..., f_m, которая по окончанию вывода образуется множество подформул понятийной структуры $S = \{f_1, f_2, \ldots, f_m\}$, причем каждая подформула f_i объявляет некоторое понятие N_i и задает способ его абстрагирования.

В исчислении понятий будем использовать следующие четыре правила вывода, представленные в виде схем, или метаформул:

1) $\dfrac{()()}{()} \overset{\neg N}{\mapsto} \dfrac{()N()}{(N)}$;

2) $\dfrac{(\ldots)N_1(\ldots)}{\operatorname{shm} N_1} \ldots \dfrac{(\ldots)N_m(\ldots)}{\operatorname{shm} N_m} \underset{\bigcap\limits_{i=1}^{m} \operatorname{shm} N_i \supset ()}{\overset{\neg N}{\mapsto}} \dfrac{(N_1 \ldots N_m)N()}{\bigcap\limits_{i=1}^{m} \operatorname{shm} N_i}$;

3) $\dfrac{(\ldots)N_1(\ldots)}{\operatorname{shm} N_1} \ldots \dfrac{(\ldots)N_m(\ldots)}{\operatorname{shm} N_m} \underset{\bigcup\limits_{i=1}^{m} \operatorname{shm} N_i \supset ()}{\overset{\neg N}{\mapsto}} \dfrac{()N(N_1 \ldots N_m)}{\bigcup\limits_{i=1}^{m} \operatorname{shm} N_i}$;

4) $\dfrac{(\ldots)N_1(\ldots)}{\operatorname{shm} N_1} \ldots \dfrac{(\ldots)N_m(\ldots)}{\operatorname{shm} N_m} \underset{\bigcap\limits_{i=1}^{t} \operatorname{shm} N_i \supseteq \bigcup\limits_{j=t+1}^{m} \operatorname{shm} N_j \supset ()}{\overset{\neg N}{\mapsto}} \dfrac{(N_1 \ldots N_t)N(N_{t+1} \ldots N_m)}{\bigcup\limits_{j=t+1}^{m} \operatorname{shm} N_j}$.

Каждое правило имеет левую и правую части. В левой части (до знака вывода $\mapsto$) задаются посылки, в правой части, после знака вывода, – заключение. Указание условий, при которых правило может быть применено, осуществляется над и под знаком вывода. По своей сути условия – это дополнительные посылки, выраженные на языке теории множеств и которые должны выполняться до применения правил.

Приведенные схемы описывают порождение множества всех формул исчисления, которые получаются, если в правой части вместо подформул-посылок записать выведенные ранее подформулы, а метапеременную N заменить на знак определяемого понятия.

Метапеременные N_i и $\mathrm{shm}\, N_i$ $(i = \overline{1, m})$ выражают соответственно знаки понятий и их схемы из подформул-посылок. Заметим, что в левой части правил одна и та же подформула может повторятся любое число раз.

Так как для установления полноты и непротиворечивости формальной теории понятий, необходимо помимо понятийной структуры иметь и описания интенсионалов понятий, далее будем предполагать, что интенсионалы понятий, если не заданы, сохраняют полноту и непротиворечивость формул исчисления понятий. Это необходимо для исследования тех свойств исчисления, которые могут привести к невозможности полного или непротиворечивого определения интенсионалов.

Простые понятия. Правило вывода 1) служит для порождения первичных понятий, или понятий-признаков, схема которых тривиальна и состоит из самого порождаемого понятия. При применении правила необходимо обеспечить невозможность вывода одного и того же понятия-признака дважды, что непосредственно следует из свойств понятийных структур (см. 2.4.4). Последнее достигается проверкой условия применения правила, которое задано в виде знака определяемого понятия с операцией неопределенности $\neg$. Ввиду конечности множества подформул будем предполагать существование конструктивного средства для выполнения этой операции, осуществляющей просмотр ранее порожденных подформул и сравнение знаков уже определенных понятий со знаком нового понятия N.

Первое правило сохраняет непротиворечивость и полноту понятийной структуры. Предполагается, что любое последующее определение интенсионала понятия N является непротиворечивым и синтаксически полным, т.е. интенсионал описывает не менее одной сущности (полнота) и одна и та же сущность не описана как принадлежащая, так и как не принадлежащая этому понятию (непротиворечивость).

Если схема понятия N будет пуста, то обеспечение синтаксической полноты при описании интенсионала станет невозможным. Следовательно, полнота понятийной структуры может следовать только из непустой схемы порождаемого понятия-признака, что является необходимым условием. Достаточность следует из посылки о существовании синтаксически полного описания интенсионала понятия N. Заметим, что формальные средства для выражения интенсионалов в исчислении не рассматриваются.

Непротиворечивость понятийной структуры после применения правила следует из ее начальной непротиворечивости и отсутствия какого-либо иного описания интенсионала понятия N, могущего привести к противоречию. Для этого достаточно потребовать отсутствия в понятийной структуре понятия с тем же именем.

Дифференциальные понятия. Правило вывода 2) служит для порождения понятий на основе дифференциации понятий. Условием применения правила является отсутствие определения понятия N в множестве подформул понятийной структуры и непустое пересечения схем дифференцируемых понятий N_1, ..., N_m, которое становится схемой образуемого понятия N. Так как схемы понятий – конечные множества, следовательно, существуют конструктивные средства для определения их пересечения, которое, заметим, выполняется с учетом повторения элементов.

Второе правило, как и первое, сохраняет непротиворечивость и синтаксическую полноту понятийной структуры. Полнота следует из непустой схемы порождаемого дифференциального понятия, а непротиворечивость – из отсутствия в понятийной структуре понятия с тем же именем.

Интегральные понятия. Правило 3) необходимо для порождения понятий на основе интеграции понятий. Условие применения правила обеспечивает, помимо единственности подформулы описания понятия N, невозможность создания понятий на основе интеграции только пустых понятий. Схемой образуемого понятия N становится объединение схем интегрируемых понятий N_1, ..., N_m, которое, как и при дифференциации, выполняется с учетом повторения элементов.

Рассматриваемое правило сохраняет непротиворечивость и синтаксическую полноту понятийной структуры: полнота, как и ранее, следует из непустой схемы интегрального понятия, а непротиворечивость – из отсутствия в понятийной структуре понятий с тем же именем.

Сложные понятия. Особую роль в формализме играет правило 4), которое служит для задания понятий, которые образуются одновременно при дифференциации и интеграции. Посылками правила служит множество подформул, образующих два подмножества – подмножество подформул, участвующих в дифференциации понятия, и подмножество подформул, задающих его интеграцию. Разделение посылок определяется параметром правила t, где t – количество подформул-посылок, участвующих в дифференциации. Оставшиеся $m-t$ посылок применяются для интеграции понятия. Понятия, образованные путем одновременной дифференциации и интеграции других понятий будем называть ***сложными понятиями***.

Необходимость правила для образования сложных понятий вызвана тем, что абстракция обобщения определена в (2.4) при строгом пересечении схем обобщаемых понятий. Однако в предметной области могут существовать понятия, которые образованы не только путем расширения экстенсионала образуемого понятия, но и при ограничении его схемы. С другой стороны, абстракция ассоциации в (2.6) не позволяет определить родови-

довые связи образуемого понятия, что также ограничивает выразительные возможности понятийной структуры и может привести к ее семантической неполноте. Поэтому образование понятий путем одновременной интеграции и дифференциации признаков является необходимым выразительным средством исчисления.

Таким образом, правило 4) позволяет определить дифференциальное понятие с ограничением его схемы до схемы специфицированного интегрального понятия. При этом сохраняется возможность для вновь образованного понятия одновременно выразить и обратные абстракции, а именно, абстракцию конкретизации, противоположную абстракции обобщения, и абстракцию индивидуализации, противоположную абстракции ассоциации. Более того, при определении сложного понятия его интегральная часть позволяет специфицировать содержательный состав понятия, непосредственно не выводимый из дифференциальной части.

Условия применения правила обеспечивают, как и ранее, единственность формулы для понятия N, а также ограничения, которые необходимо выполнить для обеспечения синтаксической полноты и непротиворечивости соответствующей формулы теории понятий. Схемой образуемого понятия N становится непустое объединение схем интегрируемых понятий N_{t+1}, ..., N_m, которое обязательно должно принадлежать пересечению схем дифференцируемых понятий N_1, ..., N_t.

Полнота и непротиворечивость. Как показано выше правила вывода исчисления понятий сохраняют синтаксическую полноту и непротиворечивость каждой выводимой в исчислении понятийной структуры. При этом аксиома рассматривается как минимальная понятийная структура, являющаяся синтаксически полной и непротиворечивой по определению. Следовательно, каждая выведенная в исчислении понятийная структура является ее теоремой. Отсюда может быть показана справедливость следующего утверждения.

Утверждение 2.6. *Исчисление понятий полно и непротиворечиво.*

Доказательство. Непротиворечивость исчисления понятий следует из ранее показанной непротиворечивости аксиомы и сохранения непротиворечивости понятийной структуры при применении каждого правила вывода.[39]

Для доказательства семантической полноты исчисления предположим, что существует область интерпретации, не имеющая выводимой в исчислении понятийной структуры. Следовательно, в этой области имеется хотя бы одно понятие, отсутствующее в неко-

[39] Заметим, что аналогичный метод доказательства использовался для обоснования непротиворечивости самого исчисления предикатов, при котором все формулы исчисления предикатов однозначно отображены в формулы алгебры предикатов. При этом показано, что все аксиомы исчисления переходят в тавтологии (тождественно истинные формулы), а правила вывода сохраняют тавтологичность формул [251, с. 161].

торой понятийной структуре, которая соответствует максимальному фрагменту области интерпретации, а такой всегда фрагмент существует, так как пустое понятие принадлежит любой области интерпретации. Используя правила вывода 1)-4) включим недостающее понятие в такую структуру.[40] Повторяя описанную процедуры для всех отсутствующих понятий, осуществим вывод понятийной структуры. Следовательно понятийная структура для рассматриваемой области существует. Получено противоречие, обосновывающее семантическую полноту исчисления понятий.

Покажем теперь синтаксическую полноту исчисления. Предположим, что существует выводимая в исчислении понятийная структура, которая не имеет области интерпретации. Так как исчисление обеспечивает выражение всех абстракций понятий для любых областей интерпретации, в том числе и «мыслимых», то сделанное предположение возможно только, если понятийная структура противоречива, что делает ее логически неприменимой в любой области, или синтаксически неполна, т.е. включает одно или несколько понятий с пустым экстенсионалом.

Однако, как показано выше, в исчисление понятий любой вывод сохраняет синтаксическую полноту и непротиворечивость понятийных структур. Следовательно, рассматриваемая понятийная структура синтаксически полна и непротиворечива. Получено противоречие, которое доказывает синтаксическую полноту исчисления.

Так как исчисление понятий оказалась синтаксически и семантически полным, делаем вывод о его полноте. ♦

Пример 2.22. Рассмотрим понятийную структуру, заданную множеством формул исчисления понятий:

$$S = \{\ ()()\ \ ()N_1()\ \ ()N_2()\ \ ()N_3(N_1N_2)\ \ (N_1N_3)N_4()\ \ (N_3)N_5(N_4)\ \}.$$

Построим вывод понятийной структуры S:

Шаг 1. $\dfrac{()()}{()} \mapsto \dfrac{()N_1()}{(N_1)}$ – использованы аксиома 0) и правило 1);

[40] Достаточность используемых в исчислении понятий четырех абстракций для выражения любой понятийной структуры имеет такое же обоснование, как и достаточность логических связок и кванторов исчисления предикатов для выражения всех высказываний относительно любой области интерпретации. В традиционной логике имеется четыре закона, составляющие ее фундамент: законы тождества, противоречия, исключенного третьего и закон достаточного основания. Многие десятилетия идут не утихающие споры о природе этих законов. Часть специалистов считает, что эти законы отражают глубинные приемы нашего мышления. Иная точка зрения состоит в том, что мы принимаем их для того, чтобы с их помощью получать знание, имеющее общезначимый характер, где общезначимое достоверно, а сама эта достоверность фиксируется как особое состояние непосредственного чувства очевидности [195].

Шаг 2. $\dfrac{()()}{()} \mapsto \dfrac{()N_2()}{(N_2)}$ – использованы аксиома 0) и правило 1);

Шаг 3. $\dfrac{()N_1()}{(N_1)}\dfrac{()N_2()}{(N_2)} \mapsto \dfrac{()N_3(N_1\ N_2)}{(N_1\ N_2)}$ – использовано правило 3);

Шаг 4. $\dfrac{()N_1()}{(N_1)}\dfrac{()N_3()}{(N_3)} \mapsto \dfrac{(N_1\ N_3)N_4()}{(N_1)}$ – использовано правило 2);

Шаг 5. $\dfrac{()N_3()}{(N_1\ N_2)}\dfrac{(N_1\ N_3)N_4()}{(N_1)} \mapsto \dfrac{(N_3)N_5(N_4)}{(N_1)}$ – использовано правило 4).

В итоге имеем,

$$S = \left\{ \begin{array}{ccccccc} \dfrac{()()}{()} & \dfrac{()N_1()}{(N_1)} & \dfrac{()N_2()}{(N_2)} & \dfrac{()N_3(N_1\ N_2)}{(N_1\ N_2)} & \dfrac{(N_1\ N_3)N_4()}{(N_1)} & \dfrac{(N_3)N_5(N_4)}{(N_1)} \end{array} \right\},$$

откуда заключаем, что понятийная структура S полна и непротиворечива. ♦

Разрешимость. Под разрешимость формальной теории обычно понимется возможность определить для произвольной формулы, выводима или нет эта формула из множества аксиом теории. Известно, что свойством разрешимости обладает исчисление высказываний, но не обладает исчисление предикатов [251, с. 157].

Аналогично разрешимость в формальной теории понятий определена как возможность установить принадлежность произвольной строки множеству формул (понятийных структур), выводимых в этой теории (определение 2.13 на с. 101). Покажем разрешимость исчисления понятий.

Утверждение 2.7*. Исчисление понятий разрешимо.*

Доказательство. Разрешающая процедура для формулы исчисления понятий начинается с последовательной проверки применимости правил вывода теории к подстрокам проверяемой строки, полученным ее разделением на части. В этом случае каждая подстрока описывают образование одного из понятий. Получение таких подстрок и проверка применимости правил вывода к каждой подстроке и в порядке следования этих подстрок разрешимо.

Следующим шагом разрешающей процедуры является вычисление схем, осуществляемое по рекуррентной процедуре, описанной в 2.4.3 на с. 89. Так как исходная строка конечна, то и конечно число неперекрывающихся подстрок, на которые она делится. Следовательно эта процедура за конечное число шагов или остановится, если в проверяемой строке не будет найдено циклов, или будет обнаружет цикл, для чего также потребуется конечное число шагов.

Отсюда заключаем, что исчисление понятий разрешимо.♦

2.5.5. Логическая теория понятий

Ранее непротиворечивость и полнота исчисления понятий установлена исходя из предположения о непротиворечивости и полноте интенсионалов понятий. Последнее может быть обеспечено использованием для выражения интенсионалов известных непротиворечивых теорий, например, исчисления предикатов, формальной арифметики, и др.[41]

Логическую теорию понятий определим как расширение исчисления понятий за счет дополнения формализма исчислением предикатов первого порядка, которое используется для выражения интенсионалов.

Для синтаксической полноты понятийной структуры требуется непустота схемы и существования хотя бы одной сущности, для которой интенсионал выполним. Это равносильно выводимости интенсионала понятий в исчислении предикатов и его выполнимость при каких-либо значениях предметных и предикатных переменных, т.е. формула интенсионала должна быть не тождественно-ложной. Заметим, что определение тождественной ложности формул в исчислении высказываний разрешимо, что следует из существования эффективной процедуры, которая по любой формуле позволяет судить, является ли она теоремой исчисления высказываний или нет [251, с. 155].

Однако проблема разрешимости исчисления предикатов решается отрицательно [251, с. 163],[42] т.е. не существует эффективной процедуры, выполнение которой гарантировало бы, что если имеется теорема исчисления предикатов, то существует конечный шаг этой процедуры, на котором она будет доказана, т.е. множество теорем исчисления (тавталогий) не является рекурсивно перечислимым. По этой причине формальная теория понятий, построенная на основе исчисления предикатов, будет обладать синтаксической неполнотой. Последнее является следствием того, что проверка формулы на противоречивость вынуждено сводится к проверке тавтологичности ее отрицания.[43]

---

[41] Заметим, что чистое исчисление предикатов включает в качестве своего фрагмента исчисление высказываний. Известны также непротиворечивые расширения исчисления предикатов, содержащие исчисление с операцией равенства и формальную арифметику [139, с. 234].

[42] Впрочем как и проблема синтаксической полноты в узком смысле относительно алгебры предикатов.

[43] Нерезрешимость исчисления предикатов следует назвать не синтаксической, а ***семантической неразрешимостью***. По своему определению язык исчисления предикатов первого порядка является контекстным языком (см. приложение 2, с. 340) и может быть синтаксически распознан линейно-ограниченным автоматом, что, собственно, и происходит всякий раз в фазе синтаксического анализа в известных языках логического программирования. Однако семантическая интерпретация исчисления – проверка тавтологичности формул, или проверка выполнимости формул относительно логической интерпретации, является неразрешимой проблемой и не может быть осуществлена в общем виде в рамках алгоритмического подхода. Имеются также серьезные вычислительные трудности проверки выполнимости формул, или проверки семантической разрешимости фор-

Но имеется один частный случай, относительно которого исчисление предикатов разрешимо – это ***одноместное*** исчисления предикатов первого порядка [139, с. 244]. Следовательно, только одноместное исчисление предикатов может служить формальным средством для синтаксически полного выражения интенсионалов понятий. В противном случае определение синтаксической полноты произвольной формулы логической теории понятий становится неразрешимым, как и нерезрешимым определение принадлежности произвольной строки формулам этой теории. Из разрешимости исчисления понятий, рассматирваемом как исчисление понятийных структур (утверждение 2.7 на с. 110), получаем справедливость следующего утверждения.

Утверждение 2.8. *Логическая теория понятий является разрешимой относительно одноместного исчисления предикатов.*

При использовании непротиворечивых формул одноместного исчисления предикатов для выражения интенсионалов понятий все формулы логической теории понятий также будут непротиворечивы. Однако в отличии от исчисления понятий, где все формулы предполагаются синтаксически полными, в логической теории, без принятия соответствующих мер, возможно порождение формул, которые приводят к синтаксической неполноте понятийной структуры.

Для обеспечения синтаксической полноты логической теории дополним правила вывода 1)-4) еще одним условием, которое устанавливает отсутствие противоречия при выражении интенсионала каждого понятия.[44] Очевидно, для такой проверки не требуется знание области интерпретации, так противоречие является тождественно ложной формулой во всех интерпретациях. Следовательно одноместное исчисление предикатов сохраняет и синтаксическую полноту логической теории. После такого уточнения и с учетом утверждения 2.6 показана справедливость следующего утверждения.

Утверждение 2.9. *Логическая теория понятий является непротиворечивой и синтаксически полной относительно одноместного исчисления предикатов.*

Важным отличием логической теории от исчисления понятий является расширение понятия семантической полноты формул на интенсионалы понятий. Так как каждый интенсионал, по своей сути, задает разрешающую процедуру для экстенсионала соответствующего понятия, то проверка семантической полноты формулы логической теории должна выполняться не только путем определения семантической полноты конкретной понятийной структуры, но и путем анализа на семантической полноту формул исчисления

мул относительно предметной интерпретации, что связано с известной проблемой вычислимости областей применимости квантификаторов [263].

[44] ***Противоречием*** называется формула логики предикатов, которая ложна при всех интерпретациях, т.е. является тождественно-ложной.

предикатов. В последнем случае требуется, чтобы каждый интенсионал выражал все сущности, принадлежащие соответствующему понятию, и только их.

В итоге, открытым остался вопрос о семантической полноте логической теории, который может быть переформулирован следующим образом: существуют ли экстенсионалы понятий произвольной предметной области, которые не выражаются на языке одноместного исчисления предикатов. Иными словами, существует ли высказывание о принадлежности той или иной сущности экстенсионалу некоторого понятия, которое является истинным в предметной области, но не выразимым в одноместном исчислении предикатов.

Общий ответ на этот вопрос оказывается отрицательным даже для полного исчисления предикатов, несмотря на то, что исчисление предикатов является непротиворечивой и семантически полной формальной теорией относительно алгебры предикатов. Однако известны примеры перечислимых, но не разрешимых множеств, например, множество самоприменимых алгоритмов [140, с. 221]. Другой пример – упомянутое ранее множество формул исчисления предикатов первого порядка, которое перечислимо, но не разрешимо. Последнее означает, что существуют множества, которые можно задать в виде перечисляющей процедуры, но не существует процедуры, которая позволяет определяет принадлежность произвольного элемента этому множеству. Перенос этого результата в логическую теорию понятий означает, что существуют понятия, экстенсионалы которых не могут быть заданы на языке исчисления предикатов.

Определение 2.14. *Областью* ***логической интерпретации*** *называется такая область интерпретации, в которой экстенсионал произвольного понятия выразим на языке одноместного исчисления предикатов.*

Таким образом, семантическую полноту логической теории понятий следует рассматривать относительно не произвольных областей интерпретации, а относительно того их подмножества, в которых интенсионал понятий представим формулами одноместного исчисления предикатов. После приведенных выше рассуждений уточним понятие семантической полноты логической теории в виде следующего утверждения.

Утверждение 2.10*. Логическая теория понятий является семантически полной относительно области логической интерпретации.*

Пример 2.23. Рассмотрим понятийную структуру из примера 2.22. Дополним эту структуру формулами исчисления высказываний, выражающими интенсионалы соответствующих понятий:

$$\frac{(\,)(\,)}{(\,)}$$

$$\frac{()N_1()}{(N_1)}\ N_1(E,\{a\,b\,c\})$$

$$\frac{()N_2()}{(N_2)}\ N_2(E,\{d\,e\})$$

$$\frac{()N_3(N_1\,N_2)}{(N_1\,N_2)}\ N_1(E,\{a\,c\})\,\&\,N_2(E,\{d\})\vee N_1(E,\{b\})\,\&\,N_2(E,\{e\})$$

$$\frac{(N_1\,N_3)N_4()}{(N_1)}\ N_1(E,\{a\})$$

$$\frac{(N_3)N_5(N_4)}{(N_1)}\ N_1(E,\{b\})$$

где E – предметная переменная, обозначающая произвольную сущность области интерпретации, а нотация вида $N(E,U)$ обозначает предикат, задающий принадлежность сущности E понятию N, который выполнимым, если $E \in U$.[45]

Из формул примера следует:

– понятие N_1 описано как простое, а его интенсионал выражает три сущности a, b и c, составляющие экстенсионал этого понятия;

– понятие N_2 является простым и имеет экстенсионал, состоящий из сущностей d и e;

– понятие N_3 интегрирует понятия N_1, N_2, образовано на сущностях ad, cd, be и имеет ограниченны экстенсионал, т.е. является ассоциацией;

– понятие N_4 является дифференциальным, выражается понятиями N_1 и N_3, причем такими их сущностями, которые имеют признак N_1, равный a;

– понятие N_5 является сложным, образовано как дифференциал понятия N_3 и интеграл понятия N_4, представляется понятием N_4 и выражается сущностями понятий N_3, которые имеют признак N_1, равный b.

В итоге, по формулам, выражающим интенсионалы понятий, находим их экстенсионалы: $\mathrm{ext}\,N_1 = \{a\,b\,c\}$, $\mathrm{ext}\,N_2 = \{d\,e\}$, $\mathrm{ext}\,N_3 = \{ad\,cd\,be\}$, $\mathrm{ext}\,N_4 = \{a\,ad\}$, $\mathrm{ext}\,N_5 = \{be\}$.

В данной понятийной структуре все интенсионалы выражены не тождественно ложными формулами исчисления предикатов. Следовательно эта понятийная структура

[45] Формально нотация вида $N(E,U)$ не является одноместным предикатом. Однако U, в отличие от предметной переменной E, является предметной константой. Следовательно, константу U можно рассматривать как один из индексов предиката N. Таким образом, использование указанной нотации не выводит нас из области одноместного исчисления предикатов, а служит лишь для повышения выразительности описания интенсионалов.

синтаксически полна и непротиворечива. Для установления ее семантической полноты требуется расширение области понятийной интерпретации на предметную область, для описания которой эта понятийная структура предназначена. После такого расширения необходимо показать, что в описанной понятийной структуре указаны все понятия области интерпретации, а интенсионалы этих понятий выражают соответствующие им множества сущностей. ♦

Как видно из примера логическая теория, хотя и является семантически полной в области логической интерпретации, однако может породить семантически неполные понятийные структуры (формулы). Это связано с тем, что, как и в исчислении предикатов, работающем с одним понятием – понятием логической истины, которое присуще всем мыслимым мирам, так и в формальной теории понятий, которая работает со всеми мыслимыми понятиями, требуется индивидуальное доказательство интерпретируемости каждой выведенной формулы в заданной предметной области. Такое доказательство не может быть получено в формальной теории, так как для его осуществления необходимо привлечение содержательных представлений относительной конкретной области интерпретации.

Последнее объясняется тем, что при описании предметной области синтаксическими средствами теории, которые «запасены» в избытке, могут быть использованы различные предметные константы (пропозиционные буквы, имена понятий) и, как следствие этого, возникает закономерный вопрос о существовании интерпретации этих предметных констант. Иными словами, всякий раз требуется доказывать, что каждое описание, полученное в рамках формальной теории, ***корректно***, т.е. имеет какое-то отношение к описываемой предметной области.

По этой причине интерпретация одной формальной теории в другую формальную теорию требует задания третьей формальной теории, которая представляет собой правила построения формул, рекурсивно перечисляющие все пары формул двух формальных теорий, между которыми устанавливается соответствие. В этом случае корректность интерпретации может быть доказана средствами той формальной теории, с помощью которой установлена интерпретация, т.е. средствами которой заданы правила рекурсивного перечисления пар соответствующих формул двух формальных теорий.[46]

[46] Интерпретация теоретических построений развитых областей научного знания носит, как правило, опосредованный характер и включает в себя многоступенчатые, иерархические системы промежуточных интерпретаций. Связь начального и конечного звеньев таких иерархий обеспечивается тем, что интерпретация интерпретаций какой-либо теории дает и непосредственную ее интерпретацию. Интерпретация формальной теории в объекты реального мира представляет собой её интерпретацию посредством мыслительного аппарата человека. В этом случае понятия и суждения естественнонаучных теорий интерпретируются посредством образов сознания (идеальных сущностей), совокупность которых должна быть адекватна интерпретируемой теории.

В итоге имеем, что логическая теория понятий позволяет полно и непротиворечиво описать произвольные предметные области, принадлежащие только области логической интерпретации. Причем, в этих описаниях не требуется перечисления всех сущностей, входящих в экстенсионалы (а это и невозможно по причине объемности реальных понятий). Разрешающая процедура для экстенсионалов понятий задается их интенсионалами, выраженными формулами одноместного исчисления предикатов. Однако рассмотренный формализм неприменим для описания достаточно сложных предметных областей, что видится следствием низких выразительных качеств используемого логического языка.[47]

2.5.6. Контекстно-свободная теория

Расширим формализм исчисления понятий на относительно произвольную формальную систему, которую будем использовать для выражения интенсионалов. Для такого расширения воспользуемся грамматической формой (см. 2.2.4 на с. 66). ***Контекстно-свободную теорию понятий*** определим как расширение исчисления понятий, выполненное на основе формализма контекстно-свободных грамматик.[48]

Пример 2.24. Рассмотрим понятийную структуру из примера 2.22, которая дополнена описанием интенсионалов понятий, заданных в грамматической форме:

$()()\ \ 'ac'|'cb'$

$()N_1()\ \ 'a'|'a'N_3\,|'b'N_1N_1$

$()N_2()\ \ 'b'|'b'N_3\,|'a'N_2N_2$

$()N_3(N_1N_2)\ \ 'b'N_1\,|'a'N_2\,|"ab/ba"N_4$

---

[47] Логический анализ фраз естественного языка на уровне одноместных предикатов был характерен для аристотелевской логики. Это существенно ослабляло «выразительные возможности» логики и служило препятствием для адекватной формализации тех объективных связей между предметами, которые мыслимы в виде отношений между соответствующими понятиями. Устранение указанного препятствия и усиление выразительных средств формализма современной логики связано с новой трактовкой предиката в работе Г. Фреге «Исчисление понятий» (1879) [287]. Главная идея этой трактовки – рассмотрение отношения предикации как частного случая функциональной зависимости: «понятие есть функция одного аргумента, значением которой всегда является истинностное значение» [229]. Это обеспечивает более ёмкое, чем аристотелевское, отображение смысловой структуры фраз естественного языка в формализме субъектно-предикатного типа и одновременно дальнейшее развитие самого этого формализма на пути повышения его выразительности. Однако, как в математическом анализе существуют функции многих переменных, которые не имеют конечного разложения по функциям одной переменной, так и в естественном языке существуют фразы, которые не представимы через одноместную предикацию.

[48] Известно использование регулярных языков для выражения понятий [134], где простое понятие представляется перечнем признаков, а сложные понятия образуются путем применения операций алгебры регулярных событий: конкатенации (мультипликации), объединения (альтерации), кратной конкатенации (возведения в степень), итерации. Однако выразительные возможности такого подхода значительно слабее возможностей, предоставляемых контекстно-свободными грамматиками.

$(N_1N_3)N_4(\ )\ \ 'c'N_5$

$(N_3)N_5(N_4)\ \ N_4''N_4$

где, например, для выражения пустого понятия заданы две строки: ac и cb, а пустое понятие, заданное в виде пустых одинарных кавычек, применено для выражения экстенсионала понятия N_5. ♦

Следует сказать, что грамматическая форма, по сравнению с логической, предоставляет более выразительные средства, так как позволяет описать интенсионалы понятий правилами вывода контекстно-свободной грамматики. В этой связи заметим, что в логической форме принадлежность признаков понятий множеству их допустимых значений выражается в виде высказываний, которые могут быть описаны как один из частных случаев грамматического представления.

Утверждение 2.11. *Контекстно-свободная теория понятий является непротиворечивой.*

Доказательство. Рассмотрим произвольное правило $N \to \alpha$ порождающей контекстно-свободной грамматики $< T, W, P, I >$, где N – нетерминальный знак выражаемого понятия, α – конечная последовательность термов, терминальных шаблонов и знаков понятий из нетерминального алфавита W, T – терминальный алфавит грамматики, P – множество правил вывода грамматики, I – ее аксиома.

Для доказательства непротиворечивости выражения интенсионалов понятий в контекстно-свободной форме отобразим правила вывода контекстно-свободной грамматики в формулы исчисления предикатов.

Для этого выполним эквивалентные преобразования грамматики, после которого правила вывода P приобретают вид: $N \to \delta$ или $N \to \Delta$, где δ – строка терминальных знаков, а Δ – строка нетерминальных знаков. Последнее возможно благодаря введению новых нетерминальных знаков, которые выражают все терминальные строки, встречающиеся в правых частях правил вывода.

Далее зафиксируем некоторый нетерминальный знак N и пронумеруем все правила, его выражающие. Если имеется правило вида $N \to \delta$ под номером j, где δ – строка терминальных знаков, то сопоставим этому правилу формулу $\varepsilon(E, \delta) \to N^{(j)}(E)$, где ε – предикат, устанавливающий равенство терминальных строк E и δ, $\to$ – знак импликации. В свою очредь j-му правилу вывода грамматики вида $N \to \Delta$ сопоставим формулу

$$\exists x_1 x_2 \ldots x_n N_1(\sigma(E, x_1)) \,\&\, \mathop{\&}_{i=1}^{n-1} N_i(\varphi(E, x_i, x_{i+1})) \,\&\, N_n(\pi(E, x_n)) \to N^{(j)}(E),$$

где N_i – предикаты, соотвествующие i-му нетерминальному понятию из строки Δ, & – знак конъюнкции, а σ, φ и π – вычислимые функторы, возвращающие соответственно префикс, подстроку и суффикс строки E, задаваемые целочисленными переменными-аргументами x_i.

Результирующий предикат для понятия N будет иметь вид:

$$\bigvee_{(j)} N^{(j)}(E) \to N(E),$$

где $\vee$ знак дизъюнкции.

Повторим описанную процедуру для всех нетерминальных понятий грамматики. В итоге получим описание грамматики в исчислении предикатов первого порядка. Предметная интерпретация такого описания – распознавание строк терминальных знаков, выражающих нетерминальные понятия формального языка.

Известно, что исчисление предикатов первого порядка непротиворечиво [251, с. 161]. Отсюда заключаем, что грамматическая форма выражения интенсионалов также непротиворечива, т.е. в контекстно-свободной грамматике невозможно вывести строку, выражающую сущность некоторого понятия, и одновременно показать невыводимость этой строки. ♦

Утверждение 2.12. *Контекстно-свободная теория понятий разрешима.*

Доказательство. Разрешимость формул, описывающих понятийную структуру, показана ранее при доказательстве утверждения 2.7 на с. 110. Для установления разрешимости формул контекстно-свободной теории понятий необходимо дополнительно показать разрешимость определения синтаксической полноты интенсионалов понятий, выраженных правилами вывода контекстно-свободной грамматики.

Разрешимость проблемы определения синтаксической полноты интесионалов следует из теоремы о разрешимости проблемы пустоты контекстно-свободных языков и выявления иррелевантных (не применяемых при выводе) правил [205, с. 65].

Действительно, синтаксическая полнота правил вывода грамматики может быть установлена по процедуре приведения контекстно-свободной грамматики к форме, не содержащей бесполезных нетерминальных знаков [140, с. 308]. Если окажется, что нетерминальный знак N будет исключен из множества нетерминальных знаков грамматики W, а соответствующие ему правила вывода удалены из множества продукций P, то это свидетельствует о неполноте интенсионала понятия N.

Таким образом показано, что контекстно-свободная теория понятий разрешима. ♦

Пример 2.25. Рассмотрим понятийную структуру из примера 2.24. По описанию понятийной структуры и интенсионалов понятий построим контекстно-свободную грам-

матику, с учетом того, что каждое понятие-дифференциал также может быть выражено дифференцируемым понятием, а все нетерминальные знаки достижимы,[49] так как могут быть использованы для выражения соответствующих им понятий. В итоге имеем $G =< T, W, P, I >$, где $T = \{a\ b\ c\}$, $W = \{N_0\ N_1\ N_2\ N_3\ N_4\ N_5\ I\}$, N_0 – пустое понятие,

$$P = \begin{cases} N_0 \to ac \mid cb, \\ N_1 \to a \mid aN_3 \mid bN_1N_1, \\ N_2 \to b \mid bN_3 \mid aN_2N_2, \\ N_3 \to aN_1 \mid aN_2 \mid abN_4 \mid baN_4, \\ N_4 \to cN_5 \mid N_1 \mid N_3, \\ N_5 \to N_4N_4 \mid N_4N_0N_4 \mid N_3, \\ I \to N_0 \mid N_1 \mid N_2 \mid N_3 \mid N_4 \mid N_5. \end{cases}$$

Так как все нетерминальные знаки достижимы из аксиомы, для приведения грамматики осталось выявить производящие нетерминальные знаки.[50] Последнее делается по следующей рекуррентной процедуре.

Начальное приближение множества производящих знаков Q зададим из тех нетерминальных знаков, которые непосредственно порождают терминальные строки, $Q = \{N_1\ N_2\}$. Каждое следующее приближение будем строить путем просмотра множества продукций, предполагая, что нетерминальные знаки из множества Q порождают терминальные строки. Завершается процедура, когда множество Q на очередной итерации осталось неизменным.

В итоге получаем $Q = \{N_0\ N_1\ N_2\ N_3\ N_4\ N_5\ I\}$, т.е. все нетерминальные знаки грамматики производящие, а значит и их интенсионалы синтаксически полны.[51] ♦

Утверждение 2.13. *Контекстно-свободная теория понятий является синтаксически полной.*

Доказательство. Для доказательства синтаксической полноты предположим, что существует выводимая в контекстно-свободной теории понятий понятийная структура S, которая не имеет области интерпретации. Это возможно только, если в понятийной структуре S имеется противоречие, делающее ее неприменимой, или одно из ее понятий имеет пустой экстенсионал. Однако в соответствии с утверждением 2.11 контекстно-свободная

---

[49] В теории формальных грамматик ***достижимым***, или выводимым, называется такой нетерминальный знак N, для которого существует вывод $I \Rightarrow \alpha N \beta$, где α и β – некоторые строки в объединенном алфавите терминальных и нетерминальных знаков, возможно пустые.

[50] Нетерминальный знак N называется ***производящим***, если существует вывод $N \Rightarrow \gamma$, где γ – строка в терминальном алфавите, возможно пустая.

[51] Если не все нетерминальные знаки окажутся производящими, то делается вывод о неполноте интенсионалов тех понятий, которые не принадлежащих множеству Q.

теория является непротиворечивой, следовательно непротиворечива и понятийная структура S.

Для запрета появления понятий, имеющих пустой экстенсионал, правила вывода контестно-свободной теории дополняются проверкой на непустоту подъязыка, описываемого каждым ее понятием. Такая проверка может буть осуществлена, так как для контекстно-свободных грамматик разрешима проблема определения пустоты языка (см. утверждение 2.12).

Таким образом, существует область интерпретации M, соответствующая понятийной структуре S. Получаем противоречие, которое доказывает синтаксическую полноту контекстно-свободной теории. ♦

Для исследования семантических свойств контекстной теории введем понятие бесконтекстной области интерпретации.

Определение 2.15. *Областью* ***бесконтекстной интерпретации*** *называется такая область интерпретации, в которой экстенсионал произвольного понятия выразим на контекстно-свободном формальном языке.*

Утверждение 2.14*. Контекстно-свободная теория понятий является семантически полной относительно бесконтекстной области интерпретации.*

Доказательство. Для доказательства семантической полноты контекстно-свободной теории понятий предположим, что существует область интерпретации M, не имеющая понятийной структуры, выводимой в контекстно-свободной теории. Однако из утверждения 2.6 о семантической полноте исчисления понятий следует, что для области M в исчислении понятий выводима понятийная структура S, включающая все понятия этой области. В силу предположения об отсутствии вывода понятийной структуры S в контекстно-свободной теории, необходимо принять, что в понятийной структуре S существуют понятия, экстенсионалы которых несравнимы с соответствующими множествами сущностей области интерпретации M.

Так как понятийная структура S содержит все понятия области интерпретации M, то несоответствие S области M определяется интенсионалами понятий, т.е. грамматическая форма по какой-то причине оказалась несостоятельной для выражения экстенсионалов. Однако мы предположили, что формализма контекстно-свободных грамматик достаточно для выражения любого экстенсионала.[52] Получили противоречие, которое доказывает семантическую полноту контекстно-свободной теории. ♦

[52] Контекстно-свободная теория понятий основана на предположении, что множества строк, выражающее сущности понятий, представимы контекстно-свободными грамматиками, т.е. интерпретация понятий не зависит от контекста их использования. Однако для понятий известны примеры

Заметим, что контекстно-свободная теория является семантически полной при бесконтекстной интерпретации понятий. Однако существуют области интерпретации, в которых понятия выражаются не абсолютно, а с учетом некоторого окружающего их контекста, т.е. одна и та же терминальная строка в одном случае выражает одно понятие, а в другом случае (в другом контексте) – другое.

2.5.7. Контекстная теория понятий

Рассмотрим исчисление понятий, в котором для выражения интенсионалов используется формализм контекстных грамматик. Получаемую в результате теорию будем называть ***контекстной теорией понятий.***

Контекстной грамматикой называется формальная система $< T, W, P, I >$, образованная конечными множествами терминальных T и нетерминальных знаков W, множеством правил вывода P вида $\alpha N \beta \to \alpha \omega \beta$ и начальным нетерминальным знаком I, с которого начинается любой вывод в грамматике, где N – нетерминальный знак, α β и $\omega \neq e$ – конечные последовательности знаков терминального и нетерминального алфавитов, e – пустая строка.

В связи с контекстной интерпретацией понятий необходимо предусмотреть некоторую форму для задания контекстов, которая была бы совместима с формализмом понятийной структуры и принятым способом выражения интенсиналов. Покажем, что произвольная контекстная грамматика может быть задана в контекстно-свободной форме, если каждое вхождение нетерминальных знаков в правые части правил грамматики сопровождается контекстом его применения.[53]

Утверждение 2.15*. Для произвольной контекстной грамматики существует эквивалентная ей грамматика, левые части правил вывода которой состоят только из одного нетерминального знака, а каждое вхождение нетерминальных знаков в правые части правил сопровождается указанием на их левые и правые контексты.*

Доказательство. Пусть задана контекстная грамматика $G =< T, W, P, I >$ с правилами вывода вида $\alpha N \beta \to \alpha \omega \beta$. Построим грамматику $G' =< T, W, P', I >$, правила вывода которой имеют вид $N \to \omega$, где каждое вхождение нетерминального знака $N \in W$ задано

контекстной интерпретации. Как будет показано далее (см. утверждение 2.16 на с. 123), для сохранения синтаксической полноты и непротиворечивости рассматриваемой теории, такое предположение является необходимым. В противном случае теория понятий становится либо неполной, либо неполной и противоречивой. Таким образом, контекстно-свободная теория понятий является полной только для бесконтекстных областей интерпретации, т.е. для таких предметных областей, которые описываются понятиями, не имеющими условной (контекстной) применимости.

[53] Содержательное обоснование этого факта для контекстной технологии обработки данных см. в 4.2.8 на с. 193.

с левым и правым контекстом, $[\alpha]N[\beta]$. Покажем, что $L(G) = L(G')$, т.е. вывод строки γ в грамматике G существует тогда и только тогда, когда существует вывод этой строки в грамматике G'.

Построим P' следующим образом. В качестве начального приближения множества P' выберем множество правил, состоящее из правил вида $N_i \to \omega$, полученных из правил $\alpha N\beta \to \alpha\omega\beta$, принадлежащих P, путем удаления контекстов.

Последовательно, для всех нетерминальных знаков N_i, $N_i \in W$ $(i = \overline{1,n})$, заменим каждое правило в P', содержащее одно вхождение N_i в правые части правил, имеющих вид $N_j \to \gamma N_i \delta$, на множество правил вида $N_j \to \gamma[\alpha_k]N_i[\beta_k]\delta$ $(k = \overline{1,m})$, количество которых равно числу правил вывода m этого нетерминального знака N_i, определяемое из P, где α_k и β_k соответсвенно k-ый левый и правый контекст N_i из правила под номером k, γ и δ – некоторые строки в объединенном алфавите грамматики, возможно содержащие бесконтекстное (незамененное) вхождение N_i.

Повторяем замену до тех пор, пока в P' содержатся бесконтекстные вхождения нетерминальных знаков в правые части правил. Так как число продукций и число нетерминальных знаков конечно, процедура завершит свою работу через конечное число шагов.

Из построения следует, что для любого вывода строки γ в грамматики G' существует вывод строки γ в грамматике G. Верно и обратное. Следовательно, грамматики G и G' эквивалентны, так как порождают один и тот же язык. ♦

Пример 2.26. Пусть задана контекстаня грамматика $G =< T, W, P, I >$, у которой $T = \{a, b\}$, $W = \{A, B, I\}$, $P = \{I \to aBa, bAa \to baa \mid bBaa, Ba \to ba \mid bAa\}$. Построим эквивалентную ей грамматику $G' =< T, W, P', I >$, правила вывода которой имеют вид $N \to \omega$, где каждое вхождение нетерминальных знаков в ω сопровождается контектом.

Начальное приближение P' равно $\{I \to aBa, A \to a \mid Ba, B \to b \mid bA\}$. Выберем знак A и заменим каждое правило с этим знаком на множество правил по числу контектов этого знака в P. После замены имеем

$$P' = \{I \to aBa, A \to a \mid Ba, B \to b \mid b[b]A[a]\}.$$

Повторяя замену до тех пор, пока имеются бесконтексные вхождения нетерминальных знаков в P', получим

$$P' = \{I \to a[]B[a]\,a, A \to a \mid []B[a]a, B \to b \mid b[b]A[a]\}.$$

Для вывода $I \to aBa \to abAa \to abaa$ в грамматике G построим соотвествующий ему вывод в грамматике G': $I \to a[]B[a]a \to ab[b]A[a]a \to abaa$. ♦

Таким образом, каждый нетерминальный знак правила вывода контекстной грамматики слева и справа будем сопровождать квадратными скобками, внутри которых задавать соответственно левый и правый контекст. В случае, если понятие не имеет левого (правого) контекста, квадратные скобки будем опускать или, при неоднозначности сопоставления скобок нетерминальным знакам, задавать пустыми квадратными скобками.

Например, в правиле $N_1 \to \gamma[\alpha]\, N_2\, [\beta]\delta$ понятие N_1 выражается через понятие N_2 с левым контекстом α_2 и правым контекстом β_2. Указанный контекст понятия N_2 означает, что суффикс последовательности терминальных и нетерминальных знаков $\alpha_1\gamma$ выражается как α_2, а префикс последовательности терминальный и нетерминальных знаков $\delta\beta_1$ – как β_2, где α_1 и β_1 – соответственно левый и правый контекст, в котором выражено понятие N_1.

Пример 2.27. Рассмотрим понятийную структуру из примера 2.22, которую дополним описанием интенсионала понятия N_5, заданное в контекстной форме:

$$()()\ \ 'ac'|'cb'$$

$$()N_1()\ \ 'a'|'a'N_3\,|'b'N_1N_1$$

$$()N_2()\ \ 'b'|'b'N_3\,|'a'N_2N_2$$

$$()N_3(N_1N_2)\ \ 'b'N_1\,|'a'N_2\,|''ab/ba''N_4$$

$$(N_1N_3)N_4()\ \ 'c'N_5$$

$$(N_3)N_5(N_4)\ \ N_4''['c']N_4[]$$

где для выражения понятия N_5 используется контекстное правило, в соответствии с которым понятие N_4 имеет только левый контекст, равный c. Последнее означает, что для выражения пустого понятия, предшествующего понятию N_4, может быть использована только строка ac. ♦

Утверждение 2.16. *Контекстная теория понятий непротиворечива и синтаксически неполна.*

Доказательство. Так как нам удалось контекстные правила выражения интенсионалов понятий представить в контекстно-свободной форме, т.е. для описания интенсионалов используются только позитивные формы выражения сущностей понятий (определяем только то, что понятию принадлежит), то противоречий при выводе формул возникать не может. Это означает, что невозможно описать сущность, которая принадлежит и одновре-

менно не принадлежит одному и тому же понятию, так как описываются только сущности, принадлежащие понятию.[54]

Для доказательства синтаксической полноты контекстной теории необходимо показать, что для каждой выводимой в этой теории формулы существует область интерпретации. Так как контекстная теория оказалась непротиворечивой, то, по аналогии с доказательством теоремы о синтаксической полноте контекстно-свободной теории (теорема 2.13), можно было бы заключить, что контекстная теория понятий является синтаксически полной.

Однако в противоположность контекстно-свободной теории, в контекстной теории определение синтаксической полноты отдельной формулы, сводимое к проверке непустоты экстенсионалов ее понятий, является неразрешимой задачей. Это следует из неразрешимости проблемы определения пустоты языков, порожденных контекстными грамматиками [140, с. 331]. Последнее приводит к невозможности универсальными средствами установления полноты формул, получаемых при выводе в контекстной теории.

Таким образом, контекстная теория понятий не является синтаксически полной. т.к. не существует конструктивных средств, запрещающих вывод формул, не имеющих области интерпретации. ♦

Следствие 2.17*. Контекстная теория понятий неразрешима.*

Заметим, что класс языков, порождаемый контексными грамматиками, является разрешимым [140, с. 298]. Следовательно, для каждой конкретной формулы контекстной теории может существовать доказательство ее полноты или неполноты, но построение такого доказательства нельзя автоматизировать. Если такое доказательство и существует, то осуществляется всякий раз по разному, с учетом вида правил вывода исследуемой грамматики.

Теперь, по аналогии с логической и контекстно-свободной теориями, дадим определение области контекстной интерпретации.

Определение 2.16. *Областью* ***контекстной интерпретации*** *называется область интерпретации, в которой экстенсионал произвольного понятия выразим на контекстно-зависимом формальном языке.*

Утверждение 2.18*. Контекстная теория понятий является семантически полной относительно контекстной области интерпретации.*

---

[54] Формальное доказательство непротиворечивости контекстной теории понятий может быть дано по аналогии с доказательством непротиворечивости контексно-свободной теории (утверждение 2.11 на с. 117), которое осуществляется путем отображения правил вывода контекстной грамматики, заданных в контекстно-свободной форме, в формулы исчисления предикатов.

Доказательство. Так как мы заведомо ограничили класс областей интерпретации областью контекстной интерпретации понятий, то по аналогии с доказательством утверждения о семантической полноте контекстно-свободной теории заключаем, что контекстная теория является семантически полной относительно этой области. ♦

Следует указать на одно немаловажное открывшееся обстоятельство. Так как все выводимые в контекстной теории формулы непротиворечивы, а множество выводимых формул покрывает область контекстной интерпретации, что следует из ее семантической полноты, то можно говорить о синтаксической полноте контекстной теории ***в слабом смысле***, предусматривающей индивидуальной доказательство синтаксической полноты каждой выведенной в этой теории формулы.

Заметим также, что в отличие от рассмотренных ранее теорий, в контекстной теории понятий возможно появление семантических неоднозначностей, связанное со сложной структурой правил грамматики, делающих невозможной непосредственную (содержательную) проверку выполнимости формул в заданной предметной области.

Однако контекстная теория имеет средства для определения семантической содержательности своих формул, так как для любых семантически несовместимых строк, выражающих то или иное понятие, может быть установлена их выводимость. Последнее следует из разрешимости неукорачивающих языков и эквивалентности класса языков, порождаемых некорачивающими грамматиками, классу контекстных языков [140, с. 298].

Утверждение 2.19. *Других непротиворечивых (синтаксически полных) формальных теорий понятий не существует.*

Доказательство. Возможности дальнейшего наращивания выразительных качеств формализма для описания интенсионалов понятий ограничиваются неразрешимостью языков, порожденных произвольными грамматиками [140, с. 299].[55] Последнее приводит к невозможности ***универсальными*** конструктивными средствами установления непротиво-

[55] Следует заметить, что класс произвольных языков является классом перечислимых, но неразрешимых языков. Перечисляющая процедура в этом случае задается произвольными правилами вывода. Для таких языков характерно ***полносвязное*** выражение сущностей предметной области, которое уже не обладает свойством композиционности (см. определение на с. 55). В этом случае становится невозможным независимая (контексно-свободная) или даже условная (контекстно-зависимая) интерпретация фрагментов текста, так как такое описание эквивалентно порождению произвольной строки-смысла и ее заданные изменения (трансформации) другими правилами грамматики, возможно даже семантически неэквивалентные. По этой причине такие грамматики иногда называются трансформационными [289].

речивости (синтаксической полноты) формул, что свидетельствует о потенциальной противоречивости (синтаксической неполноте) таких теорий.[56] ♦

Таким образом, контекстно-свободную теорию понятий можно назвать ***частной***, так как она является полной и применима к описанию только некоторое подмножества областей интерпретации, названных бесконтекстными. В свою очередь, контекстную теорию понятий, применимую для описания более широкого класса областей – областей контекстной интерпретации, следует назвать ***общей***, так как она является самой выразительной из всех непротиворечивых теорий, однако требует поиска индивидуального доказательства синтаксической полноты каждой выводимой формулы.

В заключении укажем отношения между областями интерпретации, в которых различные теории понятий являются семантически полными или, иными словами, для каких областей интерпретации эти теории предназначены (рис. 2.11).

Рис. 2.11. Стандартные области интерпретации формальной теории понятий

Эти отношения непосредственно следуют из отношений классов языков, порождаемых грамматиками различных типов [234, с. 162]. Самым обширным классом является класс предметных областей, содержащий контекстную область интерпретации. Этому классу принадлежит, но с ним не совпадает, класс предметных областей бесконтекстной интерпретации. В свою очередь, предметные области с логической интерпретацией образую класс, который включен в класс бесконтектных областей. Последнее следует из того,

[56] Однако могут существовать формализмы для описания интенсионалов понятий, выразительные возможности которых соответсвуют классу перечислимых, но неразрешимых языков. Примером такого формализма может служить то же исчисление предикатов первого порядка. Но доказательство непротиворечивости и полноты таких формализмов приходится выполнять каждый раз по своему, с учетом вида продукций соотвествующей порождающей грамматики (с учетом аксиом и правил вывода формальной теории). Так как формальная теория понятий, по своей сути, является метаформализмом и используется для задания широкого класса формальных теорий проблемного выражения интенсионалов понятий, то доказательство ее непротиворечивости, полноты и разрешимости должно осуществляться без учета конкретного вида тех языковых конструкций, которые могут быть использованы для описания содержания и объема конкретных понятий.

что исчисление предикатов описывается одной из контекстно-свободных грамматик (см. формальную грамматику на рис. П2.1, с. 343).

2.5.8. Проблема существенной неполноты

Формальный метод, заключающийся в фиксации конечного алфавита, исходного разрешимого множества аксиом и конечного множеста правил вывода, а также в определении эффективной процедуры вывода формул, оказался применимым только для задания перечислимых множеств строк конечной длины, что связано с рекурсивной перечислимостью множества формул формальных теорий [140, с. 278].

Однако на любом конечном алфавите существует более чем счетное множество языков, понимаемых как произвольные множества строк конечной длины. Последнее следует из счетности множества таких строк, заданных в конечном алфавите, и более чем счетной мощности множества их подмножеств (теорема Кантора) [116].[57]

Таким образом, существует континуум языков, которые назовем ***неперечислимыми***, и которые не имею конструктивных средств порождения (распознавания) принадлежащих им строк. Более того, найти такие средства в принципе не представляется возможным. Следовательно, неперечислимые языки непосредственно не могут быть использованы для выражения экстенсионалов в формальной теории понятий, впрочем, как и в любой другой теории, претендующей на конечность средств описания предметной области.

Этот факт нашел свое отражение в известной теореме К. Геделя о неполноте [125, с. 383], заключающийся в существовании формул, которые не только не могут быть выведены в достаточно богатых формальных теориях, но и для которых не может быть также показана их невыводимость. Более того, любая такая формула, будучи добавленной в множество аксиом формальной теории, не делает эту теорию полной.

По этой причине исследования выразительных возможностей формальных теорий пошло в направлении расширения множества аксиом.[58] Однако на этом пути мы сталкива-

---

[57] Теорема утверждает, что никакое множество не эквивалентно множеству всех подмножеств этого множества, где эквивалентность, или равномощность, понимается в смысле существования взаимно однозначного отображения одного множества в другое.

[58] Необходимо упомянуть наличие двух основных подходов к определению множества аксиом: формульный подход, основанный на задании конечного числа аксиом относительно фиксированных переменных, а также в использовании специального правила подстановки, позволяющего фиксированные переменные заменить на другие формулы теории; и метаформульный подход, при котором аксиомы задаются схемами аксиом над метапеременными, понимаемыми в смысле возможности подстановки вместо них произвольных формул, при этом правило подстановки в теорию не включается. В последнем случае, благодаря метапеременным, каждая схема аксиом выражает бесконечное число вариантов применения аксиомы. Однако как в первом, так и втором случае число самих аксиом следует признать конечным, так как все множество аксиом выражает все же конечное число некоторых закономерностей, на основе которых строится формальная тео-

емся с проблемой бесконечности, но на этот раз не с бесконечным числом формул, а с бесконечным числом самих формальных теорий, поскольку установлено, что многие конечно-аксиоматизируемые теории (исчисления) имеют бесконечное число непротиворечивых расширений. В итоге оказалось, что само множество формальных теорий образует определенную конструкцию, описываемую соответсвующей метатеорией – исчислением формальных систем, или математикой метаматематики [204].

Изучение различных множеств формальных теорий, структурированных в виде решетки, занимает все более значительное место в исследованиях и является одной из главных тенденций развития логики [120]. В частности, в результате изучения найденного многобразия формальных логик даже поставлен вопрос: имеет ли логика «какое-то отношение к мыслительной деятельности человека» [121]? Если имеет, «то тогда уровень логичности последней скрывается за «функционированием» бесконечных классов различных логических систем» [там же].

Таким образом, комбинаторные возможности конечно аксиоматизируемых теорий оказались явно недостаточными для выражения существующего многобразия формул, призванных отразить возможное многообразие предметных областей и их свойств. По этой причине предприняты исследования выразительных качеств формальных теорий в направлении предельного расширения множества аксиом, в результате чего появились теории, которые не являются конечно аксиоматизируемыми.[59] В рамках таких теорий становится возможным породить более чем счетное множество языков, однако использовать такие теории на практике не предсталяется возможным ввиду неконструктивности используемых для построения этих теорий средств. В частности, при использовании таких теорий необходимо будет предположить бесконечное число закономерностей, служащий для начального описания предметной области и из которых выводятся другие ее описания, а также бесконечную мощность множества понятий, необходимых для описания этой предметной области, что следует из бесконечности исходных посылок. Последнее не согласуется с самим принципом понятийного осмысления действительности, требующим выражения закономерностей окружающего мира через конечное число исходных понятий и суждений.

рия. По этой причине конечно аксиоматизируемые теории иногда рассматриваются как рекурсивно аксиоматизируемые [183].

[59] Например, известны теории множеств, построенные на основе формализации таких понятий как класс всех множеств, класс всех ординалов, класс всех топологических пространств, и т.д., определенные на более чем счетной сигнатуре и не являющиеся конечно аксиоматизируемыми [204, с. 229].

Следует заметить, что ход становления и развития естественных языков выработал иной механизм решения проблемы существенной неполноты конечных средств описания предметных областей. Давно замечено, что естественный язык, или U-язык, «для каждого конкретного контекста единственный: если бы нам пришлось говорить о нескольких различных U-языках, то все равно мы пользовались бы одним из них и этот язык был бы U-языком… Он не является неизменным, но постоянно находится в процессе развития: время от времени мы можем вводить новые термины и новую символику; точно также мы можем соглашаться использовать старые термины в новом смысле или же отказаться от них вообще.» [118, с. 56].

Этот механизм был ранее нами продекларирован и неявно использовался в виде утверждения о невозможности однозначной семиотической репрезентации достаточно сложных понятий. Последнее, в частности, приводит к необходимости создания специализированного предметного языка, или проблемного языка, для описания каждой проблемной области. Только при проблемном подходе, когда число проблемных языков, применяемых для описания одной и той же предметной области, ничем не ограничено, появляется возможность рассматривать все их комбинации как некоторое «интегральное» языковове средство, с помощью которого возможно порождение произвольного множества строк (в том числе и неперечислимого), предназначенного для исчерпывающего (полного) описания этой предметной области.

Иными словами, мы не используем формальную логику, основанную на противопоставлении понятий истины и лжи, для описания понятий и их взаимосвязи.[60] Мы придерживаемся позитивного подхода в выражении чего-либо: «...отрицание чего-то не есть

[60] Логический подход для формализации понятий использован в работе [173, с. 375], суть которого заключается в следующем. В каждый момент времени для каждой конкретной цели взаимоотношения понятий описываются классической логической теорией, которая называется ***ипостасью***. Причем, под понятием понимается языковая единица, имеющая четко определенный смысл. Декларируется, что понятия могут описываться лишь в их взаимосвязи, а сама совокупность взаимосвязанных понятий составляет сигнатуру (множество констант) каждой ипостаси. Для выражения семантики понятий используется логическое исчисление, позволяющее определить взаимосвязь понятий-констант в виде теорем ипостаси, являющихся, по своей сути, тавталогиями. Постулируется, что каждая ипостась имеет как подтеорию (базовая ипостась), так и свои эффективно конструируемые расширения – альтернативы (дочерние ипостаси), которые по определению являются логически несовместимыми между собой, т.е. собственные теоремы любой альтернативы теоремами базовой ипостаси не являются. Таким образом, каждая ипостась является теорией, «описывающей состояние понятий в данный момент, для данной цели и с данной точки зрения». Так как каждая ипостась перестает «работать, если мы интересуемся не истинностью в случае неизменной фиксированной точки зрения, а развитием понятий», то создается новая альтернативная ипостась, наследуемая от наиболее близкой в семантическом плане и в которой выражается новое понимание понятий и их взаимосвязи. Иными словами, путем расширения исходной теории создается решетка не имеющая максимального элемента, а минимальный элемент которой – классическая математическая логика (дерево с одним корнем и, возможно, бесконечным числом листьев).

отсутствие этого чего-то. Для фиксации такого феномена у человека нет чувственных датчиков. Отсутствие чего-то обосновывается лишь наличием того, что с этим чего-то несовместимо. Поэтому в человеческом сознании вполне укладывается картина, в которой «речка движется и не движется, вся из лунного серебра...», а в традиционной логике в этой ситуации приходится констатировать наличие противоречия» [195]. Отсюда и использование в качестве основного формализма не логических теорий, которых развелось великое множество, а единственной теории формальных языков и формальных грамматик. По этой причине актуальным видится доказательство следующего утверждения.

Утверждение 2.20. *Мощность множества языков проблемного описания предметной области, порождаемого формальными грамматиками, более чем счетна.*

Доказательство. Известно, что число конечных строк в конечном алфавите счетно. Отсюда делаем заключение о счетности множества формальных грамматик, каждая из которых представима конечной строкой над объединенным алфавитом грамматики (алфавитом терминальных и нетерминальных знаков). Таким образом, класс формальных грамматик порождает класс перечислимых языков, имеющих счетную мощность.

Следует также предположить и счетность множества проблемных областей, которые могут быть заданы на одной и той же предметной области. Последнее следует из того, что для обозначения проблемы может быть выбрана одна строка из счетного множества строк конечной длины в терминальном алфавите грамматики.

Установим некоторое взаимно-однозначное соотвествие между счетным множеством перечислимых языков и счетным множеством проблемных областей. В итоге имеем, что для описания каждой проблемной области имеется своя формальная грамматика, содержащая некоторое конечное множество терминальных знаков и некоторое множество понятий, представленных алфавитом нетерминальных знаков, а также конечное число правил вывода, описывающих интенсионалы этих понятий.

Очевидно, что одной и той же предметной области можно сопоставить любое из подмножеств множества проблемных областей, т.е. описать эту предметную область произвольной совокупностью формальных языков, соотвествующих одному из подмножеств выделенного универсума проблемных областей.

Следовательно, одна и та же предметная область может быть описана более чем счетным числом способов, так как мощность булеана счетного множества проблемных областей более чем счетна (теорема Кантора). Отсюда заключаем, что можность класса языков, которые могут быть использованы для проблемного описания одной и той же предметной области, более чем счетна. ♦

В свете вышеизложенного аксиоматический метод, как способ построения научной теории, при котором в её основу кладутся некоторые исходные положения (суждения) – аксиомы, из которых все остальные утверждения (умозаключения) этой науки – теоремы, должны выводиться чисто логическим путём, посредством доказательств, видится, в некоторой степени, несостоятельным, так как приводит к существенной неполноте как самой теории, так и знаний, представляемых в этом формализме. По этой причине ряд, состоящий из форм выражения результатов рационального познания, куда традиционно включают «понятие», «суждение», «умозаключение» и «теорию», следует дополнить новой формой, которую условно будем называть корпусом.

Традиционно понятие как форма выражения результатов рационального познания выражается в виде слов, а суждения понимаются в узком смысле и рассматриваются как предложения, выражающие понятие (логической) истины. Иными словами суждение – это взаимосвязь понятий-слов, которая имеет место быть. Умозаключение, в свою очередь, выражается последовательностью предложений-суждений, которая построена по определенным правилам. В умозаключении различают суждения-посылки и суждение-следствие. Правила построения умозаключений определяются соотвествующей теорией и обеспечивают истинность (логичность, состоятельность, убедительность) следствия при истинности (логичности, состоятельности, убедительности) посылок. В свою очередь теория служит для определения правил построения умозаключений, которые сохраняют истинность (логичность, состоятельность, убедительность) всех выводимых в ней умозаключений.

В рассматриваемом подходе формы выражения результатов рационального познания трактуются несколько шире. Основное отличие в том, что понятие представляется знаком, который не имеет своего выражения (представления). В этом случае множество суждений интерпретируется как определение и, одновременно, как способ выражения соотвествующего понятия-знака через другие понятия, задаваемые одним из способов своего выражения, т.е. суждения рассматриваются как форма выражения произвольных понятий, а не только понятия логической истины. В свою очередь умозаключение – это последовательность суждений, или текст, не выражающий никакого понятия, или выражающий «пустое» понятие. Правила выражения «пустого» понятия задаются соотвествующей (формальной) теорией, которая определяет разрешенные способы соединения понятий, представленных в одной из форм своего выражения – понятиями-суждениями, для получения допустимых (содержательных) описаний предметной области.

Тогда ***корпус***, по аналогии с соотвествующим понятием в современной корпусной лингвистике,[61] – это множество теорий, описывающих одну и ту же предметную область и предназначенных для всестороннего (полного) представления знаний, которые накоплены относительно этой предметной области в аспекте различных проблемных ситуаций. Только при корпусном подходе к описанию предметных областей становится возможным преодоление теоретических и практических проблем, связанных с существенной неполнотой и непополняемостью формальных теорий.

В отличие от решеточной и древовидной структуры множества логических теорий, при корпусном описании предметной области имеем сетевую структуру, в которой между теориями устанавливаются произвольные связи. В отличие от корпусной лигвистики, где корпус текстов служит эмпирическим материалом для построения формальной теории, объясняющей те или иные аспекты языковой практики, при корпусном подходе строится множество теорий, которые покрывают (порождают) все многообразие текстов, необходимых для описания той или иной предметной области.

Как это и наблюдается в естественных языках, по мере изучения любой предметной области видоизменяется и сам язык, предназначенный для выражения результатов познавательной деятельности, причем изменяется таким образом, чтобы обеспечить адекватность языковых средств тем идеальным объектам, которые появляются в сознании как результат отражения объективной действительности. Не всегда такие изменения являются логически непротиворечивыми. Более того, требовать такую непротиворечивость не представляется возможным в принципе по причине существенной многоаспектности (проблемности) знакового выражения понятий. Отсюда, в частности, получаем алогичность и паралогичность естественного языка, являющегося, по своей сути, множеством умозаключений взаимно противоречивых теорий. В аспекте вышеизоженного понятие, как первичная форма рационального познания, является сложным когнитивным феноменом, не имеющим однозначных и логически согласованных между собой семиотических репрезентаций.

[61] Лингвистический корпус текстов – большой, представленный в электронном виде, унифицированный, структурированный, размеченный, филологически компетентный массив языковых данных, предназначенный для решения конкретных лингвистических задач. Назначение языкового корпуса – показать функционирование лингвистических единиц в их естественной контекстной среде. Критерий отбора текстов для включения их в корпус изменяется исходя из целей исследования. В этом случае для репрезентативности корпуса требуется максимальная объективность представления изучаемого явления [206].

2.6. Сравнительный анализ формализма

В настоящем параграфе рассмотрены известные формы спецификации результатов концептуального анализа предметной области и приведен их сравнительный анализ с формальной теорией понятий, рассматриваемой как исчисление понятийных структур.[62]

2.6.1. Семантические сети

Семантические сети получили свое развитие в рамках научного направления, связанного с представлением знаний для моделирования рассуждений. Эта область исследований была ориентирована на разработку специальных языков и графических средств для представления декларативных знаний о предметной области. Результаты исследований семантических сетей в последующем были конкретизированы и успешно использованы при построении концептуальных моделей и схем реляционных баз данных.

Семантические сети появились как средство моделирования процесса ассоциирования понятий и иерархической организации знаний [337, 273]. Графы оказались хорошим средством для формализации ассоционистских теорий за счет точного представления отношений между понятиями посредством дуг и узлов. Вывод новых знаний в семантической сети делается путем прослеживания связей между понятиями.

В общем случае под семантической сетью S понимается двойка [152]

$$S =< O, R >,$$

где O – множество объектов предметной области (понятия и факты); R – множество отношений (ассоциации) между объектами.

Например, для формализации взаимосвязи понятий Г.С. Цейтин [238] предложил использование ассоциативных сетей. Ассоциативные сети содержат узлы, выражающие некоторые сущности (соответствующие объектам в тексте или во внешнем мире), и направленные дуги, соединяющие эти узлы. Содержимым узла может являться число, строка символов, процедура или конечное множество других узлов. Дуги именованы, и имена всех дуг, покидающих данный узел, должны быть различны, чаще всего они предназначены для обозначения семантических ролей.

Фреймы в этой модели представлялись в виде ассоциативной подсети, один узел которой обозначен как входной для фрейма. Порождение нового фрейма из его прототипа трактуется как копирование участка семантической сети. Важной операцией над сетью является слияние двух узлов: дуги с одинаковыми именами при этой операции сливаются.

---

[62] Анализ формальной теории понятий с позиций ее выразительных качеств по описанию интенсионалов понятий будет дан в 4.3 на с. 196.

Несмотря на то, что в семантических сетях использовано иерархическое наследование и вывод на основе ассоциативных связей, они не позволили преодолеть сложность многих проблемных областей. Одна из причин этого – ограниченность связей между понятиями как основного выразительного средства описания семантики предметной области. Большая часть связей реальных семантических сетей представляла общие ассоциации между понятиями и не обеспечивала реальной основы для структурирования семантических отношений [156].

2.6.2. Исчисление предикатов

Те же проблемы, которые имели место в семантических сетях при представлении взаимосвязи понятий, возникли и в исчислении предикатов (описание исчисления приведено в Приложении 4).

Само по себе исчисление предикатов имеет мало преимуществ перед семантическими сетями [156]. Во-первых, этот формализм слишком общий и переносит бремя борьбы со сложностью на программиста, во-вторых, отсутствуют выразительные средства для представления наследования и ассоциации понятий.

На подмножестве хорновских дизъюнктов логики предикатов основана модель логической программы в языке Пролог, где имеются встроенные механизм логического вывода с поиском и возвратом и механизм сопоставления с образцом. Язык Пролог является декларативным языком запросов, позволяющим формулировать сложные запросы в терминах переходов и состояний [274].

Особую трудность при использовании языка вызывает сокращение размерности достижимого пространства состояний, для чего решается нетривиальная задача определения ограничительных условий. Ограничения могут быть наложены как на область допустимых значений отдельных переменных, так и на различные комбинации значений предметных переменных. Другим недостатком языка являются большие требования в отношении вычислительных ресурсов, что связано с его универсальностью и декларативной природой конструкций.

Иными словами, между языком логического программирования, реализующим исчисление предикатов, и содержательными формулировками относительно предметной области существует трудно преодолимый семантический разрыв. Ограничение синтаксических средств для описания предметных областей, имеющее место в исчислении предикатов, преодолевается в формальной теории понятий на основе использования метаформализма для создания проблемных языков, предназначенных для описания каждой проблемной области в отдельности. Это, в совокупности со средствами определения семантики

таких языков (глава 3), позволяет автоматизировать процесс преодоления указанного выше семантического разрыва.

2.6.3. Теория концептуальной зависимости

Большинство последних работ по семантическим сетям посвящено ограничению типов связей между понятиями путем их стандартизации. Например, в теории концептуальной зависимости Р. Шенка и К. Ригера [335] число базовых примитивов, на основе которых определяется сеть, сокращено до четырех: действия, объекты, модификаторы действий и модификаторы объектов. В свою очередь сами примитивы рассматриваются как имеющие фиксированное число своих проявлений. Так для действий выделено всего 12 компонентов, которые могут использоваться для определения концептуальной зависимости.

Отношение концептуальной зависимости – это концептуальные синтаксические правила, которые выражают семантические связи между понятиями языка, предназначенного для того, чтобы помочь в реализации рассуждений. В рамках теории эти отношения считаются простейшими семантическими категориями, на основе которых могут быть выражен более сложный смысл. Через выделенные базовые концептуальные зависимости предпринята попытка моделирования внутреннего (глубинного) представления предложений на английском языке. Каноническое представление сети, при котором между понятиями задаются только специфицированные зависимости, использовалось:

– для устранения проблем, связанных с двусмысленностью;

– для введения семантической метрики, позволяющей отождествлять высказывания, имеющие один и тот же смысл, так как их внутренние представления синтаксически идентичны;

– для упрощения выводов, необходимых для понимания предметной области;

– для уменьшения числа операций при работе с сетью.

Однако подход, связанный с ограничением числа видов связей между узлами не позволяет выразить более тонкие понятия, играющие важную роль в предметной области. Высказывались также критические замечания по поводу вычислительной сложности преобразования предложений в набор примитивов низкого уровня [368].

Таким образом, в отличие от теории концептуальных зависимостей, понятийная структура определена не множеством отношений между понятиями, которые несут различную семантическую нагрузку, специфицированную конечным множеством примитивов, а задается четырьмя видами отображений, единственное назначение которых – показать способы абстрагирования понятий.

2.6.4. Концептуальный вывод

Некоторое обобщении теории концептуальной зависимости можно найти в работе Г.С. Плесневича [189], где рассмотрена теория силлогистик для семантических сетей. Суть подхода заключается в том, что рассматриваются произвольные семантические сети с конечным множеством типов бинарных отношений между узлами-понятиями. Путем исследования свойств этих бинарных отношений выявляются состоятельные схемы транзитивного вывода (силлогистики), состоящие из двух исходных высказываний (отношений между тремя узлами сети), и третьего высказывания, являющегося следствием первых двух. Для установления истинности высказываний задается их предметная интерпретация. На основе выявленной силлогистики для каждого множества типов отношений строится формальная теория, которая исследуется на дедуктивную полноту и замкнутость. Для пояснения предлагаемого подхода рассмотрена силлогистика Аристотеля, силлогистика с бинарными отношениями логики первого порядка и силлогистика интервальной логики.

Заметим, что описанный выше подход к концептуальному моделированию неприменим к понятийной структуре по следующим причинам:

– используемый формализм понятий и их взаимосвязи основан не на выявлении транзитивных бинарных отношениях между понятиями, а на отображениях одних понятий в другие различной местности, соотвествующих четырем формам абстрагирования понятий;

– виды отображений понятий являются фиксированными и не допускают поиск новых силлогистик – формальных схем вывода, призванных установить новые содержательные отображения понятий на основе анализа существующих.

Заметим, что понятийный анализ в рассматриваемой нами постановке не требует осуществления вывода на понятийной структуре, т.к. основным назначением понятийной структуры является выражение способов абстрагирования понятий, а не выполнение автоматического синтеза формул для их предметной интерпретации.

2.6.5. Формальный анализ понятий

Современные технологии инженерии знаний не предлагают систематической процедуры, позволяющей вывести концептуальную схему предметной области из доступных о ней данных. Некоторый конструктивный подход для решения этой задачи просматривается в теории формального анализа понятий (Formal Concept Analysis) [365].

Суть подхода заключается в следующем. Формальное понятие (концепт) представляется как двойка <объем, содержание>, где объем – некоторое множество объектов предметной области, содержание – множество свойств, которыми обладают все эти объекты. Для выявления понятий составляется формальный контекст для соответствующего

фрагмента предметной области, который представляется как бинарная таблица <объект, свойство>. Множество получаемых формальных понятий упорядочено отношением частичного порядка и образует полную решетку концептов.

Формальный аппарат теории основан на рассмотрении множеств объектов O и признаков P, на которых определено отношение $R \subseteq O \times P$, такое, что oRp, где $o \in O$, $p \in P$, тогда и только тогда, когда p есть признак объекта o.

Тройка $K =< O, P, R >$ называется формальным контекстом. Формальный контекст представим в виде бинарной матрицы, строки которой помечены именами объектов, а столбцы – значениями признаков.

Определение формального понятия осуществляется на основе соответствия Галуа: для произвольных подмножеств $\mathrm{Ext}\,C \subseteq O$ и $\mathrm{Int}\,C \subseteq P$, таких что

$$\begin{cases} \mathrm{Int}\,C = \{p \in P \mid oRp \text{ для всех } o \in \mathrm{Ext}\,C\}; \\ \mathrm{Ext}\,C = \{o \in O \mid oRp \text{ для всех } p \in \mathrm{Int}\,C\}, \end{cases}$$

упорядоченная пара $C = (\mathrm{Ext}\,C, \mathrm{Int}\,C)$ объявляется формальным понятием C контекста K, а $\mathrm{Ext}\,C$ и $\mathrm{Int}\,C$ рассматриваются соответственно как экстенсионал (объем) и интенсионал (содержание) этого понятия. Каждый объект $o \in \mathrm{Ext}\,C$ обладает всеми признаками из подмножества $\mathrm{Int}\,C$. Таким образом, формальное понятие – это множество объектов из данной предметной области, каждый из которых обладает всеми признаками из некоторого подмножества свойств, присущих этим объектам.

Показывается, что множество всех понятий формального контекста K образует решетку $L(K)$:

– множество понятий является частично упорядоченным, т.е. для любых формальных понятий $C_1 =< \mathrm{Ext}\,C_1, \mathrm{Int}\,C_1 >$ и $C_2 =< \mathrm{Ext}\,C_2, \mathrm{Int}\,C_2 >$, $C_1 \geq C_2$, если $\mathrm{Ext}\,C_1 \supseteq \mathrm{Ext}\,C_2$ и $\mathrm{Int}\,C_1 \subseteq \mathrm{Int}\,C_2$;

– для любых двух понятий C_1 и C_2 определены операции $C_1 \wedge C_2 = C_\wedge$ и $C_1 \vee C_2 = C_\vee$, где

$$C_\wedge =< \mathrm{Ext}\,C_\wedge, \mathrm{Int}\,C_\wedge >,\ \mathrm{Ext}\,C_\wedge = \mathrm{Ext}\,C_1 \cap \mathrm{Ext}\,C_2,\ \mathrm{Int}\,C_\wedge = \mathrm{Int}\,C_1 \cup \mathrm{Int}\,C_2;$$

$$C_\vee =< \mathrm{Ext}\,C_\vee, \mathrm{Int}\,C_\vee >,\ \mathrm{Ext}\,C_\vee = \mathrm{Ext}\,C_1 \cup \mathrm{Ext}\,C_2,\ \mathrm{Int}\,C_\vee = \mathrm{Int}\,C_1 \cap \mathrm{Int}\,C_2.$$

Большинство работ по формальному анализу понятий ограничиваются адаптацией результатов математической теории для построения таксономической модели предметной области, в меньшей степени исследуются вопросы объективного формирования самих контекстов и совсем не рассматривается проблема описания структурных отношений между объектами.

По этой причине С.В. Смирнов [210] предпринял попытку вычисления (конструирования) онтологии предметной области. На основе анализа экспериментальных данных создается объектно-признаковая модель $< O, P, R >$, которая является контекстом K в терминологии формального анализа понятий, и, по утверждению автора, потенциально выражает:

– классификацию, воплощенную в определении состава множества объектов O;

– агрегирование как приписывание объекту некоторого множества признаков и задаваемое отношением R;

– ассоциацию, задаваемую семантической нагрузкой подмножества признаков P, которое составлено валентностями, определяемыми как способность объектов вступать в различные взаимосвязи.

В свою очередь, обобщение выявляется средствами формального анализа понятий и выражается отношением частичного порядка, заданного на решетке $L(K)$: понятие C_1 является обобщением понятия C_2, если $C_1 \geq C_2$.

Несмотря на присутствие в формальном анализе понятий некоторых проявлений четырех основных видов абстракции, развитыми можно назвать только средства для выражения обобщения, а средства для выражения ассоциативных связей между понятиями не задаются. Другая особенность теории – отсутствие возможности как априорной, так и апостериорной содержательной интерпретации выявленных понятий, что приводит к серьезным проблемам описания их семантики. И, наконец, усматривается практическая невозможность использования формального анализа понятий для решения прикладных задач большой размерности, что связано с необозримость таблицы, задающей формальный контекст.

Заметим, что в понятийной структуре имеется возможность явного задания всех четырех форм абстрагирования понятий, в то время как при формальном анализе используется решетка, явно выражающая только обобщение.

2.6.6. Концептуальные графы

Одним из наиболее известных средств языкового представления знания является формализм концептуальных графов, предложенный Дж. Сова [343]. Целью разработки этого формализма, наследующего идеи Ч. Пирса о реляционных графах [188], послужило стремление построить механизм, объединяющий мощность выразительной силы естественного языка и возможности символической логики. В цели входило также стремление обеспечить возможность для реализации зарекомендовавших себя формализмов, основанных на семантических сетях.

По своей идеологии формализм концептуальных графов разрабатывался как средство представления высказываний на естественном языке [344]. По этой причине в рамках данной теории обсуждаются такие вопросы, как представление синтаксических и семантических структур естественных языков, структуры дискурса, видовременных признаков, лексического выбора, темо-рематических отношений, контекста, метафоры и аналогий [248].

Концептуальный граф является двудольным графом, состоящим из вершин двух типов: понятий (конкретных и абстрактных) и концептуальных отношений, связывающих понятия и обозначаемых словами естественного языка.

Каждое понятие в графе представляет уникальный экземпляр конкретного типа и снабжается меткой типа, определяющей класс или тип экземпляра, представленного этим узлом. Вторая часть описателя понятия задает ссылку на объект, относящийся к данному типу. Ссылка задается либо индивидуально, либо для всех экземпляров в целом. Если для объекта ссылки используется маркер экземпляра (индексный референт #), понятие является конкретным, а если общий маркер (кореферентная связь *) – понятие считается родовым. Дополнительно в нотации имеется возможность представления лямбда выражений (оператор λ), расширяющих концептуальный граф на формулу лямбда-исчисления. Вводится также оператор ϕ, который отображает построенный граф на формулу исчисления предикатов.

В теории концептуальных графов используется понятие высказывания, или пропозиции, объектом ссылки которого является множество концептуальных графов. Пропозиционные понятия изображаются прямоугольником, содержащим другой концептуальный граф, и используются для представления знаний о высказываниях. Для этого прямоугольники пропозиционных понятий соединяются с узлами концептуальных отношений, помеченных словами для выражения модальности и доверия, например, «верится», «возможно», «вероятно», «обязательно» и т.д. Такое концептуальное отношение называется контекстом. Использование контекстов, задаваемых в виде графа, который содержит вложенный в него подграф, позволяет представлять вложенные утверждения и ограничивать область действия пропозиций.

Для описания концептуальных графов на языке предикатов формализм был дополнен унарной операцией отрицания, выражаемой в виде специального контекста, операндом которой является пропозиционное понятие. Явных средств для представления дизъюнкции в формализме нет – дизъюнктивная связь пропозиций выражается через конъюнкцию и отрицание на основе правил логики. Аналогично выражается квантор всеобщности, исходя из предположения, что родовые понятия связаны квантором существования.

Хотя концептуальные графы можно описать в синтаксисе исчисления предикатов, поддерживает ряд специальных механизмов формирования осмысленных (правдоподобных) высказываний путем создания новых графов на основе существующих, а именно: копированием, ограничением, объединением и упрощением. Эти механизмы позволяют генерировать новый граф путем либо специализации, либо обобщения существующего графа. Однако такие операции не являются правилами вывода и не гарантируют, что из истинных графов будут получены истинные графы.

Таким образом, каждый концептуальный граф представляет собой одно высказывание, а база знаний состоит из ряда таких графов. Для представления второго базового механизма структурирования знаний – обобщения понятий – в теории концептуальных графов используется иерархия типов, которая является решеткой, описывающую общий вид множественного наследования [345]. Однако иерархия типов задается отдельно от концептуальных графов, которые выражают ассоциативные связи между понятиями конкретного высказывания. Более того, эта иерархия представляется как некоторая статическая структура, в то время как ассоциативные связи задаются динамически для каждого концептуального графа.

Последнее приводит к проблемам, связанным с проверкой базы знаний на противоречивость, так как в рамках формализма могут существовать графы, выражающие взаимоисключающие представления о семантике одних и тех же понятий-ассоциаций, задаваемых концептуальными отношениями.

В итоге имеем, что понятийная структура, в отличие от концептуальных графов, позволяет на одной диаграмме выразить как обобщение, так и ассоциацию понятий. Более того, имеется возможность рассматривать ассоциации как отдельные понятия, на которых задавать другие понятия, в том числе и через их обобщение.

2.6.7. Модель «сущность-связь»

ER-модель (Entity Relationship) как концептуальная структура для инфологического проектирования баз данных предложена П. Ченом [240]. Ее развитие и совершенствование, связанное, в основном, с введением специализации, привело к появлению EER-модели (Extended Entity Relationship) [267], наиболее полное описание которой можно найти в коллективной работе [277].

Основными конструктивными элементами EER-моделей являются сущности, связи между ними и их свойства (простые и составные атрибуты). Сущность – любой различимый объект, причем отдельно выделяется тип сущности и экземпляр сущности. Тип сущности выражает набор однородных сущностей, выступающих как целое. Экземпляр сущности является конкретной сущностью. Атрибут – именованная характеристика сущности.

Имя атрибута уникально внутри типа, но может быть одинаковым для различных типов. Абсолютное различие между типами и атрибутами отсутствует. Атрибут является таковым только внутри типа. В другом контексте атрибут может выступать как отдельная сущность или самостоятельный тип. Ключ – минимальный набор атрибутов, по значениям которых можно однозначно найти требуемый экземпляр сущности внутри типа. Минимальность означает, что исключение из набора любого атрибута не позволяет идентифицировать сущность по оставшимся.

Связи рассматривается как соединение двух или более сущностей и относятся к одному из трех видов:

– агрегация – для выражения связей типа «часть» (part-of) и «свойство» (property-of);

– ассоциация (группировка) – для построения множеств объектов из существующих типов;

– обобщение-специализация – для представления подмножеств или отношений типа «есть экземпляр» (is-a).

Недостатком рассматриваемого формализма является семантическая атомарность ассоциативных связей, являющаяся следствием различимости двух и более одноименных ассоциаций, связывающих различные типы или сущности. Это приводит к ограничению выразительных качеств модели по причине невозможности использования ассоциаций как полноправных понятий (типов). Другой недостаток модели – невозможность выражения абстракции типизации, кроме как путем объединением однотипных сущностей в понятие-тип и задания ключа для идентификации сущностей внутри этого понятия.

Перечисленные недостатки отсутствуют в формализме понятийных структур. Более того, в EER-моделировании далее, чем ограниченная фиксация концептуальных отношений между понятиями, не пошли: описание семантики понятий переложено на стандартные универсальные языки программирования со всеми присищими им проблемами.

2.6.8. Категорный подход

Категорный подход к представлению знаний является естественным развитием реляционного подхода к представлению данных, дополненного идеями и методами абстрактных типов данных из области программировании.

В категорном подходе к представлению знаний предполагается, что знания прикладной области могут быть структурированы в виде системы понятий. С каждым понятием связывается имя понятия и его определение, а теория категорий предоставляет формальный аппарат для выражения семантики понятия.

Категорный подход основан на предположении, что определение всякого понятия может быть представлено в виде наборов имен областей данных, имен преобразований данных и отношений между данными. При этом не все элементы областей могут быть известны внутри данного понятия и могут уточняться в других понятиях.

Каждому представляемому понятию при категорном моделировании сопоставляется алгебраическая модель (категория), в котором потенциально предусматривается отражение полного знания о содержании понятия, а также строится конечная аппроксимация понятия (фрагмент категории), которая отражает уже имеющиеся знания о представляемом понятии. Категория, как идеальный математический объект, мыслится как отражающая полное знание о понятии и является пределом своих конечных аппроксимаций [18].

Средства категорного подхода к представлению знаний условно можно разделить на три части:

– средства построения алгебраической модели (категории);

– средства построения конечной аппроксимации (фрагмента категории);

– язык описания понятий и их взаимосвязи.

Все три составляющие сильно связаны между собой:

– категория и ее аппроксимация задаются языковыми средствами;

– корректность новых языковых выражений определяется по уже построенной аппроксимации.

В основе категорного подхода лежат категорные операции, которые предназначены для определения понятия и построения теоретико-множественных конструкций по описанию понятий, не все элементы которого известны. Представление сложных понятий основано на операции декартова произведения, выделения элементов в экстенсионалах понятий, не все элементы которых заданы. Для этого используются категорные конструкции топоса, в частности, булевы топосы, в которых реализуется классическая логика и которые могут быть конструктивно заданы.

Категорный подход основан на отображениях одних множеств (областей) в другие. Если $f_1 : T_1 \to T_2$ и $f_2 : T_2 \to T_3$ – отображения на множествах T_1, T_2 и T_3, заданные именами f_1 и f_2 такие, что , то строится имя $f_1 * f_2$ и отображение T_1 в T_3: $f_1 * f_2 : T_1 \to T_3$. Описанная операция называется композицией отображений. Композиция отображений ассоциативна, т.е. для любых отображений $f_1 : T_1 \to T_2$, $f_2 : T_2 \to T_3$ и $f_3 : T_3 \to T_4$ выполняется равенство $(f_1 * f_2) * f_3 = f_1 * (f_2 * f_3)$. Кроме того, для любого множества T вводится тождественное отображение $i(T) : T \to T$, композиция которого с любым отображением $f : T_1 \to T_2$ дает следующие равенства: $i(T_1) * f = f$, $f * i(T_2) = f$. Математический струк-

тура C, обладающая операцией типа композиции с описанными выше свойствами называется ***категорией***. В свою очередь под ***морфизмом*** понимается множество имен отображений, рассматриваемых с точностью до равенства.

При категорном подходе требуется существование двух специальных множеств. Вводится множество I – ***финальное***, которое обладает следующим свойством: для каждого множества T категории C существует единственный морфизм из T в I, который обозначается как $I(T): T \to I$. Другое специальное множество – ***инициальное***, обозначается $\varnothing$ и обладает следующим свойством: для любого множества категории C множество всех морфизмов $C(\varnothing, T)$ состоит из единственного морфизма $\varnothing(T): \varnothing \to T$.

Для представления понятий используются категории с операциями, которые позволяют по одним множествам и морфизмам строить другие множества и морфизмы. К таким операциям относятся операции:

– $\_(f)$ и $(f)\_$ выделения области определения T_1 и области значений отображения $f: T_1 \to T_2$;

– операции выделения инициального и финального множества $\varnothing(T)$ и $I(T)$;

– операция тождественного отображения $i(T): T \to T$;

– другие операции вида $(f_1, f_2) \to f_1 * f_2$ при $(f_1)\_ = \_(f_2)$, которые сопоставляют паре морфизмов (f_1, f_2) морфизм $f_1 * f_2$, получаемый при их композиции.

Заметим, что категорный подход аналогичен формальному анализу понятий, где также имеется минимальный ($\varnothing$) и максимальный (I) элементы в решетке обобщения. Однако для представления сложных понятий перечисленных выше операций оказывается недостаточно. В отличие от формального анализа понятий при категорном подходе водится операция декартова произведения, суммы и пересечения множеств, операция взятия фактор-множества.

Другое отличие категорного подхода от формального анализа понятий заключается в том, что множество не определяется своими элементами, поэтому перечисленные теоретико-множественные операции определяются специальным образом – на языке морфизмов [85]. Категория, на которой определены категорные операции, моделирующие полный набор теоретико-множественных операций, называется ***топосом***.

Например, декартово произведение множеств $T_1 \times T_2$ выражается как множество T с морфизмами (проекциями) $pr_1: T \to T_1$ и $pr_2: T \to T_2$, обладающие следующими свойствами: для любой пары морфизмов $f_1: T_0 \to T_1$ и $f_2: T_0 \to T_2$ существует морфизм $(f_1, f_2): T_0 \to T$, композиция которого с проекциями дают равенства $pr_1 * (f_1, f_2) = f_1$ и

$pr_2 * (f_1, f_2) = f_2$, и наоборот, любой морфизм вида $f : T_0 \to T$ представляется в виде $f = (pr_1 * f, pr_2 * f)$. Таким образом, при определении декартова произведения задаются операции проекции, произведение морфизмов, которые по произвольным множествам T_1, T_2 и морфизмам f_1, f_2 выражают множество T, равное $T_1 \times T_2$. В итоге, построение декартова произведения множеств путем перечисления упорядоченных пар элементов заменяется на описание свойств этого произведения через специальным образом введенные категориальные операции.

Таким образом, в категорном подходе понятия представляются категориями, а их отношения – морфизмами, причем вместо существования категории или морфизма с требуемыми свойствами полагается, что эта категория или морфизм строится соответствующей операций. Последнее позволяет задавать понятия из конечными аппроксимациями – формулами, выражающими построение категории через операции над другими категориями.

Заметим, что при описании понятийной структуры, как и в категорном подходе, не требуется перечисления элементов множеств, на которых эта понятийная структура определена. Вместо этого используются четыре операции (абстракции), которые позволяют описать схемы понятий-абстракций через схемы абстрагируемых понятий, а определение экстенсионалов осуществляется конечными конструктивными средствами, которые при категорном подходе соответствуют определению новых морфизмов через композицию имеющихся.

Однако в понятийной структуре, в отличие от категорного подхода, для выражения понятий требуется всего четыре простые и интуитивно понятные операции, в то время как при категорном подходе используется открытое множество операции, которые, к тому же, имеют непрозрачную семантику и трудны в использовании.

2.6.9. Понятийная структура

Как итог рассмотренного выше анализа известных формальный теорий понятий сформулируем отличия предлагаемого подхода, основанного на высокоуровневой спецификации предметной в виде ее понятийной структуры.

Понятийная структура строится на четырех множествах отображений, соответствующих четырем универсальным формам абстрагирования понятий, которые задаются на множестве априорно определяемых понятий. Благодаря этому принципиальными отличиями понятийной структуры от других известных формализмов является:

– отсутствие разделения терминов на понятия, связи, сущности и признаки, а использование одного общего термина – понятие, частными проявлениями которого являются сущности, признаки и связи;

– наличие средств для явного выражения типизации понятий, а не только при классификации сущностей для образовании соответствующих им понятий;

– возможность представления ассоциаций как самостоятельных понятий, что позволяет, например, выразить обобщение ассоциативных связей;

– выражение на одной диаграмме как абстракций обобщения, так и абстракций ассоциации;

– определение понятий, которые одновременно могут быть как обобщением, так и ассоциацией других понятий;

– семантическая прозрачность формализма, не требующего для своей интерпретации привлечения предметных знаний.

Таким образом, принципиальное отличие предлагаемого подхода заключается в возможности определения ассоциации между сущностями и понятиями предметной области в виде самостоятельных понятий, в то время как в других формализмах ассоциации понятиями не являются, выражаются специализированными связями между сущностями и определяются как атомарные единицы смысла, посредством которых производится описание предметной семантики.

2.7. Методика понятийного анализа

Необходимость анализа предметной области до начала ее формализованного описания осознана давно и используется при разработке масштабных программных проектов. В этом случае процесс разработки существенно отличается от написания программ для решения вычислительной задачи [147]. Например, главное отличие проектирования баз данных заключается в том, что осуществляется предварительная разработка концептуальной схемы, которая отражает взаимосвязи типизированных сущностей проблемной области и особенности организации представляющих их данных.

В отличие от известных подходов, основанных на построении объектной модели, ***понятийный анализ*** определим как методику построения и верификации понятийной структуры проблемной области.

2.7.1. Этапы понятийного анализа

Понятийный анализ выполняется с учетом некоторой активной проблематики и состоит из следующих этапов:

– разделение сущностей предметной области на сигнификативные и денотационные;

– означивание сигнификативных сущностей и выявление существенных признаков у денотационных сущностей;

– сопоставление сущностей и определение их общих и различающихся признаков;

– образование новых или определение уже существующих понятий на основе интеграции и дифференциации признаков;

– создание понятийной структуры предметной области путем описания отображений одних понятий на другие;

– уточнение способа абстрагирования понятий (обобщения или типизации, ассоциации или агрегации);

– вычисление схем понятий и задание ключей – для типизации, связей – для ассоциации;

– верификации понятийной структуры путем проверки ее на полноту и непротиворечивость.

Разделение сущностей предметной области на сигнификативные и денотационные представляет, в общем случае, нетривиальную задачу. Сложность данной задачи проявляется в неформальном характере правил, которые можно применять для этой цели:

– сущности, составляющие объемные понятия, относят к денотационным, а единичные – к сигнификативным;

– часто выражаемые сущности относят к денотационным, редко выражаемые – к сигнификативным;

– сущности, имеющие сложную семантику относят к денотационным, а простую – к сигнификативным.

– сущности, используемые в множествах контекстов относят к денотационным, а в одном или нескольких контекстах – к сигнификативным;

– сущности, являющиеся основными носителями смысла относят к дентационным, а вспомогательные – к сигнификативным.

Означивание сигнификативных сущностей осуществляется путем присвоения им индивидуальных имен, которые образуют базовый словарь проблемной области. ***Выявление*** существенных признаков у денотационных сущностей производится путем перечисления тех признаков, которые являются значимыми в рассматриваемой проблемной области. Заметим, что на этом этапе происходит и выявление самих признаков, или первич-

ных понятий проблемной области, которые рассматриваются как денотационные сущности, не имеющие признаков.

Следующим этапом понятийного анализа является ***сопоставление*** сущностей и определение их общих и различающихся признаков. Это позволяет выполнить первичную группировку сущностей на основе анализа тождества и различия множества признаков, на которых эти сущности определены.

Образование новых или определение уже существующих понятий осуществляется на основе выявления интеграции и дифференциации признаков у выявленных на предыдущем этапе сущностей. Очевидно, сравнению подвергаются только те сущности и их группы, которые видятся связанными по условию решаемой задачи.

Понятийная структура призвана в формализованном виде отразить результаты предыдущих этапов и выражает отображения одних понятий в другие. В случае необходимости производится ***уточнение*** способа образования понятий: дифференциальные понятия помечаются как понятия-обобщения или понятия-типы, а интегральные понятия – как понятия-ассоциации или понятия-агрегаты.

Вычисление схем понятий, входящих в понятийную структуру, осуществляется по рекуррентной процедуре, описанной выше, и используется для проверки ее полноты и непротиворечивости.

Методы ***верификации*** понятийной структуры непосредственно следуют из утверждений о свойствах понятийных структур и формальной теории понятий.

В конечном итоге понятийный анализ позволяет на основе фундаментальных абстракций понятий получить декомпозиционную схему проблемной области в виде семантически прозрачной формальной спецификации ее понятийной структуры. Используемый для этого формализм имеет простые методы верификации путем проверки понятийной структуры на полноту и непротиворечивость. Оценим, как соотносятся полученные результаты с распространенной в настоящее время методологией объектного анализа.

2.7.2. Объектный и понятийный анализ

Фундаментальными понятиями объектного анализа являются понятия класса и объекта. При этом под классом понимают некоторую абстракцию совокупности объектов, которые имеют общий набор свойств и обладают одинаковым поведением. Каждый объект в этом случае рассматривается как экземпляр соответствующего класса. Объекты, которые не имеют полностью одинаковых свойств или не обладают одинаковым поведением, по определению, не могут быть отнесены к одному классу.

Важной особенностью классов является возможность их организации в виде некоторой иерархической структуры, которая по внешнему виду напоминает схему классифи-

кации понятий. Однако иерархическая схема организации понятий не тождественна иерархии классов, поскольку взаимосвязи между классами могут иметь и другие качественные особенности. С другой стороны, иерархия понятий является более общей категорией по сравнению с иерархией уровней абстракции классов в объектном анализе [147].

Целью объектного анализа является разработка объектной модели проблемной области, содержащей описания объектов, а также различные зависимости между ними. Считается, что декомпозиция проблемы на объекты – творческий, плохо формализуемый процесс. Процесс построения объектной модели включает в себя следующие этапы:

- определение объектов и классов;
- подготовка словаря данных;
- определение зависимостей между объектами;
- определение атрибутов объектов и связей;
- организация и упрощение классов при использовании наследования;
- дальнейшее исследование и усовершенствование модели.

Объединение объектов в классы позволяет ввести абстракцию типизации и рассмотреть проблему в более общей постановке. Классам приписываются атрибуты. Атрибут – это значение простого типа данных, характеризующее объект в его классе. Текущие значения атрибутов характеризуют текущее состояние объекта. В объектном анализе используется понятие операции, которая определяется как функция, или преобразование, которую можно применять к объектам данного класса. Каждой операции соответствует метод – реализация этой операции для объектов данного класса. Таким образом, операция – это спецификация метода, а метод – это реализация операции.

Так как в модели используемая сложная семантическая разметка, то важным этапом объектного анализа является построение словаря данных, который содержит четкие и недвусмысленные определения объектов (классов), атрибутов, операций и ролей, задаваемые на естественном языке.

Между объектами устанавливаются связи, которые выражают отношения между классами указанных объектов. Связи, как и классы, могут иметь атрибуты. В объектном анализе различают шесть видов связей: зависимость, агрегация, ассоциация, композиция, генерализация и реализация.

Прежде всего из классов исключаются атрибуты, являющиеся явными ссылками на другие классы; такие атрибуты заменяются зависимостями. Зависимость задает отношение зависимости и характеризуется специальными атрибутами, называемыми ролью и квантификатором, где роль несет семантическую нагрузку, а квантификатор определяет кратность связи.

Ассоциация определяется как отношение взаимодействия и, как все оставшиеся связи, описывается двумя ролями и двумя квантификаторами. Агрегация выражает отношение «часть-целое», а композиция рассматривается как частный случай агрегации, при которой жизненный цикл частей и целого совпадает. Генерализация задает отношение «частное-общее», а реализаций – отношение выполнения соглашения.

В конечном итоге, объектная модель представляется в виде совокупности объектов, каждый из которых является экземпляром определенного класса, а классы образуют иерархию наследования, где под наследованием понимается абстракция конкретизации классов, при которой производный класс приобретает свойства и поведение базового класса. Иерархия наследования является основной формой представления объектной модели: остальные виды связей наносятся на уже полученную иерархию классов.

В основе проверки правильности объектной модели лежат внешние признаки, объединены в следующие группы:

- признаки пропущенного объекта (класса);
- признаки ненужного (лишнего) класса;
- признаки пропущенных зависимостей;
- признаки ненужных (лишних) зависимостей;
- признаки неправильного размещения зависимостей;
- признаки неправильного размещения атрибутов.

В табл. 2.2 показано сравнение терминов и процедур понятийного и объектного анализа. Из таблицы видно, что понятийный анализ по сравнению с объектным является некоторым обобщением последнего, так как базируется на более общих принципах и подходах. Заметим, что в отличие от объектного, результаты понятийного анализа могут быть легко формализованы и подвергнуты проверке на полноту и непротиворечивость.

Следующим шагом в реализации результатов понятийного анализа видится разработка контекстной технологии обработки данных, основанной на создании для каждого класса задач и в процессе их решения характерного только для этих задач проблемного языка, отражающего понятийную структуру проблемной области в его нетерминальные понятия, а сигнификативные сущности – в терминальные понятия языка.

Таблица 2.2. Сравнение понятийного и объектного анализа

Понятийный анализ	**Объектный анализ**
Сущность (единичное понятие):	***Объект:***
– обладает уникальностью; – различается признаками; – выражает смысл.	– обладает идентичностью; – имеет состояние; – проявляет поведение.
Признак (элементарное понятие):	***Свойство*** (атрибут):
отличает одну сущность от другой; имеет имя, домен и семантическую роль.	принимает различные значения и характеризует состояние объекта.
Понятие:	***Класс***:
множество сущностей, образованное на основе абстрагирования.	множество объектов, имеющих общую структуру и общее поведение.
Спецификация понятия:	***Спецификация класса***:
– имя (уникальность), схема (признаки); – интенсионал (содержание); – экстенсионал (объем).	– имя (идентичность); – свойства (состояние); – методы (поведение).
Взаимосвязь понятий:	***Взаимосвязь классов***:
– обобщение (есть некоторый); – агрегация (есть часть); – ассоциация (есть участник); – типизация (есть экземпляр).	– общее и частное (наследование); – целое и часть (агрегация); – зависимость (ассоциация).
Обобщение (расширенная типизация):	***Наследование***:
объединение сущностей дифференцируемых понятий.	наследуемый класс повторяет структуру и поведение базового класса.
Агрегация:	***Агрегация***:
соединение сущностей интегрируемых понятий.	отношения целого и части, приводящие к иерархии объектов.
Ассоциация (ограниченная агрегация):	***Ассоциация***:
связывание сущностей интегрируемых понятий.	зависимость классов, обеспечивающая переход между объектами этих классов.
Типизация:	***Виртуальные классы***:
идентификация сущностей дифференцируемых понятий.	объект базового класса замещается объектами различных классов.
Понятийная структура:	***Диаграммы***:
множество понятий с отображениями абстрагирования.	иерархия классов, диаграммы состояний и поведения объектов.

2.8. Заключительные замечания

В области создания и сопровождения современных информационных систем наблюдаются трудности, которые в значительной мере объясняются семантическим разрывом, возникающим между содержательными представлениями относительно предметной области и языковыми средствами, служащими для решения стоящих прикладных задач. Для преодоления этого вида семантического разрыва предлагается использование разработанной методологии понятийного анализа и соответствующей ей контекстной технологии обработки данных.

В предлагаемом подходе понятийный анализ используется для разделения понятий предметной области на денотационные и сигнификативные, построения понятийной структуры на денотационных понятиях, отражения сигнификативных понятий в терминальные знаки проблемного языка, а денотационных – в его нетерминальные знаки. В свою очередь, декомпозиционные схемы, полученные при понятийном анализе, отражаются в языковых конструкциях проблемного языка. Языковые конструкции задают формы выражения денотационных понятий в тексте и строятся как последовательность денотационных и сигнификативных понятий. Это позволяет в рамках заданной проблематики получить наиболее компактное описание предметной области, исходных данных и решения прикладной задачи по обработке этих данных.

Следует заметить, что получаемая при понятийном анализе модель предметной области обладает свойствами, позволяющими выразить следующие свойства прикладных областей знания [172]:

– недоопределенность – имеется возможность пополнения понятийной структуры новыми понятиями путем добавления понятий и определения способов их абстрагирования, детализации (доопределения) понятий путем добавления новых форм их выражения;

– неоднозначность – имеется возможность использования неоднозначных грамматических форм выражения понятий, что приводит к необходимости принимать решение о выборе той или иной формы на уже этапе решения прикладной задачи;

– некорректность – имеется возможность определить ту или иную семантически некоррекную форму выражения понятия, Однако при последующем анализе, или удалить эту форму, или ее переопределить в соответствие с открывшимися новыми обстоятельствами;

– неточность – имеется возможность введения и использования приближенных или неточных понятий, позволяющих описывать неполные или неточные величины-сущности, например, через такие количественные и качественные понятия как точность, точность точности, степень точности, класс точности, и др;

– нечеткость – имеется возможность использования для выражения понятий нечетких высказываний, например, «маленький», «средний», «большой», и т.п., с последующим описанием семантики таких форм, в том числе и с помощью средств нечеткой логики.

Более детально выразительные возможности разработанного типа модели предметной области рассмотрены в следующей главе.

Выводы к Главе 2

1. Показано, что понятийный анализ базируется на фундаментальных принципах, лежащих в основе познавательной деятельности человека, и может рассматриваться как предельно общий инструмент декомпозиции предметной области.

2. Выполнена формализация известных абстракций понятий: обобщения, типизации, агрегации и ассоциации, на основе чего определена понятийная структура и найдены методы ее верификации.

3. Разработано исчисление понятий, предназначенное для выражения семантически инвариантных свой предметных областей и доказана его полнота и непротиворечивость.

4. Для выражения специфических свойств предметных областей созданы общая (контекстная) и специальная (контекстно-свободная) теории понятий

5. Показано, что для синтаксической полноты формальной теории понятий необходимо бесконтекстное использование понятий предметной области в рамках формализма специальной теории; при контекстной интерпретации понятий непротиворечивым формализмом является общая теория, которая требует индивидуального доказательства синтаксической полноты каждой выводимой в ней формулы.

Глава 3.
Контекстная технология

Настоящая глава посвящена контекстной технологии обработки данных, которая появилась как объединение объектно-ориентированной и фортоподобной технологий [11, 266], осуществленное на основе понятийного анализа предметной области, контекстной интерпретации текстов программ [54] и определения семантики создаваемого в процессе решения задачи проблемного языка, предназначенного для описания этого решения [69].

Основной отличительной особенностью контекстной технологии является то, что для формализации знаний о некоторой предметной области, данные, выражающие эти знания, сопровождаются описанием их структуры и содержания, т.е. для представления предметных знаний используются данные, которые дополнены описанием их синтаксического (структурного) строения и правилами их семантической (смысловой) интерпретации.

Последнее, в частности, позволяет отказаться от использования универсальных или жестко заданных формальных систем для описания дискретной обработки данных, а использовать для каждой дискретной обработки свой, присущий только этой обработке, формализм.

3.1. Содержательная постановка задачи

Семантический разрыв между формальной системой и содержательными представлениями относительно предметной области будем рассматривать как следствие различия понятий, предоставляемых этим формализмом, и понятий, используемых при постановке и решении прикладных задач. Например, широко известно, что любой универсальный язык программирования навязывает разработчику информационной системы некоторую систему понятий, в то время как содержательные представления о предметной области с этими понятиями согласуются плохо или не согласуются вообще.

Однако если предоставить разработчику возможность выражать необходимые для него понятия и найденные им декомпозиционные схемы в виде специализированной формальной системы, которая предназначена для содержательного описания предметной области и заданного класса прикладных задач, а также описать семантику этой системы, то следует ожидать сокращения семантического разрыва между предметной областью и средствами формальной спецификации этой области.

Понятно, что основной семантический разрыв между содержательными представлениями и целевой вычислительной платформой, используемой для решения прикладных задач, устранен быть не может. Однако предоставление возможности разработчику на каждом уровне архитектурной иерархии спецификаций предметной области и решаемых в ней задач выражать требуемую ему систему понятий и найденные им декомпозиционные схемы в форме, близкой постановке и решению стоящих прикладных задач, позволит значительно облегчить преодоление основного семантического разрыва.

Для преодоления семантического разрыва между содержательными представлениями и языком моделирования (программирования) применим контекстную технологию. Суть подхода заключается в том, что для каждого класса задач (и для каждого уровня описания) и в процессе решения этих задач будем создавать свой, присущий только этим задачам (и уровню описания) проблемный язык.

Для описания семантики проблемного языка применим принципы семантического замыкания и семантической индукции, которые заключаются в использовании семантических категорий, которые определяются по мере необходимости, в процессе описания семантики определяемого языка и средствами самого языка. Для привязки самого нижнего уровня описания к целевой вычислительной платформе будем использовать базовые семантические категории, реализуемые командами и встроенными структурами данных этой платформы.

Таким образом, в результате приближения выразительных средств формальной системы к постановке и решению прикладных задач следует ожидать повышение качества и надежности информационных систем, созданных по контекстной технологии. Очевидно, предлагаемый подход основывается на допущении, что уже в процессе изучения предметной области и специфики прикладных задач, еще до начала формализации, создается система понятий и декомпозиционные схемы, наиболее приспособленные для постановки и решения этих задач.

3.2. Принципы контекстной обработки

Контекстная технология, реализуемая путем обработки специальным образом организованных данных, представляющих собой формальную спецификацию предметной области, включает следующие этапы:

– фиксация предметной области и некоторой активной проблематики в ней;

– понятийный анализ проблемной области и определение ее понятийной структуры;

– описание проблемного языка путем создания понятийной модели проблемной области;

– ситуационное описание проблемной области в виде совокупности имеющих место фактов (суждений);

– описание решения или свойств решения активной проблемы в форме допустимых для такого решения выражений (умозаключений).

Базовым принципом контекстной технологии является создание в процессе решения задачи проблемного языка. Выявленные в процессе понятийного анализа множество понятий и декомпозиционные схемы кладутся в основу создаваемого языка: понятия проблемной области становятся понятиями языка, а декомпозиционные схемы – его синтаксическими конструкциями.

Другим принципом, именем которого названа описываемая технология, является контекстная интерпретация текстов. Контекстная интерпретация как принцип заключается в определении семантического значения произвольного фрагмента текста в зависимости от окружающего его контекста. В отличие от генераторов компиляторов, у которых возможности задания контекстных условий достаточно бедны [232], в контекстной технологии такие условия не только естественным образом определяются, но и систематически используются.

И наконец, в контекстной технологии реализовано семантическое замыкание определяемого языка, заключающееся в описании семантики его конструкций не внешними, а внутренними средствами, т.е. необходимые семантические категории объявляются по мере необходимости, в процессе описания языка и средствами самого языка, а первичные семантические категории предоставляются целевой вычислительной платформой.

3.2.1. Семантическое замыкание

Из практических соображений метаязыковый путь описания семантики формальных языков видится неконструктивным, так как приводит к бесконечной рекурсии: для описания семантики метаязыка тербуется другой метаязык, который также нуждается в описании. Для решения возникающих проблем выполним замыканием метаязыка на его предметный язык, суть которого заключается в том, что, во-первых, множество первичных семантических категорий оставляется открытым и пополняется на этапе решения конкретной прикладной задачи, во-вторых, используется индуктивная грамматическая форма определения более сложного смысла через первичные семантические категории, а также через семантические категории, определенные ранее.

Для этого предметный язык будем рассматривать как тройку, состоящую из предметного синтаксиса, или правил построения высказываний; предметной онтологии, или

системы понятий языка; предметной семантики, или правил интерпретации высказываний (рис. 3.12).

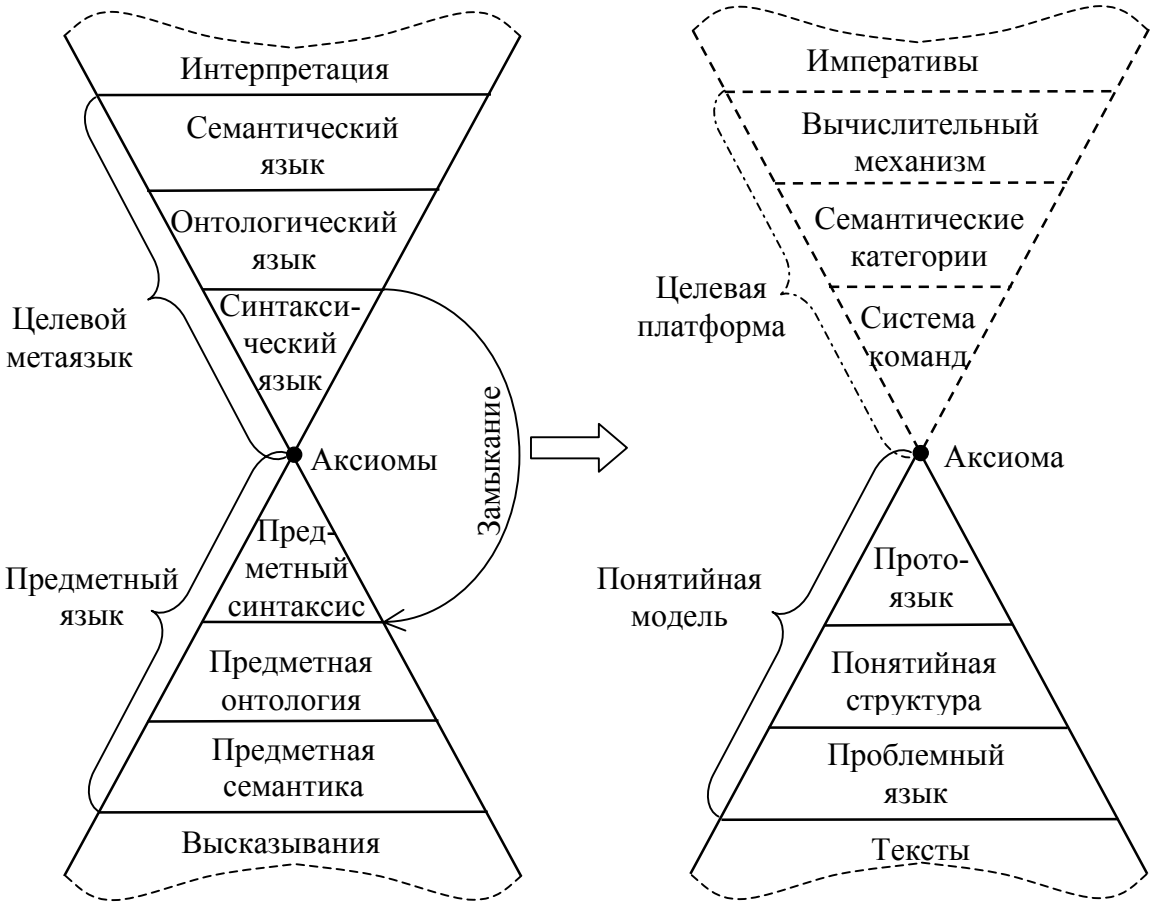

Рис. 3.12. Семантическое замыкание

Для описания предметного языка необходим метаязык. В соответствии с определенным выше делением предметного языка, в метаязыке выделим три вида выразительных средств: для описания предметного синтаксиса (синтаксический язык); для определения предметной онтологии (онтологический язык); для описания предметной семантики (семантический язык). Связь метаязыка и предметного языка осуществляется через некоторую систему аксиом, регламентирующих правила представления, структурирования и кодирования текстов.

Отождествим составные части метаязыка с соответствующими частями предметного языка, а семантику предметного языка будем определять на самом предметном языке, который, в этом случае, играет роль своего метаязыка, а то общее, что присутствует при определении произвольных предметных языков вынесем в ***протоязык***.[63] Так как внутри

[63] За внешним многообразием естественных языков, которых в настоящее время насчитывается около шести тысяч, проступают общие контуры коммуникативной системы человека. Так, во всех языках имеется деление звучащей речи на слоги – соответственно, противопоставлены гласные (слогообразующие) и согласные (неслогообразующие) звуки. Во всех языках представлено разли-

замкнутой семиотической системы семантику определить нельзя, воспользуемся внешним ***семантическим интерпретатором*** – целевой вычислительной платформой, состоящей, как и целевой метаязык, из следующих частей:

– алфавита метазнаков, задающих синтаксические средства метаязыкового уровня (система команд);

– семантических категорий, сопоставленных метазнакам (действия, реализуемые командами);

– механизма интерпретации метавысказываний (вычислительный механизм исполнения императивов).

В итоге получаем понятийную модель предметной области, которая состоит из аксиомы, протоязыка, понятийной структуры и некоторого специализированного предметного языка, или ***проблемного языка***.

Таким образом, для каждого класса задач и в процессе их решения будем создавать свой, присущий только этим задачам проблемный язык, отражающий понятийную структуру проблемной области. Выявленные в процессе анализа понятия включим в множество понятий создаваемого языка, а способы выражения понятий положим в основу его синтаксиса. Описание понятийной структуры и синтаксиса создаваемого проблемного языка выполним на протоязыке, который имеет фиксированный синтаксис и семантику, минимально достаточные для определения синтаксиса проблемного языка и пополнения его множества базовых семантических категорий.

Для описания семантики проблемного языка применим метод семантической индукции, заключающийся в использовании семантических категорий, которые определяются по мере необходимости, в процессе описания синтаксиса проблемного языка и средствами этого языка. Базу индукции, или первичные семантические категории, объявим с помощью аксиомы и реализуем средствами целевой вычислительной платформы.

чие между номинацией (называнием), предикацией (суждением) и дискурсом (умозаключением). Все языки имеют средства для выражения прагматических компонентов высказывания. Во всех языках есть слова, синтаксис, и между отдельными составляющими синтаксической структуры могут быть усмотрены закономерные корреляции. Все это дает возможность говорить о наличии некоторой универсальной грамматики – общей врожденной подосновы всех реально засвидетельствованных и потенциально возможных языков, или протоязыка [15]. Лингвисты, которые признают существование этой универсальной грамматики, в первую очередь задаются вопросом происхождения универсалий – как появились те свойства, которые являются общими для всех языков, и почему они оказались именно такими. Однако нами ставится задача не анализа исторического пути развития и становления языка, а задача поиска и выражения наиболее общих и значимых свойств современных языков, преимущественно искусственных, формализация которых позволит в наиболее простой и естественной форме определять искомые проблемные языки.

На содержательном уровне семантическое замыкание заключается в использовании создаваемого проблемного языка для описания его семантики. Последнее возможно ввиду отрытого множества базовых семантических категорий и наличия механизма его пополнения посредством использования единственно необходимой для этого аксиомы.

3.2.2. Понятийная модель

Под ***моделью*** проблемной области будем понимать данные, отражающие накопленные знания об этой проблемной области и обладающие синтаксической и семантической полнотой.

Для формализации знаний о проблемной области будем строить ее понятийную модель. Построение понятийной модели осуществим путем выявления и выражения понятийной структуры в объеме, достаточном для решения некоторого класса задач. В отличие от других технологий, например, структурной и объектно-ориентированной, которые основаны соответственно на структурном и объектном анализе, контекстную технологию определим на основе понятийного анализа проблемной области.

В качестве программы будем использовать понятийную модель, дополненную описанием решения одной или нескольких прикладных задач:

Программа = Понятийная модель + Решение задачи.

В отличие от технологии структурного программирования, в которой программа представляется совокупностью алгоритмов и структур данных [44], и объектно-ориентированной технологии, где программа – это декларация классов и алгоритмов создания и функционирования объекта класса, который во время своего существования осуществляет решение некоторой задачи [32], в контекстной технологии программа представляет собой определение некоторого объема знаний – в виде понятийной модели проблемной области, и описания искомого решения – в рамках определенных знаний.

Контекстная технология близка технологии создания экспертных систем [96], где экспертная система представляется как база знаний, содержащая структуру и факты о проблемной области, и механизм вывода, реализуемый машиной логического вывода, осуществляющей поиск решения, специфицированного некоторым запросом. Машина логического вывода рассматривается как программа, моделирующая механизм рассуждений и оперирующая данными с целью получения новых данных, выражающих результат рассуждений. Обычно машина логического вывода использует программно реализованный механизм дедуктивного логического вывода или механизм поиска решения в сети фреймов или семантической сети.

В качестве базы знаний в контекстной технологии выступает понятийная модель, а описание решения задачи задает вывод в этой базе знаний, приводящий к решению требуемой задачи.

В понятийной модели условно выделим четыре части:

– описание понятийной структуры;

– описание синтаксиса понятий;

– описание семантики понятий;

– описание процесса компиляции.

Для обеспечения синтаксической и семантической полноты модели понятийная структура и синтаксис понятий описываются на декларируемом в контекстной технологии протоязыке, а описание семантики понятий и решаемых задач выполняется на определяемом в процессе описания проблемном языке, задаваемом понятийной структурой и описанием синтаксиса понятий. В итоге имеем:

Понятийная модель = Структура + Синтаксис + Семантика.

Для ***привязки*** понятийной модели к целевой вычислительной платформе используется описание процесса компиляции, которое осуществляется как на определяемом языке, так и с использованием некоторого множества базовых примитивов. По своей сути описание процесса компиляции является определением специализированного компилятора создаваемого проблемного языка.

Базовые примитивы являются сущностями, реализующими элементарные семантические единицы модели, которые необходимы для определения ее семантики. Базовые примитивы имеют аппаратно-зависимую реализацию для каждой целевой вычислительной платформы.

В качестве ***целевых платформ*** могут использоваться такие аппаратурные и аппаратно-программные платформы как:

– микроконтроллеры, не имеющие операционных систем,

– вычислительные системы с развитой операционной средой,

– кросс-платформенные виртуальные машины,

– другие системы программирования.

В случае аппаратной платформы привязка понятийной модели осуществляется низкоуровневыми средствами, например, путем прямой генерации исполняемого кода. Другим крайним случаем является привязка модели с помощью другой системы программирования. В этом случае описание семантики проблемного языка осуществляется на языке, поддерживаемом целевой системой программирования.

3.2.3. Протоязык

Ограничим роль ***протозыка*** в контекстной технологии описанием структурных свойств проблемного языка. В протоязык включим средства для выражения наиболее устойчивых концепций, лежащих в основе как понятийного анализа, так и контекстной технологии. Протоязык будем использовать для описания понятийной структуры проблемной области и синтаксиса правил выражения понятий в тексте. Семантическую роль протоязыка ограничим реализацией аксиомы, позволяющей ввести в использование первичные семантические категории.

Лексический, синтаксический и семантический анализ, а также компиляцию части модели, написанной на протоязыке, реализуем известными методами. Однако для части модели, предназначенной для описания семантики на определяемом проблемном языке и по мере его определения, разработаем специальные методы грамматического разбора и компиляции. Для привязки определяемого языка к целевой вычислительной платформе будем использовать некоторое множество первичных семантических категорий (базовых примитивов), имеющих непосредственную аппаратную или программную реализацию средствами целевой платформы.

Таким образом, протоязык является подразумеваемой частью любой понятийной модели. Однако в отличие от базовых примитивов, которые декларируются перед использованием с помощью аксиомы, протоязык по своей природе является предопределенной частью любой модели.

Подробное описание принципов построения и одной из возможных реализаций протоязыка для контекстной технологии обработки данных дано в 3.3.

3.2.4. Семантический язык

Под ***семантикой*** понимается смысловая или содержательная интерпретация текста, в то время как форма представления или структура текста задаются его ***синтаксисом*** [126]. Обычно семантика языков определяется совокупностью неформальных правил и соглашений, устанавливаемых описанием языка и предназначенных для выявления смысла текстов на этом языке с целью их интерпретации человеком или автоматом [184].

Семантику понятийной модели будем задавать в виде текста, который построен по правилам определяемого в модели проблемного языка. Для этого каждое синтаксическое правило определяемого языка (предложение), служащее для выражения некоторого понятия, дополняется описанием его семантики. Семантика выражается совокупностью прагматик, задающих семантическую интерпретацию предложения в одном или нескольких возможных для него аспектах. После компиляции прагматики в последовательность команд целевой платформы имеем последовательность действий (императив), которую сис-

тема программирования выполняет всякий раз, когда понятие выражается этим предложением. По своей сути императивы являются единицами вызова целевой платформы, в то время как предложение модели – описанием синтаксиса таких вызовов.

Методы и средства контекстной технологии, предназначенные для определения семантики синтаксических правил проблемного языка, подробно описаны в 3.4.

3.2.5. Грамматический разбор

В контекстной технологии лексический, синтаксический и семантический анализ, а также компиляция части модели, содержащей описание понятийной структуры и синтаксиса предложений, реализуются традиционными средствами. Однако имеется часть понятийной модели, которая выражается на определяемом проблемном языке. Сюда относится описание семантики предложений и процесса компиляции.

Суть проблемы, возникающей при описываемом подходе, заключается в том, что необходимо реализовать универсальные средства для определения семантики проблемного языка.

Известные методы описания семантики, например, используемые при автоматическом конструировании компиляторов [232], порождают серьезные трудности при задании и проверке контекстных условий. Эти трудности, в основном, связаны с неразвитостью средств контекстной интерпретации фрагментов текста и жестко заданным набором первичных семантических категорий, которые могут использоваться для формального описания семантики [352].

В контекстной технологии пополнение множества первичных семантических категорий обеспечивается аксиомой. Однако формализм контекстно-свободных грамматик, используемый для задания синтаксиса правил выражения понятий, не позволяет осуществить контекстную интерпретацию текста. Но проблемный язык по своей выразительной мощности должен превосходит контекстно-свободные языки.

Под ***контекстной интерпретацией*** будем понимать определение семантического значения фрагмента текста по его месту в тексте программы. В общем случае одна и та же последовательность знаков, может быть разбита на смысловые части различным образом. Более того, одна и та же последовательность знаков может служить носителем различных смыслов.

При контекстной интерпретации традиционное разделение анализа текста на фазы лексического и синтаксического анализа не представляется возможным. Последнее связано, во-первых, с тем, что решение о лексическом делении текста, написанного на проблемном языке, не может быть предпринято до его определения, а это определение задается как раз анализируемым текстом. Во-вторых, в процессе грамматического разбора

текста, являющегося описанием проблемного языка, необходимо использовать ранее определенные конструкции этого языка, а для этого такие конструкции должны уже быть распознаны, интерпретированы и откомпилированы.

Для решения описанных выше трудностей в контекстной технологии необходимо использовать грамматический разбор языков, порожденных контекстными (контекстно-зависимыми) грамматиками. Для этого применен метод разнесенного грамматического разбора, заключающийся в объединении традиционных фаз лексического и синтаксического анализа в одну – фазу грамматического разбора, основанную на последовательной компиляции предложений модели и разделении определения применимости откомпилированных ранее предложений на две части: контекстное сопоставление, осуществляемое просмотром текста назад, и структурное распознавание, выполняемое при просмотре вперед.

Подробное изложение разнесенного грамматического разбора дано в Главе 4.

3.2.6. Контекстная обработка данных

Система контекстной обработки данных предназначена для разбора и интерпретации понятийной модели проблемной области, дополненной описанием решения некоторой прикладной задачи, и решения этой задачи путем интерпретации ее описания в соответствии с обработанной ранее понятийной моделью.

Для этого разбор понятийной структуры и описания синтаксиса понятий осуществляется известными методами лексического, синтаксического и семантического анализа, а компиляция описания семантики и текста решения прикладной задачи в исполняемый код осуществляется создаваемым в процессе программирования специализированным компилятором, который определяется в процесе описания понятийной модели совместно с проблемным языком. Последнее обеспечивается тем, что любое синтаксическое правило может дополняться описанием процесса его компиляции, которое также выражается на определяемом языке.

В итоге, в процессе своего функционирования система контекстной обработки находится в одной из трех фаз: в фазе разбора, в фазе компиляции и в фазе выполнения. В ***фазе разбора*** выполняется разбор текста, заданного на протоязыке, а в ***фазе компиляции*** – на определенном языке. В ***фазе выполнения*** исполняется код, порожденный при компиляции текста, выраженного на проблемном языке.

Более подробное описание одной из реализаций системы контекстной обработки данных приведено в Главе 4.

3.3. Протоязык понятийных моделей

Рассмотрим ***протоязык*** контекстной технологии, предназначенный для синтаксического и семантического определения проблемных языков, являющихся формализованным представлением результатов понятийного анализа проблемной области и служащих для описания решения определенного класса прикладных задач.[64]

3.3.1. Контекстная интерпретация

Терм определим как элементарную синтаксическую единицу, состоящую из последовательности знаков некоторого фиксированного алфавита. Сам ***алфавит*** и входящие в него знаки назовем терминальными. ***Лексему*** определим как элементарную семантическую единицу, представленную одним или несколькими термами. Иными словами, лексема – это множество термов вместе с приписанным (сопоставленным) им смыслом. ***Текстом*** будем называть последовательность лексем, предназначенную для выражения некоторого сложного смысла. Текст, в отличие от лексем, предполагает свое деление на смысловые части, в то время как лексема такого деления не допускает.

Для задания синтаксиса воспользуемся известным формализмом контекстно-свободных грамматик [205, 83]. Для обозначения термов в предложениях грамматики будем использовать одинарные кавычки. Например, 'x' обозначает терм, состоящий из одного знака терминального алфавита. В свою очередь, последовательность терминальных знаков, не заключенную в кавычки, будем использовать для обозначения нетерминальных знаков грамматики. В этом случае x может обозначать какой-то нетерминальный знак. Для разделения нетерминальных знаков грамматики будем использовать пробелы, для чего запретим появление пробелов в их обозначениях.

Как обычно, каждому нетерминальному знаку грамматики сопоставим некоторое понятие, называемое ***нетерминальным понятием*** языка, которое для краткости будем называть просто понятием. Тогда правила вывода грамматики могут быть интерпретированы как синтаксические правила выражения понятий языка в тексте.

---

[64] Протоязык, или язык-основа, – термин, обозначающий в лингвистике гипотетическое состояние группы или семьи родственных языков, реконструируемое на основе системы соответствий, которые устанавливаются между языками в области фонетики, грамматики и семантики [22, ст. Праязык]. Один из наиболее цитируемых авторов, исследовавший проблему протоязыка в семиотическом смысле, Д. Бикертон сформулировал различие естественного языка и протоязыка так: 1) в протоязыке допустима произвольная вариативность способов выражения языковых функций, 2) в протоязыке нет ни одного зарезервированного знака (слова), 3) глагол в протоязыке не может быть поливалентным (многопараметрическим), 4) в протоязыке не существует правил «грамматического развертывания», т.е. протоязык не имеет флективности (грамматической неоднозначности и вариативности языковых конструкций) [253, ст. Протоязык].

Под ***контекстной интерпретацией*** будем понимать определение семантического значения терма по его месту в тексте. В общем случае один и тот же терм может быть сопоставлен различным лексемам, т.е. одна и та же последовательность терминальных знаков может служить носителем различных смыслов. Следовательно, при контекстной интерпретации семантическое значение терма в общем случае определяется окружающим его контекстом.

Пример 3.28. Пусть в грамматике определяемого языка имеются два предложения с термом 'x':

C → A 'x' B
F → 'x' E

где A, B, C, E, F – некоторые понятия, а знак → разделяет определяемое понятие (слева) от понятий и термов (справа). Таким образом, понятие C определяется первым предложением, а понятие F – вторым. В свою очередь, где-то имеются предложения, определяющие понятия A, B и E.

Анализ приведенных предложений показывает, что если терм 'x' встретится в окружении последовательности терминальных знаков, выражающих понятия A и B, то он интерпретируется как лексема x первого предложения. В свою очередь, последовательность лексем A 'x' B (как и составляющая ее последовательность термов) выражает понятие C. Если терм 'x' встретится в контексте 'x' E, то он сопоставляется с лексемой x второго предложения, а рассматриваемая часть текста выражает понятие F. Очевидно, если ни один из разрешенных контекстов терма 'x' не распознан, то это говорит об ошибке в тексте. ♦

Формальным языком будем называть множество текстов, построенных по правилам, задаваемым формальной грамматикой. Следует различать язык как множество всевозможных текстов (язык-текст) и язык как множество правил (язык-правило). Язык-правило будем задавать в виде формальной грамматики.

Пример 3.29. В демонстрационных целях выберем в качестве определяемого язык булевых выражений, описываемый обозримой грамматикой. Рассмотрим тексты '(not x or y) and z', 'not and x' и грамматику, заданную предложениями на рис. 3.13, где терминальный алфавит содержит некоторое достаточно полное множество знаков, а нетерминальный алфавит состоит из одного понятия Boolean, являющегося аксиомой.

Предложения грамматики перечислены в порядке убывания их приоритета, а знак | используется для разделения правых частей предложений с одинаковым приоритетом, имеющих одну и ту же левую часть. Второе предложение определяет синтаксис переменных, заданный на языке регулярных выражений [369], где регулярное выражение пред-

ставлено в виде терминального шаблона и, для отличия от терма, заключено в двойные кавычки.

```
Boolean → 'false' | 'true'
Boolean → ″[A-Za-z][A-Za-z0-9]*″
Boolean → '(' Boolean ')'
Boolean → 'not' Boolean
Boolean → Boolean 'and' Boolean
Boolean → Boolean 'or' Boolean
Boolean → Boolean 'imp' Boolean
```

Рис. 3.13. Грамматика языка булевых выражений

Заметим, что текст '(not x or y) and z' может быть порожден этой грамматикой, следовательно, он принадлежит языку булевых выражений. Однако текст 'not and x' языку не принадлежит, так как построен не по правилам заданной грамматики. ♦

3.3.2. Грамматика протоязыка

Протоязык контекстной технологии определим как язык для выражения суждений о синтаксисе проблемного языка, создаваемого на основе понятийного анализа проблемной области и служащего для решения одной или нескольких прикладных задач. Протоаязык зададим грамматикой на рис. 3.14, представленной металингвистическими формулами Бэкуса-Наура.

Представленная грамматика задана с точностью до обозначенных курсивом свободных нетерминальных знаков ***notion***, ***aspect***, ***terms***, служащих для выражения имен определяемых понятий, аспектов и лексем (алиасов), а также с точностью до пробельных знаков, которые могут появляться между нетерминальными понятиями грамматики. В квадратных скобках обозначены части правил, которые могут быть опущены.

Текст программы program состоит из понятийной модели model и ситуаций situation (правило 1). В понятийной модели определяется язык, на котором в ситуационной части описывается решение некоторой прикладной задачи.

Понятийная модель является законченной единицей определения языка, а ситуация описывает решение прикладной задачи полностью, если в предшествующих описаниях определяемый язык задан полностью, или частично, когда язык определен в объеме некоторого подъязыка, достаточного для описания части решения.

1	program	→	model [situation]
			model situation program
2	model	→	essences
			essences model
3	essences	→	differenciation ***notion*** [integration] [intension]
4	differenciation	→	'(' [notions] ')'
5	integration	→	'(' [notions] ')'
6	notions	→	notion
			notion notions
7	intension	→	sentence
			sentence intension
8	sentence	→	[***aspect***] syntax semantic
9	syntax	→	item [parse] [compile]
			item [parse] [compile] syntax
10	item	→	notion [alias]
			lexeme [alias]
11	alias	→	'`' ***terms*** '`'
12	lexeme	→	term
			pattern
13	term	→	'/' [***terms***] '/'
14	pattern	→	'//' [***terms***] '//'
15	semantic	→	pragmatic
			pragmatic semantic
16	parse	→	'<' [text] '>'
17	compile	→	[aspect] '[' [text] ']'
18	pragmatic	→	[***aspect***] '{' [text] '}'
19	situation	→	[aspect] '<' [text] '>'
20	text	→	phrase
			phrase text
21	phrase	→	terms
			[aspect] '{' text '}'[65]

Рис. 3.14. Грамматика протоязыка

3.3.3. Понятийная структура

Понятийная модель состоит из описаний сущностей проблемной области essences (правило 2). Каждой сущности присваивается имя notion нетерминального понятия определяемого языка, а само понятие задается как находящееся в отношениях дифференциа-

[65] Правило phrase → [aspect] '{' text '}' предназначено для изменения аспекта произвольной фразы в текстах parse, compile, pragmatic и situation. Это правило является необязательным и включено в состав рассматриваемой версии грамматики протоязыка в демонстрационных целях. Определение синтаксиса указанного правила может быть выражено на проблемном языке, возможно даже другим способом, а его семантика задана путем вызова соотвествующего сервиса системы программирования, ответственного за смену аспекта интерпретации (см. 4.6.1 на с. 223).

ции differenciation и интеграции integration с системой других, ранее определенных понятий notions (правила 3-6).

Описание сущностей essences содержит указание как на обобщение или типизацию, так и на агрегацию или ассоциацию определяемого понятия ***notion*** (правило 3), что служит для выражения понятийной структуры проблемной области. Для указания на отсутствие у понятия абстракций дифференциации используются пустые круглые скобки (правило 4). В этом случае понятие считается обобщенным (типизированным) от пустого понятия empty. Аналогично, при отсутствии понятий в списке интеграции, понятие считается агрегирующим (ассоциирующим) пустое понятие empty.

Для выражения перекрестных и рекурсивных связей между понятиями используется их предварительное объявление, которое не включает конструкцию intension (правило 3).

3.3.4. Интенсионалы понятий

Каждое описание intension состоит из предложений sentence (правило 7). Предложения служат для задания интенсионала понятий путем определения синтаксиса выражений понятий в тексте программы и их семантики (правило 8). С каждым предложением связывается определяемое понятие-результат, именем ***notion*** которого названа описываемые сущности (правило 3), определяемые предложением полностью, когда больше нет предложений с тем же понятием-результатом, или частично, если такие предложения имеются (правило 7).

Синтаксис предложения syntax выражается последовательностью элементов item: имен понятий notion и лексем lexeme (правило 9). Лексема является терминальным понятием определяемого языка. Для выражения лексем могут использоваться как термы term, так и множества термов, задаваемые на языке регулярных выражений [369] в виде терминальных шаблонов pattern (правила 12-14). Терминальные знаки ***terms*** вводят в конструкцию предложения терм и заключаются в одинарные кавычки, а терминальные знаки, задающие терминальный шаблон, – в двойные. Для ссылок на отдельные элементы описания синтаксиса предложений используются алиасы (идентификаторы), задаваемые в виде терминальных строк в обратных одиночных кавычках (правила 6, 11).

По своей сути предложение sentence является представлением правил грамматики определяемого языка на протоязыке с тем отличием, что для задания особенностей обработки и компиляции предложения используются конструкции parse и compile (правило 8). Их роль – указание грамматическому анализатору (parse) и компилятору (compile) на действия, которые необходимо выполнить во время грамматического разбора предложения и во время его компиляции (после распознавания в тексте) соответственно. Например, ука-

зание на проверку более сложных контекстных условий, чем это задается средствами протоязыка. Более подробно система контекстной обработки данных, а также конструкции parse и compile, описаны в Главе 4.

3.3.5. Выражение абстракций

Дифференциация понятий differentiation служит для выражения абстракций обобщения и типизации (правило 4). Если схемы обобщаемых понятий (их списки дифференциации) являются одинаковыми или совместимыми,[66] то имеем случай типизации. Если схемы обобщаемых понятий различаются, то конструкция differentiation выражает обобщение.

Дифференциация понятий несколько похожа на наследование, используемое в объектно-ориентированных языках, но к последнему не сводится. При выявлении родового понятия обобщение указывает на его видовые составляющие, что позволяет при задании родовых понятий не повторять описания его видовых реализаций. В объектно-ориентированных языках базовый (родовой) класс определяется ранее и реализует общие свойства и методы, которые распространяются на все наследуемые (видовые) классы, определяемые позже. В итоге базовый класс «не знает» своих видовых реализаций. Последнее приводит к трудностям при типизации. Для решения этой проблемы в объектно-ориентированных языках был специально введен механизм виртуальных методов и виртуальных классов.

Интеграция понятий integration позволяет выразить абстракции агрегации и ассоциации (правило 5). Если интенсионал понятия, задаваемый описанием intension, ограничивает экстенсионал определяемого понятия, то реализуется абстракция ассоциации. В противном случае интеграция понятий выражает их агрегацию.

В итоге каждое предложение sentence выделяет некоторое множество сущностей экстенсионала понятия-агрегата и тем самым служит для задания связи между агрегируемыми понятиями. Последнее отличает интеграцию в контекстной технологии от агрегации в объектно-ориентированных языках, где отсутствуют явные средства для задания ассоциативных связей между объектами различных классов. Покажем на примерах выражение различных абстракций.

Пример 3.30. Для типизации понятий воспользуемся шаблоном, приведенным на рис. 3.15, где notion – понятие-тип; notion1, …, notionN – типизируемые понятия, key – агрегированное понятие-ключ.

---

[66] Здесь под совместимостью понятий понимается возможность представления одного понятия другими в рамках денотационного описания понятий в форме конверторов и с учетом абстракций обобщения (см. 4.2.6 на с. 192).

(notion1 … notionN) notion (key …) notion1 {…} … notionN {…} …	(…) notion1 (key …) notion {…} … (…) notionN (key …) notion {…}

Рис. 3.15. Выражение типизации понятий

Предложения вида notion1→notion используются для преобразования типизированного понятия в понятие-тип, а вида notion→notion1 – для преобразования понятия-типа в типизируемые понятия. В последнем случае предложение, которое используется для преобразования, задается текущей сущностью (значением) агрегированного понятия key и автоматически применяется системой контекстного программирования. ♦

Заметим, что в контекстной технологии встроена интенсиональная реализация абстракции типизации, заключающаяся в том, что понятие рассматривается как совокупность сущностей, имеющих одинаковую схему. Это позволяет путем добавления предложений, выражающих то или иное подмножество сущностей из экстенсионала понятия, разделить этот экстенсионал на подмножества-типы, имеющие различное выражение в тексте. В этом случае ключ не задается явно, а подразумевается присутствующим в синтаксисе соответствующих предложений.

Пример 3.31. Агрегация или ассоциация может быть выражена фрагментом, приведенным на рис. 3.16, где notion – понятие-агрегация или понятие-ассоциация; notion1, …, notionM – агрегируемые или ассоциируемые понятия, link – понятие-связь (используемая при ассоциации).

Если предложения, описывающие понятие notion, ограничивают его экстенсионал, т.е. не любая комбинация сущностей понятий notion1, …, notionM может стать сущностью понятия notion, то имеем ассоциацию понятий. В противном случае задается их агрегация и необходимости в определения понятия-связи link нет. ♦

(…) notion (notion1 … notionM) …	(…) notion1 (link …) … (…) notionM (link …)

Рис. 3.16. Выражение агрегации и ассоциации понятий

Пример 3.32. Выражение абстракций специализации и декомпозиции может быть осуществлено предложениями, приведенными на рис. 3.17 и помеченными знаком *. Здесь notionG1, …, notionGN – обобщаемые понятия; notionA1, …, notionAM – агрегируемые понятия, notion – понятие-обобщение и понятие-агрегат.

Предложения вида notion→notionG1 выражают специализацию понятий, а предложения вида notion 'notionA1'→ notionA1 – их декомпозицию. ♦

```
(notionG1 … notionGN ) notion
                (notionA1 … notionAM)
        …
(…) notionG1 (…)
        …
(…)   notionGN (…)
        notion {}*
```

```
        …
(…) notionA1 (…)
        notion 'notionA1' {}*
        …
(…) notionAM (…)
        notion 'notionAM' {}*
        …
```

Рис. 3.17. Выражение специализации и декомпозиции

3.3.6. Контекстные условия

Известно, что контекстно-свободные грамматики не отражают синтаксис современных языков программирования. Это обстоятельство вызвано тем, что эти языки не являются контекстно-свободными и имеют более сложную синтаксическую структуру [28, 200]. Синтаксические правила, которые не описываются контекстно-свободными грамматиками, называются ***контекстными условиями***.

Контекстные условия в протоязыке задаются для имен понятий notion, имен аспектов aspect и лексем lexeme. На рис. 3.14 элементы определения контекстных условий для перечисленных нетерминальных понятий протоязыка выделены курсивом: это ***notion***, ***aspect*** и ***terms***.

Задание имени понятия ***notion*** как терминального понятия протоязыка осуществляется при описании сущности essences (правило 3). После такого определения имена понятий могут появляться при описании абстракций в списке notions (правила 4, 5) и как элементы item при определении синтаксиса предложений (правила 9, 10).

Имена аспектов ***aspect*** служат для задания имен конструкций pragmatic (правило 18), назначение которых – задание имен той или иной семантической интерпретации фрагментов текста phrase (правило 21), а также text в описании семантики (правило 18), текстов грамматического разбора (правило 16), компиляции (правило 17) и описания ситуации (правило 19).

Лексемы lexeme в протоязыке задаются в виде термов term и шаблонов pattern (правило 12). Как терм, так и шаблон используются для задания терминальных понятий ***terms*** определяемого языка (правила 13, 14). Причем последние сохраняются не в таблице идентификаторов, как это было для понятий и аспектов, а непосредственно в структуре определяемого предложения.

Заметим, что имена понятий notion являются терминальными понятиями протоязыка и одновременно нетерминальными понятиями определяемого языка. Имена аспектов aspect, как и имена понятий notion являются терминальными понятиями для протоязыка, но в определяемом языке имена аспектов интерпретируются как терминальные понятия. И наконец, лексемы lexeme представляются терминальными понятиями ***terms*** определяемого языка.

3.3.7. Прагматика

Семантику предложений понятийной модели будем задавать в виде текста, который построен по правилам определяемого языка и является описанием решения некоторой задачи или подзадачи в аспекте различных проблемных ситуаций. Для этого каждое предложение sentence дополним описанием его семантики semantic, которую выразим как совокупность определений pragmatic (правила 8, 15). Описание задает семантическую интерпретацию понятия, реализуемую после компиляции предложения в виде ***императива*** – некоторой последовательности действий, которую система программирования выполняет всякий раз, когда понятие выражается этим предложением (правило 18).

В контекстной технологии императивы могут задаваться последовательностью команд аппаратной платформы, текстом на создаваемом языке, программой на некотором целевом языке и др. Система контекстного программирования выполняет трансляцию текста описания в императив (код аппаратной платформы) непосредственно или организует такую трансляцию путем вызова соответствующих средств целевой системы программирования. По своей сути императивы являются единицами вызова, в то время как предложение – описание синтаксиса таких вызовов.

3.3.8. Аспекты

Понятийная структура проблемной области может быть общей для нескольких задач. Однако решение таких задач может преследовать различные цели, в том числе и исключающие друг друга. Для отражения этой ситуации в контекстной технологии предусмотрена возможность задания нескольких прагматик pragmatic (правило 15). Для отличия одного прагматики от другой они именуются и задаются в виде aspect {…}, где aspect – имя определяемого императива, который соответствует именованной семантической интерпретации предложения или его аспекту (правило 18).

При использовании именованных императивов, определяемых содержательной постановкой задачи, ситуационное описание необходимо дополнить указанием на одну из возможных его семантических интерпретаций. В итоге, ситуационная часть модели может стать независимой от содержательной интерпретации конкретной прикладной задачи и задавать некоторое общее решение для целого класса задач.

3.3.9. Семантическое замыкание

Семантика понятий задается путем декларации первичных семантических категорий, которые непосредственно реализуются целевой платформой (база семантической индукции) и выражением сложного смысла текстом на проблемном языке, для чего используются ранее определенные семантические единицы (индуктивный семантический переход).

Для обеспечения полноты и целостности понятийной модели во всех ее частях проблемный язык используется не только при описании ситуационной части (text в строке 19), но и для задания семантики предложений и описания процесса разбора и компиляции (text в правилах 18, 16, 17). Это позволяет использовать для обработки этих описаний одни и те же средства.

Семантика различных частей понятийной модели определяются как текст, состоящий из фраз phrase (правило 20), построенный по правилам, задаваемым в понятийной модели. Очевидно, для привязки модели к целевой платформе необходимо некоторое множество предложений реализовать низкоуровневыми средствами или на целевом языке.

3.3.10. Демонстрационный пример

Рассмотрим текст, приведенный на рис. 3.18. В рассматриваемом примере Constant обозначает понятие «логическая константа», Variable – «пропозиционную переменную», а Logic – «логическое значение». Остальные понятия соответствуют более сложным представлениям: Negation обозначает понятие «отрицание», Conjunction – понятие «конъюнкция», а Disjunction – понятие «дизъюнкция».

Очевидно, наиболее сложным по своей структуре является понятие Boolean, обозначающее «булево выражение». Constant, Logic, Negation, Conjunction и Disjunction являются частными случаями Boolean. Однако выражение этих частных случаев осуществляется по-разному. Для описания понятия Negation используется понятие Logic и Constant, т.е. понятие Negation получено в результате обобщения понятий Logic и Constant. В свою очередь, понятия Logic и Constant являются конкретизацией понятия Negation. Последнее означает, что любое выражение в тексте понятий Logic или Constant является и выражением некоторого частного случая для обобщающего понятия Negation.

Императивы в примере определены в виде последовательности ассемблерных команд достаточно распространенной аппаратной платформы на базе процессора Intel [104]. Для доступа к данным и организации их временного хранения использован аппаратный стек, а для хранения переменных – память произвольного доступа с линейной организацией.

```
() Variable
    ″[A-Za-z][A-Za-z0-9]*″ […] {}
() Constant
    'false' [ asm{ mov eax, 0; push eax } ] {}
    'true' [ asm{ mov eax, -1; push eax } ] {}
        bit[ asm{mov eax, 1; push eax} ] {}
(Variable) Logic
    Variable [ asm{ pop ebx; mov eax, [ebx]; push eax } ] {}
    Integer [asm{ pop eax; cmp eax, 0; je label; mov eax, -1; label: push eax }] {}
        bit[ asm{pop eax; and eax, 1; push eax } ] {}
    '(' Boolean ')' {}
(Constant Logic) Negation
    'not' Logic [ asm{ pop eax; not eax; push eax } ] {}
(Negation) Conjunction
    Negation 'and' Negation [asm{ pop eax; pop edx; and eax, edx; push eax }] {}
(Conjunction) Disjunction
    Conjunction `a` 'or' Conjunction `b` { not a or not b }
(Disjunction) Boolean
    Disjunction `a` 'imp' Disjunction `b` { not a or b }
```

Рис. 3.18. Понятийная модель «Логические выражения»

В предложении ″[A-Za-z][A-Za-z0-9]*″ текст компиляции (не показан) описывает создание переменной путем включения имени переменной в таблицу идентификаторов и выделения для ее хранения памяти требуемого объема.

Логические константы 'false' и 'true' реализованы как занесение нуля и минус единицы на вершину стека, для чего использованы команды загрузки в регистр константы и записи регистра в стек.

Переменная определена адресом ячейки памяти, в которой хранится ее текущее значение. Для получения значения логической переменной Variable ее адрес извлекается из стека, содержимое адресуемой ячейки пересылается в регистр, который затем сохраняется в стеке.

Для преобразования целого числа в булево значение использовано предложение Integer. Если число на вершине стека не равно нулю, то в регистр записывается минус единица, в противном случае там уже имеется требуемое значение. Результат преобразования сохраняется в стеке.

В свою очередь предложение '(' Boolean ')' не нуждается в императиве, т.к. его роль заключается в задании правил разбора и приоритета булева выражения, заключенного в круглые скобки. При изменении решаемой задачи возможно появление императивов и у этого предложения.

В примере определены две семантические интерпретации: семантическая интерпретация по умолчанию (без имени) и с именем bit. В первом случае логический ноль кодируется арифметическим нулем, а логическая единица – минус единицей. Во втором случае логическая единица кодируется арифметической единицей. Заметим, что для порождения исполняемого кода аппаратной платформы используется аспект asm (в примере не определен).

Анализ примера показывает, что все предложения, кроме двух последних, реализованы низкоуровневыми средствами. Причем этих предложений уже достаточно для определения семантики последних предложений на проблемном языке. Для дизъюнкции и импликации справедливы тождества $a \vee b = \overline{a}\,\&\,\overline{b}$ и $a \to b = \overline{a} \vee b$. Отсюда получаем предложения

```
Conjunction `a` 'or' Conjunction `b` { not a or not b }
Disjunction `a` 'imp' Disjunction `b` { not a or b },
```

где a – является алиасом первого понятия предложения (первое вхождение Disjunction), b – второго. Заметим, что из соображений эффективности семантику этих предложений можно было определить низкоуровневыми средствами.

Для описанной понятийной модели текст ситуационной части <(not x or y) and z> приведет к вычислению булева выражения при арифметическом кодировании логических значений, а bit < (not x or y) and z > породит исполняемый код при битовом представлении.

Для рассмотренной предметной области возможно задание семантики понятийной модели в других проблемных областях, например, в области нечеткой логики. Тогда текст fuzzy < (not x or y) and z > может быть интерпретирован как нечеткое логическое выражение, где fuzzy – имя аспекта для нечетких вычислений.

3.4. Семантика проблемного языка

В контекстной технологии решение прикладных задач осуществляется на основе создания проблемного языка, что требует использования развитых средств для описания его синтаксиса и семантики. Под семантикой обычно понимается отношение между выражениями искусственных формальных языков и их интерпретацией в некоторой модели мира. Иными словами «семантика» является техническим способом «дать значение» неинтерпретированным знакам «синтаксиса» [22, ст. Семантика].

3.4.1. Формальная семантика

С формальной точки зрения множество S называется ***формальной семантикой*** (множеством смыслов) формального языка L, если задано интерпретирующее отображение φ языка L в S. Тогда для строки $\alpha \in L$ элемент $\varphi(\alpha) \in S$ будет ее смыслом [140,

с. 325]. Отсюда формальная семантика – это способ попарного сопоставления строк, которые имеют структуру, но не имеют значения, с моделями – множеством строка, которые также имеют структуру, но не имеют значения. Эти модели понимаются как то, что «дает значение» строкам формального языка, семантическая реализация языка совпадает с понятием интерпретации этого языка [204, с. 322].

Под интерпретацией в широком смысле понимают приписывание значений исходным выражениям некоторого исчисления, в силу чего получают смысл все правильно построенные выражения данного исчисления, а интерпретированное исчисление является формализованным языком, в котором формулируются и доказываются различные высказывания, имеющие смысл. ***Интерпретация*** – это распространение положений какой-либо формальной теории на объекты реального мира или другой формальной теории, являющейся в этом случае метатеорией [153, с. 86]. Интерпретация наделяет смыслом знаки и формулы формальной теории, а теоремы формальной теории соответствуют истинным утверждениям об объекте.

Известны три основных метода описания семантики формальных языков: операционный, аксиоматический и алгебраический [46, с. 24]. ***Операционная*** семантика – это сопоставление строкам формального языка машинно-ориентированных денотатов, или последовательности команд вычислительной машины. ***Аксиоматическая*** семантика – сопоставление строкам формального языка формул аксиоматической теории, задаваемой аксиомами и правилами вывода. В свою очередь ***алгебраическая*** семантика – это сопоставление строкам формального языка алгебраических денотатов – формул алгебраической теории, задаваемой множеством и замкнутыми на этом множестве операциями. Частным случаем алгебраической семантики является ***денотационная*** семантика, при описании которой происходит сопоставление строкам формального языка функциональных денотатов, для выражения которых вместо операций используются функции.

В частности, ГОСТ 34.320-96 рекомендует определять смысл (семантику) формального языка аксиоматическим методом через множество неопределенных понятий – примитивов (семантических категорий). Другие понятия тогда получают смысл, выводимый из неформальных примитивных понятий с помощью формальных определений. Соответствующие аспекты смысла каждого примитивного понятия формально вводятся посредством задания аксиом, которые считаются истинными, а правила вывода сохраняют истинность [89, с. 14]. Такая семантика получила название ***логической***.

Многообразие возможных подходов к формальному определению семантики можно представить двумя основными типами, а именно, семантиками, ориентированными на компиляцию, и семантиками, ориентированными на интерпретацию. В первом случае се-

мантика описывается в виде множества преобразований, выполняемых над деревом грамматического разбора нетерминальных понятий языка, или некоторой другой синтаксической моделью языка. Во втором случае семантика задается на некотором метаязыке и представляет собой описание, сопровождающее конструкции формального языка, т.е. представляет собой описание некоторых преобразований синтаксически правильных конструкций языка.

Проблемы описания семантики формальных языков связаны, в основном, с необозримостью получаемых описаний, выполняющихся через небольшое множество базовых семантических категорий. Суть предлагаемого подхода заключается в том, что категории, необходимые для описания семантики формальных языков, не задаются заранее, а определяются по мере необходимости, в процессе описания языка и средствами самого языка. Такой подход позволяет определить семантику формального языка внутренними, а не внешними по отношению к нему средствами. Иными словами, формальный язык и семантический язык определяются одновременно и взаимно обусловлено. Отсюда, в частности, следует, что не должно существовать ни одной семантической категории, которая определялась бы вне формального языка, средствами которого осуществляется описание семантики.

Последнее достигается путем наличия в протоязыке контекстной технологии специальных выразительных средств для декларации базовых примитивов – первичных семантических категорий, непосредственно реализуемых целевой вычислительной платформой (см. 4.4.4 на с. 212). Другая часть выразительных средств контекстной технологии позволяет посредством описания семантики некоторого выражения языковыми конструкциями, определенными ранее, определять более сложный смысл через уже определенные смысловые единицы (см. пример на рис. 3.18, с. 174).

3.4.2. Семантическая индукция

Описание семантики понятийной модели осуществляется на основе ***семантической индукции***, заключающейся в том, что семантические категории модели определяются по мере необходимости, в процессе определения проблемного языка и средствами самого языка, т.е. проблемный язык одновременно является и семантическим языком, служащим для описания семантики.

Базой семантической индукции являются базовые примитивы, или первичные семантические категории, которые непосредственно реализуются целевой вычислительной платформой и декларируются протоязыковыми средствами перед использованием.

Предположением индукции служат все ранее определенные семантические категории. Семантическая категория соответствует определяемому в модели понятию и выра-

жается совокупностью прагматик тех предложений, которые предназначены для выражения этого понятия в тексте.

Индуктивный переход осуществляется всякий раз, когда описывается новая семантическая категория в одном из аспектов своей интерпретации. Для выражения индуктивного перехода используется текст, сформированный по правилам уже определенного до этого проблемного подязыка.

В процессе каждого индуктивного перехода происходит описание одной из прагматик предложения. Совокупность таких прагматик образует семантику предложения. Семантика понятия раскрывается как объединение семантик предложений, служащих для выражения этого понятия в тексте. Таким образом, ***заключением*** семантической индукции является определение новой семантической категории.

Так как в рамках контекстной технологии заявлен некоторый универсальный метод для описания семантики формальных языков, покажем реализацию известных способов описания семантики, таких, например, как синтаксически-управляемые схемы и атрибутные грамматики. Укажем, что пример, приведенный на рис. 3.18 следует рассматривать как одну из возможных реализаций операционной семантики для понятийной модели «Логические выражения».

3.4.3. Синтаксически-управляемые схемы

Синтаксически-управляемые схемы [10] и атрибутные грамматики [126] позволяют задавать соответствия между текстами входного и выходного языков, называемые ***переводом***. Такие соответствия отражают структурные или синтаксические свойства входных и выходных текстов.

Пример 3.33. Рассмотрим пример реализации в рамках контекстной технологии синтаксически-управляемого перевода (рис. 3.19).

Перевод осуществим на примере формального дифференцирования выражений, включающих целочисленные константы, переменную x, функции sin и cos, а также алгебраические операции: изменение знака -, сложение +, вычитание –, умножение *.

Процесс перевода опишем понятийной моделью, показанной на рис. 3.19, где знаком & обозначена операция конкатенации строк, которая определена при описании, например, понятия String (в примере не показано).

Свяжем с каждым предложением модели два перевода, формируемые императивами без имени и именованными dif. Неименованный императив указывает на то, что выражение не дифференцируется, а именованный императив dif – что выражение необходимо продифференцировать. Формальная производная некоторой строки s – это dif{s}, где dif{…} задает имя семантической интерпретации текста в фигурных скобках.

Текст ситуационной части 'dif { sin(5*cos(x)) – x*x }', включающий выражение, которое необходимо продифференцировать, будет переведен и представлен в виде:

'(cos(5*cos(x))*(5* – sin(x)*(1) + 0*cos(x)*(1))–(x*1 + 1*x)'.

Можно определить именованный императив equ, который выполнит очевидные тождественные преобразования выражений, получаемых после дифференцирования, например путем задания ситуации 'equ { dif { sin(5*cos(x)) – x*x } }'. В результате его применения получим: -5*(cos(5*cos(x))*sin(x))–2*x. ♦

```
() Expression ()
    ″ [0-9]+ ″ `n`
        { n }
        dif { '0' }
    'x'
        { 'x' }
        dif { '1' }
    '(' Expression ')' `exp`
        { '(' & exp & ')' }
        dif { '(' & dif { exp } & ')' }
    'sin' '(' Expression `exp` ')'
        { 'sin(' & exp & ')' }
        dif { 'cos(' & exp & ')*(' & dif { exp } & ')' }
    'cos' '(' Expression `exp` ')'
        { 'cos(' & exp & ')' }
        dif { '–sin(' & exp & ')*(' & dif { exp } & ')' }
    '–' Expression `exp`
        { '–' & exp }
        dif { '–' & dif { exp } }
    Expression `exp1` '*' Expression `exp2`
        { exp1 & '*' & exp2 }
        dif { '(' & exp1 & '*' & dif { exp2 } & '+' & dif { exp1 } & '*' & exp2 & ')' }
    Expression `exp1` ″+ | –″ `oper` Expression `exp2`
        { exp1 & oper & exp2 }
        dif { '(' & dif { exp1 } & oper & dif { exp2 } & ')' }
```

Рис. 3.19. Формальное дифференцирование выражений

3.4.4. Атрибутные грамматики

Формализм атрибутных грамматик оказался очень удобным средством для описания семантики языков программирования. Атрибутные грамматики позволяют описывать синтаксис и семантику формальных языков благодаря наличию атрибутов, связанных с каждым нетерминальным знаком, и правил вычисления этих атрибуты. Вместе с тем вы-

яснилось, что реализация вычислителей для атрибутных грамматик общего вида сталкивается с большими трудностями.

Реализация перевода атрибутными грамматиками сопряжена с трудностями описания контекстно-зависимых условий или смысла конструкций языков программирования. Эти трудности связаны как с самим формализмом, так и с некоторыми технологическими проблемами [17]. К трудностям первого рода можно отнести неупорядоченность процесса вычисления выходных атрибутов при жесткой упорядоченности синтаксического анализа входного текста. Это несоответствие требует использования искусственных приемов для их сочетания. Технологические трудности определяются низкой эффективностью трансляторов, сгенерированных с помощью атрибутных систем, что связано с большим расходом памяти и наличием искусственных приемов итерационного вычисления атрибутов.

Особую трудность при использовании атрибутных грамматик вызывает из сложность, заключающаяся во введение в описание грамматик вспомогательных структур данных и функций, выраженных на одном из языков программирования, а также большого числа операций для передачи контекста между областями видимости нетерминалных знаков, соседствующих в дереве разбора [6].

В связи с этим было сделано множество попыток рассматривать те или иные классы атрибутных грамматик, обладающих требуемыми свойствами. К числу таких свойств относятся прежде всего простота алгоритма проверки атрибутной грамматики на зацикленность и простота алгоритма вычисления атрибутов [208].

Рассмотрим пример реализации в контекстной технологии трансляции текстового представления числа в формате с фиксированной запятой в его двоичный эквивалент. Семантика такой задачи традиционно описывается атрибутной грамматикой.

Пример 3.34. Для выражения числа с фиксированной запятой Fixed введем два дополнительных понятия: Integer (целая часть) и Fraction (дробная часть). Понятию Integer припишем атрибут int, равный значению целой части числа, а понятию Fraction – атрибут frac, равный дробной части. Атрибуты сопоставим сущностям понятий, определяемым соответствующими предложениями (рис. 3.20).

В понятийной модели использованы не определенные в примере понятия Number и Float, содержащие необходимые арифметические операции целочисленной арифметики и арифметики с плавающей запятой. По терминологии атрибутных грамматик атрибут int является синтезируемым, в то время как атрибут frac – наследуемым.

```
(Number) Integer ()
        ″[0-9]″ `digit` { digit }
        Integer `int` ″[0-9]″ `digit` { int * 10 + digit }
(Float) Fraction ()
        ″[0-9]″ `digit` { digit }
        ″[0-9]″ `digit` Fraction `frac` { digit + frac / 10 }
() Fixed (Integer Fraction)
        Integer `int` '.' Fraction `frac` { int + frac / 10 }
```

Рис. 3.20. Число с фиксированной запятой

В отличие от атрибутных грамматик, не имеющих средств задания порядка вычисления атрибутов, последовательность вычисления сущностей в примере определена выразительными средствами самой модели. Для правильного вычисления синтезируемого атрибута понятие Integer определено как подлежащее грамматическому разбору справа налево (от младших цифр к старшим). В свою очередь для вычисления наследуемого атрибута понятие Fraction подвергается разбору слева направо (от старших цифр к младшим).

Для атрибутного описания семантики правил грамматического разбора в примере использовано обращение к сущностям распознаваемых понятий, выражаемых соответствующими алисами, выступающих в роли имен атрибутов. ♦

Заметим, что в понятийной модели с каждым понятием изначально связывается атрибут, соответствующий обозначаемой в данный момент сущности-результату. При необходимости, эти сущности можно описать как составные – принадлежащие сложным понятиям, образованным как агрегация других понятий, т.е. содержащие требуемое число атрибутов. В связи с тем, что семантика каждого правила грамматики выражается на проблемном языке, использование контекстной технологии позволяет уменьшить сложность описания семантики языков программирования. Такое уменьшение достигается за счет введения и использования понятий-атрибутов, которые наиболее естественным и простым способом выражают семантику описываемого правила.

В противоположность этому, при описании семантики языков программирования атрибутными грамматиками, используется специальный встроенный язык, предназначенный для определения атрибутов и выполнения над ними некоторых операций [208].

3.5. Заключительные замечания

Пример использования контекстной технологии обработки данных для решения одной из прикладных задач приведен в Приложении 1.

Укажем также на ряд публикаций, в которых приведены результаты, посвященные понятийному анализу и контекстной технологии. Так в работе [54] высказана идея об оп-

ределении конструкции языка средствами самого языка и в процессе его определения, введена контекстная технология программирования и рассмотрено использование контекстных грамматик для описания определяемого языка. В работе [57] показана возможность использования самоопределения языка при создании программ, описаны общие принципы контекстной обработки данных и приведен первый вариант формальной грамматики протоязыка. Вопросы, связанные с выразительными качествами протоязыка, обсуждаются в работе [65].

В работе [110] затронут вопрос о возможности универсального описания семантики формальных языков. Общим вопросам представления знаний на основе понятийных структур посвящена работа [68], где рассмотрены вопросы создания, хранения и использования знаний, представленных в виде откомпилированных понятийных моделей. Необходимость реализации семантической замкнутости при представлении знаний обосновывается в работе [67].

Работы [106, 107, 108] посвящены прикладным вопросам контекстной технологии, а в работе [109] рассмотрено ее применение при математическом моделировании. В работе [111] описана понятийная структура более сложной проблемной области, чем это показано в примерах.

Подходы к представлению понятийных моделей различных проблемных областей рассмотрены в работах [111, 67]. В работе [66] предложена организация процессора с сокращенным набором команд, оптимизированная для работы с данными и программами, организованными в виде словарей (понятийных структур).

Выводы к Главе 3

1. Показано, что протоязык контекстной технологии позволяет выразить абстракции агрегации-декомпозиции, ассоциации-индивидуализации, обобщения-специализации и типизации-конкретизации в более широком смысле, чем это реализовано в объектно-ориентированной технологии.

2. Разработан метод семантической индукции, предназначенный для описания семантики формальных языков и обобщающий такие известные методы как денотационный, аксиоматический и операционный.

3. Предложен принцип семантического замыкания понятийной модели, заключающийся в описании семантики на создаваемом проблемном языке и предусматривающий выразительные средства для задания необходимых семантических категорий по мере необходимости, в процессе описания конструкций определяемого проблемного языка и средствами самого языка.

4. Реализован и исследован механизм аспектов, позволяющий использовать одну и ту же понятийную модель для решения различных прикладных задач, объединенных общей понятийной структурой, но различающихся правилами выражения понятий или их прагматиками.

Глава 4.
Система программирования

Настоящая глава посвящена исследованию системы контекстного программирования, которая является одной из реализаций технологии контекстной обработки данных на основе спецификации предметной области в форме языка-письма. Однако не видится препятствий для построения на тех же принципах, например, системы обработки речевых данных, реализующей другой способ обработки – в форме языка-речи.

Формализация знаний о предметной области осуществляется путем построения ее понятийной модели. Понятийная модель является выражением понятийной структуры как некоторой концептуальной схемы решения задачи. В отличие от других парадигм в области программирования, например структурной и объектно-ориентированной, контекстная парадигма определяет методологию постановки и решения задач специфическими языковыми средствами, эквивалентными по выразительным возможностям формализму контекстных грамматик.

В описываемой системе контекстного программирования в качестве программы используется описание понятийной модели предметной области, дополненное решением прикладной задачи, выраженное на создаваемом в процессе создания понятийной модели проблемном языке (специализированном предметном языке). Определение семантики проблемного языка определим на этом же языке, т.е. в процессе описания понятийной модели создается и семантический язык. Отсюда, в частности, следует, что понятийная модель и решение задачи должны быть подвергнуты грамматическому разбору и компиляции таким образом, чтобы откомпилированные ранее предложения могли быть использованы при грамматическом разборе и компиляции следующих.

Основные результаты, представленные в настоящей главе, опубликованы в работах [63, 69].

4.1. Содержательная постановка задачи

В ***понятийной модели***, используемой в системе контекстного программирования, прикладные знания будем структурировать в виде понятийной структуры предметной области и выражать описанием синтаксиса форм выражения понятий и определением семантики для каждого такой формы.

В ***понятийной структуре*** определяются способы образования (абстрагирования) понятий в виде отображений одних понятий в другие. Для представления и манипулирования знаниями для каждой предметной области и для каждого класса решаемых задач,

задаваемых некоторой активной проблематикой, будем строить свой ***проблемный язык***, или специализированный предметный язык, позволяющий адекватным образом описать как множество известных фактов относительно предметной области, так и решение или свойства решения стоящих прикладных задач.

Построение проблемного языка осуществим путем включения множества понятий предметной области в понятия языка, а формы выражения понятий – в его языковые конструкции. Для задания синтаксиса понятий воспользуемся грамматической формой, позволяющей декларировать один или несколько способов выражения понятий в тексте. Семантику понятий будем задавать для каждой такой формы на некотором подязыке проблемного языка, который к тому времени уже определен в понятийной модели. Учет прагматики осуществим путем многоаспектного описания семантики, для чего воспользуемся механизмом аспектов, позволяющим задавать множество именованных семантических интерпретаций для каждой формы выражения понятия в тексте.

В итоге, в процессе программирования, описание фактов предметной области и решаемых в ней задач осуществим в наиболее компактной, легко читаемой и верифицируемой форме на проблемном языке. Последнее есть следствие приближения проблемного языка к формам словесно-логического выражения накопленных знаний о рассматриваемой предметной области. Такой подход позволит облегчить преодоление семантического разрыва между высокоуровневыми представления и теми средствами, которые служат для описания этих представлений.

Для повышения эффективности системы контекстного программирования применим принцип ***контекстной интерпретации*** текстов, заключающийся в определении семантического значения некоторого фрагмента по его месту в тексте. В общем случае одна и та же последовательность терминальных знаков может быть разбита на термы различным образом. Более того, один и тот же терм может быть сопоставлен различным лексемам, т.е. одна и та же последовательность терминальных знаков может служить носителем различных смыслов. Следовательно, при контекстной интерпретации разделение анализа текста на фазы лексического и синтаксического анализа не представляется возможным. Необходимо разработать новый метод грамматического разбора учитывающий эту особенность. Для этого применим метод грамматического разбора, названный ***разнесенный*** и заключающийся в разделении определения применимости предложения при анализе текста на две части – на контекстное сопоставление предложений, осуществляемое при просмотре текста назад, и структурное распознавание, выполняемое при просмотре вперед.

Результат разнесенного грамматического разбора будем сохранять в виде, позволяющем использовать эти результаты при разборе следующих предложений без повторе-

ния грамматического разбора ранее определенных. Следовательно, понятийная модель нуждается в ***компиляции***, т.е. в получении некоторого ее представления, отличающегося от исходного (текстового) и служащего для повышения эффективности системы контекстного программирования.

В заключение исследуем особенности реализации системы контекстного программирования, которые вытекают из методологии понятийного анализа и принципов контекстной технологии.

Материалы, приведенные в настоящей главе, получены в результате экспериментальной части исследований, которая выполнена на действующей системе контекстного программирования.

4.2. Разнесенный грамматический разбор

Грамматический разбор текста, как правило, делится на фазы лексического и синтаксического анализа [36]. В фазе лексического анализа входной текст, рассматриваемый как поток знаков, разбивается на лексемы.

Лексические анализаторы реализуются двумя способами: либо в виде подпрограммы, вызываемой синтаксическим анализатором для получения очередной лексемы, либо в виде модуля, осуществляющего полный просмотр текста (проход), результатом которого является входной текст, разбитый на лексемы и рассматриваемый как поток лексем [232].

В свою очередь синтаксический анализ предназначен для получения структуры текста, где под структурой понимается дерево грамматического разбора [205]. Чаще всего используется либо LL(1)-анализ и его вариант – рекурсивный спуск, либо LR(1)-анализ и его варианты: LR(0), SLR(1), LALR(1) и др. [10, 122]. Рекурсивный спуск применяется при ручном, а LR(1) – при автоматизированном построении синтаксических анализаторов [9].

Рассмотрим особенности грамматического разбора в контекстной технологии, которые отличают ее от других технологий программирования.

4.2.1. Структура предложения

Разделим структурно каждое предложение понятийной модели, описывающее некоторое понятие notion, на четыре области: область контекста context, лексему lexeme, область разбора parsing и результат. Расширенный фрагмент протоязыка для конструкции sentence приведен на рис. 4.1.

Контекстом предложения назовем последовательность понятий, предшествующих первой лексеме. Под ***лексемой*** предложения будем понимать его первую лексему. Область ***разбора*** определим как часть предложения, непосредственно следующую за первой лексемой. И наконец, ***результат*** предложения – это понятие notion, определяемое

предложением полностью (когда больше нет предложений с тем же результатом) или частично (если такие предложения имеются).

```
sentence  →  context [lexeme [parsing]]
             lexeme [parsing]
context   →  notion [context]
lexeme    →  term
             pattern
parsing   →  item [parsing]
item      →  notion
             lexeme
```

Рис. 4.1. Структура предложения

4.2.2. Контекстное сопоставление

Пусть имеется структура данных, которую назовем ***текущим контекстом*** и реализуем в виде стека контекста (рис. 4.2). Текущий контекст содержит последовательность понятий, которые уже распознаны, а соответствующие им терминальные знаки интерпретированы и извлечены из входного потока (сопоставлены лексемам предложений). Следовательно, в каждый момент времени грамматический анализатор имеет некоторое текущее состояние, определяемое состоянием стека контекста и текущей позицией во входном потоке.

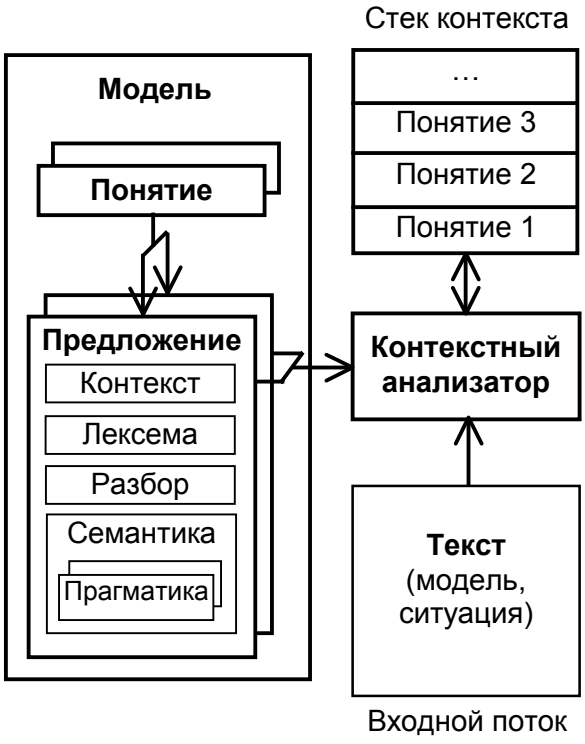

Рис. 4.2. Контекстное сопоставление

Поиск предложений, применимых в текущем состоянии анализатора осуществим путем сравнения контекста предложений модели с текущим контекстом. Предложения, имеющие контекст, сопоставимый с текущим, могут использоваться в текущем состоянии анализатора. Здесь под сопоставимостью понимается соответствие понятий на вершине стека контекста понятиям из контекста предложений, возможно с учетом их эквивалентных преобразований. Под эквивалентным преобразованием понимается замена понятия-обобщения на его конкретизацию.

Однако из сопоставимых предложений следует выбрать те, лексема которых представляется термом, находящимся в текущей позиции входного потока. В итоге, из всего множества предложений дальнейшему анализу подвергаются те, которые применимы в текущем состоянии анализатора. Тем самым осуществляется ***контекстное сопоставление*** предложений путем просмотра назад и определение применимости сопоставленных предложений путем сравнения терма из входного потока с лексемой предложения.

Заметим, что грамматический разбор текста, подлежащего просмотру назад, уже выполнен и представлен в виде текущего контекста.

4.2.3. Структурное распознавание

После выявления предложения, применимого в текущем контексте, наступает этап разбора его оставшейся части, которая еще не сопоставлена входному потоку. Для этого грамматический анализатор запоминает свое состояние, очищает текущий контекст и выполняет просмотр вперед, начиная с текущей позиции входного потока.

Если элементом разбора при просмотре вперед является лексема из области разбора предложения, то выполняется ее сравнение с термом входного потока. При успешном сравнении терм, соответствующий лексеме предложения, извлекается из входного потока. В случае неудачи состояние анализатора восстанавливается, а анализируемое предложение считается нераспознанным.

Если требуется распознать понятие из области разбора предложения, то для анализа выбираются те предложения, которые сопоставимы с текущим состоянием анализатора и имеют это понятие в качестве своего результата. При удачном распознавании анализатор переходит к следующему элементу области разбора. В случае неудачи анализируемое предложение считается нераспознанным, состояние анализатора восстанавливается, и он переходит к разбору следующего применимого предложения.

Если все элементы области разбора сопоставлены входному потоку, то предложение считается ***структурно распознанным***, выполняется восстановление исходного контекста, в котором контекст предложения заменяется на понятие-результат. При этом состояние входного потока, полученного после извлечения текста, сопоставленного области

разбора предложения, не изменяется и объявляется текущим. Тем самым из входного потока извлекается распознанная часть текста, представленная текущим контекстом.

4.2.4. Приоритет предложений

Для упорядочивания грамматического разбора установим приоритетность предложений и понятий исходя из естественных представлений, выражаемых их местом в описании предметной области.

Для реализации приоритетности одних предложений над другими всем им припишем приоритет в соответствии с их порядком в понятийной модели. Этот приоритет, определяемый содержательным описанием предметной области, назовем ***естественным***, или абсолютным, приоритетом предложений. При проверке применимости из двух сопоставимых предложений выбирается то, которое определено позже (далее по тексту), т.е. имеет меньший приоритет. Поясним роль приоритетов предложений в процессе грамматического разбора на примере.

Пример 4.1. Рассмотрим понятийную модель, описывающую булевы выражения (рис. 4.3). В модели определено понятие Boolean, а предложения расположены в порядке естественного приоритета операций булевой алгебры, т.е. переменные имеют наибольший приоритет (первое предложение), в то время как операция импликации imp имеет наименьший приоритет (последнее предложение).

```
() Boolean
″[A-Za-z][A-Za-z0-9]*″ {}
″false|true″ {}
'(' Boolean ')' {}
'not' Boolean {}
Boolean 'and' Boolean {}
Boolean 'or' Boolean {}
Boolean 'imp' Boolean {},
```

Рис. 4.3. Понятийная модель «Исчисление высказываний»

При грамматическом разборе текста 'true or x and y' с пустым текущим контекстом сопоставимыми являются первые четыре предложения. Однако применимы только первое и второе. Так как разбору в первую очередь подлежит второе предложение, то терм 'true' интерпретируется как логическая константа.

Если поменять местами первое и второе предложения, то терм 'true' будет интерпретирован как переменная с именем true. В последнем случае, если такой переменной не окажется, предложение помечается как неприменимое, и разбору подлежит следующее предложение, расположенное выше и интерпретирующее true как константу. ♦

4.2.5. Приоритет понятий

При наличии в понятийной модели дифференциации понятий порядок отбора предложений при грамматическом разборе несколько видоизменяется, так как в этом случае необходимо учитывать возможность представления понятия-обобщения другим понятием, являющимся его конкретизацией.

При структурном распознавании внутри группы предложений с одним и тем же понятием-результатом приоритеты предложений, как и ранее, задаются их абсолютным приоритетом. Однако приоритет самого понятия-результата определим относительным местом этого понятия в понятийной структуре, образованной отображениями обобщения.

В этом случае из двух предложений, выражающих сопоставимые понятия-результаты, выбирается то, которое имеет меньший приоритет своего результата, т.е. при структурном распознавании предпочтение отдается конкретизации понятия по отношению к его обобщению. Тем самым учитываются ***относительные*** приоритеты понятий, задаваемые понятийной структурой модели.

Заметим, что и при контекстном сопоставлении, когда происходит сравнение понятий из текущего контекста с понятиями из контекста предложения, следует учитывать возможность представления одного понятия другим на основе установленных связей обобщения. Поясним роль приоритетов понятий на примере.

Пример 4.2. Преобразуем понятийную модель из примера 4.1, детализируя ее понятийную структуру (рис. 4.4).

Продолжим грамматический разбор текста 'true or x and y' при состоянии входного потока, равном 'or x and y' и текущем контексте Constant. В этом случае анализируемыми являются предложения не только понятия Constant, но и понятий Negation, Conjunction, Disjunction и Boolean (расположены в порядке их относительных приоритетов), так как понятие Constant является конкретизацией этих понятий.

В рассматриваемом случае сопоставимыми являются предложения 16, 14 и 12, контекст которых сопоставим с текущим. Однако применимо только предложение 14, в виду того, что только его лексема может быть выражена термом 'or' входного потока.

После контекстного сопоставления наступает фаза структурного распознавания единственного отобранного предложения Conjunction 'or' Conjunction, применимого в текущем состоянии контекстного анализатора. Теперь из входного потока при пустом контексте необходимо извлечь терм, выражающий понятие Conjunction или его конкретизацию: Variable, Logic, Constant или Negation (перечислены в порядке относительных приоритетов).

```
1     () Variable
2           ″[A-Za-z][A-Za-z0-9]*″ {}
3     () Constant
4           ″false|true″ {}
5     (Variable) Logic
6           Variable {}
7           Integer {}
8           '(' Boolean ')' {}
9     (Constant Logic) Negation
10          'not' Logic {}
11    (Negation) Conjunction
12          Negation 'and' Negation {}
13    (Conjunction) Disjunction
14          Conjunction 'or' Conjunction {}
15    (Disjunction) Boolean
16          Disjunction 'imp' Disjunction {}
```

Рис. 4.4. Понятийная модель «Исчисление высказываний»

Находим, что в рассматриваемом случае сопоставимыми являются предложения 2, 4, 8 и 10, а применимо только предложение 2. При применении предложения 2 из входного потока извлекается терм 'or', а текущий контекст устанавливается равным результату предложения – понятию Variable.

Продолжаем распознавание до тех пор, пока текущий контекст анализатора сопоставим с понятием Conjunction. Очевидно, при текущем состоянии входного потока 'and y' применимо предложение 12, производящее в стеке контекста искомое понятие Conjunction.

В итоге получаем схему грамматического разбора, приведенную на рис. 4.5. ♦

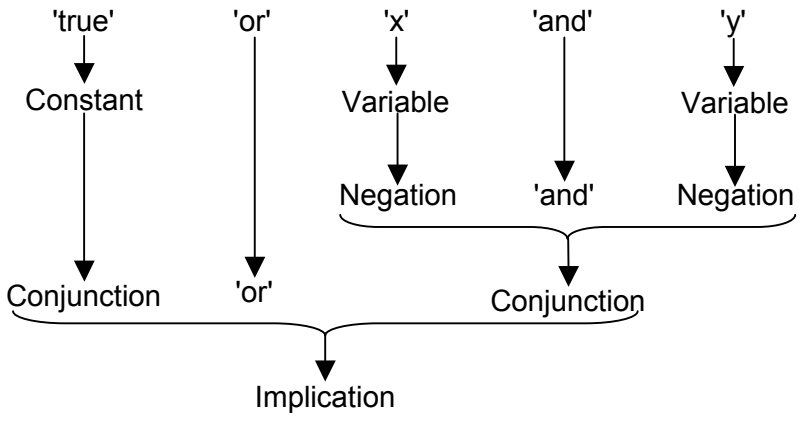

Рис. 4.5. Схема грамматического разбора

4.2.6. Конверторы

Протоязык контекстной технологии позволяет определять предложения вида sentence → context, не имеющие лексем и состоящие из одного контекста. Такие предложения будем называть конверторами. Конверторы задают денотационную форму выражения понятия, в то время как предложения, состоящие только из лексем, – сигнификативную.

В частом случае имеем конвертор одного понятия в другое, sentence → notion, что выражает семантическую синонимию понятий. Очевидно, общий вид предложения, у которого присутствуют все структурные части (контекст, лексема и разбор), позволяет выразить понятие в общем виде – произвольной языковой конструкцией.

Контекстное сопоставление и структурное распознавание следует осуществлять с учетом конверторов, устанавливающих эквивалентность одного или нескольких понятий другому.

Пример 4.3. В понятийной модели на рис. 4.4 имеются два конвертора, выраженных предложениями 6 и 7. Они устанавливаю синонимию понятия Variable и понятия Integer понятию Logic, однако обратное неверно, т.к. соответствующие конверторы в модели не определены. Назначение перечисленных конверторов – преобразование пропозиционной переменной и целого числа в булево значение.

Например, при анализе текста 'true or 2 and y' целая константа 2 благодаря конвертору 7 может быть интерпретирована как истинное булево значение, а переменная y, представленная адресом ячейки памяти, конвертором 6 преобразована в присвоенное ей значение. ♦

Следует заметить, что абстракция обобщения понятия, задаваемая списком дифференциации, может быть выражена множеством конверторов вида: sentence → notion, где notion – одно из понятий из указанного списка.

4.2.7. Неоднозначные грамматики

При разнесенном грамматическом разборе каждое предложение сопоставляется входному потоку не полностью. Область контекста предложения не подлежит сопоставлению входному потоку, она есть результат грамматического разбора уже обработанного текста. Лексема предложения служит для выбора применимого предложения в текущем состоянии анализатора. Следовательно, грамматическому разбору подлежит последняя часть предложения (область разбора), если, конечно, она присутствует в предложении.

Разнесенный грамматический разбор позволяет повысить эффективность контекстного анализа. Последнее необходимо ввиду возможности анализа текста, описываемого

неоднозначными грамматиками, которые, как известно, имеют несколько применимых предложений в одном и том же текущем состоянии [82].

Для учета неоднозначности в выборе предложений анализатор перед началом разбора каждого нового предложения сохраняет свое состояние (создает точку отката назад) и начинает грамматический разбор. В каждом новом состоянии анализатор производит поиск применимых предложений. Если таковых предложений не найдено, восстанавливается сохраненное ранее состояние (производится откат назад) и анализируется следующее предложение.

Описанная процедура реализуется рекурсивным вызовом анализатора всякий раз, когда начинается контекстное сопоставление предложений. Учитывая то, что понятийная модель призвана описать некоторую предметную область непротиворечивым образом, доля неоднозначности в описании предметной области не должна быть высока. Более того, неоднозначность понятийной модели становится выразительным средством контекстной технологии и используется для повышения уровня проблемного языка.

Как бы то ни было, при реализации грамматического разбора может быть предусмотрено некоторое критическое количество точек отката назад. При достижении последнего делается вывод о недостаточной проработке понятийной модели или об ошибке в тексте программы.

4.2.8. Специальные лексемы

При разнесенном грамматическом разборе используются две лексемы специального вида, инициирующие грамматический разбор «пустого» понятия empty и произвольного (заранее неизвестного, любого) понятия, которое на момент разбора включено в понятийную модель.

«Пустое» понятие. Грамматический разбор пустого понятия задается лексемой вида '' (две одинарные кавычки) и используется в том случае, когда из входного потока необходимо извлечь текст, не выражающий никакого понятия.. Использование лексемы пустого понятия связана с тем, что могут существовать фрагменты текста, которые нельзя сопоставить ни с одним понятием, а если и можно, то требуется не заносить это понятие на вершину стека контекста во время грамматического разбора.

«Любое» понятие. На практике встречаются ситуации, когда некоторая языковая конструкция может содержать в качестве одного из своих элементов заранее неизвестное понятие. Примером такой конструкции может служить следующее предложение на естественном языке «Обозначает ли фраза «X» понятие Y?».

В области языков программирования известен другой пример – оператор генерации исключения с переменной T произвольного типа в качестве параметра: 'throw' T. Для обра-

ботки таких исключений служит оператор: 'try' … 'catch' N1 … 'catch' N2 …, где N1, N1, … – имена типов переменных, переданных операторам throw.

В контекстной технологии грамматический разбор и извлечение из входного потока некоторого произвольного понятия задается лексемой "" (две двойные кавычки). Если в качестве алиаса лексемы "" задать некоторый идентификатор, то он принимает значение имени понятия, распознанного при грамматическом разборе.

4.2.9. Контекст

Ранее показано (см. 2.5.7 на с. 121), что существует семантически полная и непротиворечивая формальная теория понятий, выразительные возможности которой эквивалентны формализму контекстных грамматик. Покажем, что в рамках описываемой контекстной системы программирования возможно создание проблемных языков, описываемых контекстными грамматиками.

Грамматика является ***контекстной*** (контекстно-зависимой), если ее правила вывода имеют вид: $\alpha N\beta \to \alpha\omega\beta$, где α, ω, β – произвольные строки над терминальным и нетерминальным алфавитом, N – нетерминальный знак грамматики, а $\omega \neq e$, где e – пустой знак, $|e|=0$. Контекстные грамматики являются более мощным формализмом, чем контекстно-свободные, так как последние являются частным (вырожденным) случаем контекстных при $\alpha = e$ и $\beta = e$.

При дополнении множества правил контекстно-свободной грамматики расширяющими правилами вида $e \to \gamma$, где $|\gamma| \geq 0$, каждый нетерминальный знак N при выводе может интерпретироваться как имеющий слева и справа от себя пустой знак, $N \to eNe$. Это эквивалентно замене любого правила грамматики $N \to \omega$ на множество правил вида $\alpha N\beta \to \alpha\omega\beta$, где α и β – строки, непосредственно выводимые из пустого знака e, или сам знак e.

Однако в этом случае все нетерминальные знаки грамматики приобретают одно и то же множество левых и правых контекстов, равное множеству правил вывода пустого знака. Чтобы снять это ограничение определим выразительные средства для управления контекстами.

Если при выводе необходимо разрешить использование пустого знака слева или справа от нетерминального знака N, будем указывать пустой знак в виде квадратных скобок слева или справа от N, например, $[\alpha]N \to \omega$, $N[\beta] \to \omega$ или $[\alpha]N[\beta] \to \omega$, а в самих квадратных скобках задавать список правил, применимых для выражения пустого знака.

Так как сопровождать каждое правило контекстом не всегда удобно, в частности, это приводит к увеличению общего числа правил, контекст нетерминального знака будем указывать в месте его вхождения в правые части правил, например, $N \to \gamma[\alpha] N [\beta]\delta$.

Определение контекстов для понятий в предложениях понятийной модели осуществляется в текстах компиляции, заключаемом в квадратные скобки. Для доступа к строкам, которые извлечены из входного потока до квадратных скобок, или находятся во входном потоке после них, используется обращение к системе программирования, которая предоставляет такие строки через сервисы обратного вызова. В этом случае строки, извлеченные из входного потока, могут быть подвергнуты грамматическому разбору на предмет их соотнесения с тем или иным понятием или терминальной строкой.

В итоге, выразительные возможности проблемных языков становятся эквивалентными контекстным языкам. Последнее послужило основанием для названия описываемой технологии контекстной.

4.2.10. Иерархия языков

На рис. 4.6 показана иерархия выразительных возможностей языковых средств контекстной технологии.

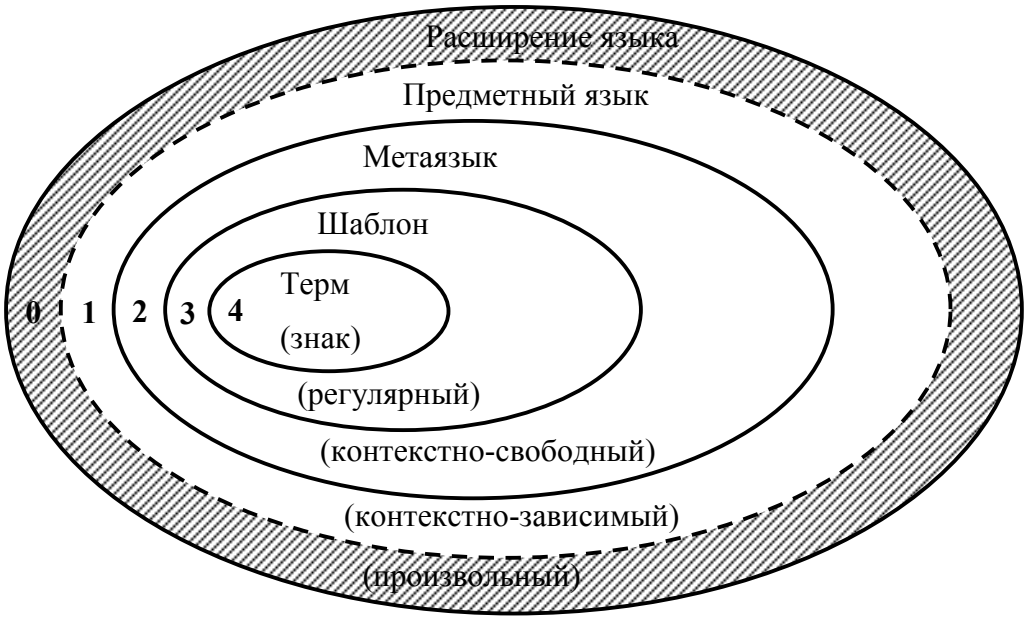

Рис. 4.6. Иерархия языков

На самом глубоком четвертом уровне располагается знаковое представление понятий, для чего используются термы. Следующий третий уровень представлен шаблонами, выразительная способность которых соответствует регулярным языкам по классификации Хомского [269]. На втором уровне находится протоязык, являющийся контекстно-

свободным. Самые высокие выразительные способности имеет проблемный язык, находящийся на первом уровне и соответствующий контекстным языкам.

Нулевой уровень, представленный языками произвольного вида, в контекстной технологии не имеет непосредственного применения, так как языки этого класса достаточно трудны для практической реализации, ибо порождают трудно преодолимые вычислительные трудности при грамматическом разборе.

В случае необходимости предметному языку можно придать выразительные возможности языков нулевого типа. Это может быть осуществлено путем принудительного вызова компилятора с передачей ему текста, сформированного при грамматическом разборе.

Если при разборе текста, выраженного некоторым предложением понятийно модели, породить другой текст, являющийся трансформацией анализируемого, то путем повторной его компиляции можно сформировать новый императив, который заменит или дополнит уже созданный. В итоге имеем преобразования анализируемого текста в новый текст, который заменяет исходный. Такого рода выразительные возможности присущи грамматикам нулевого типа.

В качестве иллюстрации трансформационного подхода может служит пример 3.33, где для выражения, полученного после дифференцирования функции, применена трансформация с целью его упрощения.

Таким образом, контекстная технология имеет развитые выразительные возможности, позволяющие, в конечном итоге, реализовать эффективный грамматический разбор и компиляцию текстов на предметных языках, относящихся к классу контекстно-зависимых языков по классификации Хомского. В случае необходимости выразительные возможности предметного языка могут быть повышены до языка, описываемого грамматикой с переписыванием термов. Однако такое расширение следует использовать с осторожностью, ибо оно приводит к снижению эффективности грамматического разбора.

4.3. Сравнительный анализ систем

Покажем, что система контекстного программирования покрывает известные подходы к грамматическому разбору текстов, сохраняя при этом выразительность и ясность понятийной модели. Для этого выполним сравнительный анализ разнесенного разбора и других подходов к грамматическому разбору текстов.

4.3.1. Грамматики конечных автоматов

Непосредственная реализация контекстно-зависимых и произвольных языков не получила своего распространения, что связывают с трудностями грамматического разбора [341, 156]:

– резко возрастает количество правил и нетерминальных знаков;

– размывается структура фраз языка, столь ясно представляемая контекстно-свободными грамматиками;

– теряются преимущества, связанные с разделением синтаксического и семантического компонентов языка.

Самым простым считается грамматический разбор, описываемый регулярной грамматикой. Однако этот формализм не применим для представления многих лингвистических феноменов, его оказалось достаточно только для описания морфологии [300].

В системе контекстного программирования регулярные грамматики используются в терминальных шаблонах и служат для выражения лексем в виде разрешимых множеств строк, что соответствует морфологическому описанию слов в лингвистике.

4.3.2. ATN-грамматики

Грамматики конечных автоматов достаточно эффективны в реализации, но обладают слишком ограниченными возможностями. По этой причине одним из широко используемых механизмов анализа текста является формализм расширенных сетей переходов (Augmented Transition Networks, ATN) [49].

Каждая расширенная сеть соответствует одному нетерминальному знаку грамматики предметного языка. Дуги в таких сетях помечены одним нетерминальным или терминальным и нетерминальным знаками. Все пути, ведущие от начального узла к конечному соответствуют некоторому правилу грамматики. В конечном счете каждая сеть реализует недетерминированную регулярную (конечно-автоматную) грамматику.

Формализм ATN расширяет грамматику конечных автоматов, например, вводя аппарат рекурсивного вызова другой сети (операция PUSH) и набор регистров, в которых хранятся текущие результаты разбора фразы, а также средства работы с этими регистрами. Значения регистров могут выступать условиями для переходов по веткам, что обеспечивает частичную зависимость от контекста и выход за пределы грамматики. Благодаря регистрам и операциям над значениями, которые там хранятся, ATN-формализм эквивалентен процедурному языку программирования.

Популярность ATN-формализма привела к реализации на его основе большого количества инструментальных систем [236]. В частности, система LINGOL [325] служит одним из известных примеров эффективной реализации базового механизма анализа. В сис-

теме определено три набора правил: грамматики, когнитивных функций и генеративных функций. Правила грамматики бинарные, т.е. могут содержать только два знака в своей правой части. Когнитивные функции описывают семантические ограничения на построение дерева разбора. Генеративные функции дают возможность связывать переменные среды языка программирования ЛИСП и использовать значения связанных переменных в других правилах.

В отличие от ATN-грамматик, в системе контекстного программирования с каждым терминальным или нетерминальным знаком может быть связан текст времени компиляции, выраженный на предметном языке и заключенный в квадратные скобки. Так как выразительность протоязыка и предметного языка эквивалентна контекстно-свободной и контекстно-зависимой грамматике соответственно, а сам предметный язык не ограничен каким-либо конкретным языком программирования, то совокупная выразительная возможность и гибкость предлагаемого формализма выше, чем у ATN-грамматик.

4.3.3. Категориальные грамматики

Теория категориальных грамматик занимается методами описания искусственных и естественных языков, связанными с типизацией языка. Каждый элемент словаря языка отнесен к одной или к нескольким категориям, так что каждая категория является либо базисной, либо определяемой комбинаторным путем через другие, более простые [306]. Считается, что анализ текста с использованием категориальных грамматик является альтернативным порождающему подходу, предложенного Хомским [269].

В категориальных грамматиках, являющихся по своим выразительным качествам эквивалентными контекстно-свободным грамматикам [83], все знания о синтаксической структуре хранятся в виде формул, описывающих синтаксические возможности отдельных слов или их типов. Определяется язык построения этих формул на основе базовых категорий: если A и B категории, то $A \circ B$ – тоже категория, где $\circ$ – одна из операций построения категорий.

Механизм применения категориальных формул – сокращение. Операция правого сокращения в построении категорий обозначается как /, а левого – \. Если α является выражением категории A/B и β – выражение категории B, тогда последовательность $\alpha\beta$ является выражением категории A, т.е. выражение категории A/B ожидает категорию B справа от себя. В нотации категориальных грамматик правила сокращения записываются так:

$$< \frac{A}{\alpha\beta}, \frac{A/B}{\alpha}, \frac{B}{\beta} >, \quad < \frac{A}{\beta\alpha}, \frac{B}{\beta}, \frac{B \setminus A}{\alpha} >.$$

Смысл нотации следующий: если имеется строка β, принадлежащая категории B, и строка α, принадлежащая категории A/B ($B\backslash A$), то конкатенация строк α и β (β и α) будет принадлежать к категории A.

Основные законы грамматической композиции имеют форму правила отделения Modus Ponens [317]:

$$A/B, B \mapsto A, \quad B, B\backslash A \mapsto A$$

или

$$\frac{\Gamma \mapsto A/B, \Delta \mapsto B}{\Gamma, \Delta \mapsto A}, \quad \frac{\Gamma \mapsto B, \Delta \mapsto B\backslash A}{\Gamma, \Delta \mapsto A},$$

где запись вида $\Gamma \mapsto A$ ($\Delta \mapsto B$) обозначает базовое заключение грамматики, выражающееся в том, что структурированная конфигурация лингвистических выражений Γ (Δ) может быть категоризована как правильное построенное выражение типа A (B).

Категоризация как процесс приписывания фрагменту текста некоторой категории может быть реализован в рамках системы контекстного программирования. Для этого предложения понятийной модели используются для выражения категориальных формул следующим образом. Для нотации A/B или $B\backslash A$ описываются понятия:

() A_B () … { } или () B_A () … { },

() A () A_B B { } или () A () B B_A { },

() B () … { },

где понятие A определено как выражаемое фрагментом текста, который состоит из двух частей: первая часть соответствует понятию A_B или B, а вторая – B или B_A. После такого определения нотаций при разнесенном грамматическом разборе анализируемый текст будет сопоставлен описанным категориям.

4.3.4. Расширения контекстно-свободных грамматик

Хотя синтаксис контекстно-свободных правил очень прост, для описания многих лингвистических феноменов его оказывается недостаточно. В частности, контекстно-свободными правилами неудобно описывать согласование, например, в лице и числе между подлежащим и сказуемым [248].

Альтернативой для контекстно-зависимых грамматик является сохранение простой структуры правил контекстно-свободной грамматики и расширения этих правил за счет добавления процедур, выполняющих необходимые контекстные проверки. Такие процедуры выполняются в процессе анализа текста. Поэтому для описания лингвистических феноменов, в основном, применяются контекстно-свободные грамматики, но с некоторыми расширениями.

Например, вместо использования грамматики, выражающей формы единственного и множественного числа, времена глаголов, одушевленные и неодушевленные предметы и т.д., их можно представить с помощью средств, связанных с терминальными и нетерминальными знаками контекстно-свободной грамматики. Связанные с правилами грамматики процедуры обращаются к этим средствам для присвоения значений атрибутам нетерминальных знаков и выполнения необходимых проверок.

К грамматикам, использующим расширения контекстно-свободных грамматик для реализации контекстной зависимости, относятся:

– расширенные грамматики с фазовой структурой [295];

– расширения логических грамматик [261];

– расширения ATN-грамматик [367].

Например, при автоматизированного построения синтаксических анализаторов в среде ANTLR (ANother Tool for Language Recognition) [320] реализована генерация текста анализатора на одном из целевых языков (Java, C++, C#) на основе LL(*)-грамматики, причем имеется возможность в каждом правиле грамматики задавать на используемом целевом языке действия, которые будут выполняться в процессе разбора правила.

В контекстной технологии контекстно-свободная грамматика используется только для описания протоязыка, причем выражена она в наиболее простой детерминированной форме, допускающей грамматический разбор методом рекурсивного спуска (см. рис. 3.14). Последнее необходимо для повышения эффективности грамматического разбора. Однако для обеспечения возможности расширения протоязыка путем задания контекстных условий, предусмотрена специальная конструкция (текст на предметном языке, заключенный в угловые скобки), которая подлежит грамматическому разбору, компиляции и выполнению при анализе соответствующего предложения понятийной модели. Последнее, в случае необходимости, позволяет развить выразительные возможности протоязыка вплоть до учета контекстной зависимости его правил.

4.3.5. Трансформационные грамматики

Наиболее мощными из известных являются системы, построенные на теории трансформационных грамматик [233, 270]. Изначально эта теория появилась для решения проблем структурной интерпретации естественных языков, которые не выражаются с помощью контекстно-свободных правил.

В трансформационных грамматиках в дополнение к контекстно-свободным вводятся контекстно-зависимые правила, или правила трансформаций, предназначенные для преобразования синтаксической структуры фраз к контекстно-свободному эквиваленту [235]. Хотя применение трансформаций относится к небольшому числу лингвистических

феноменов, грамматический анализ текстов, описываемых трансформационными грамматиками, достаточно сложен, ибо недетерминирован. Особая сложность грамматического разбора наблюдаются для естественных языков с относительно свободным порядком слов.

Известная система построения и анализа грамматик LIFER [299], так же как и LINGOL [325], обладает возможностью использовать ЛИСП-функции в правых частях правил. В частности, в приложениях, разработанных с помощью данной системы, ЛИСП-функции используется для выражения семантического представления фразы как запроса к базе данных.

С точки зрения авторов работы [289], формализм трансформационных грамматик оказывается чересчур мощным, ибо эквивалентен грамматике нулевого типа и потому малоэффективным. Кроме того, вследствие своей мощности он когнитивно недостоверен, поскольку для языка, описанного грамматикой нулевого типа, невозможно за конечное время доказать допустимость произвольного предложения.

В системе контекстного программирования возможности, реализованные в трансформационных грамматиках, представляются как с помощью определения семантики предложений понятийной модели текстом на предметном языке (текст в правых частях правил, заключенный в фигурные скобки), так и путем грамматического разбора трансформированного текста, осуществляемого при вызове соответствующего сервиса системы программирования. Описание сервисов системы контекстного программирования дано в подразделе 4.6.

4.3.6. Сопоставление с образцом

К системам, основанным на сопоставлении с образцом, относятся системы, определяющие некоторый язык задания правил, причем правила имеют строго определенную структуру. Ключевым в этой структуре является образец, который ищется по тексту или по множеству уже построенных над текстом объектов, и действия, выполняемые в случае успешного поиска образца. В действиях могут конструировать новые объекты или модифицироваться уже существующие, а также выполняться другие действия.

По своей сути, сопоставление с образцом заключается в покрытии текста некоторым набором шаблонов. Шаблон является образцом, узлами которого являются слова или другие шаблоны, а также переменные части, которые при наложении на текст заполняются его материалом.

Развитый механизм сопоставления с образцом, основанный на средствах языка ПЛЭНЕР [186], реализован в системе TULIPS [160], где элементы словарного представления (словарные статьи и правила) имеют несколько уровней описания: морфологический, синтаксический, лексико-семантический и прагматический. После проведения синтакси-

ческого анализа фразы на основе построенного дерева грамматического разбора, производится операция прагматической интерпретации с помощью заполнения фрейма задачи.

Современным примером реализации анализа текста, основанного на настраиваемых лексических шаблонах, является система Alex [209], где в качестве шаблона используется выражение, аналогичное регулярному, но с многими дополнительными особенностями, в частности, вложенными шаблонами, контекстом, словоизменением, и т.д. Шаблоны образуют некоторую систему классов, определяемых в терминах объектно-ориентированного подхода. При анализе текста выделяются фрагменты, покрываемые шаблонами, и образуются лексические объекты тех классов, которым принадлежат шаблоны. Объект имеет начальную и конечную позицию в тексте. При создании лексического объекта значения его свойств могут принимать некоторые значения, которые могут зависеть от параметров покрываемого фрагмента текста и свойств вложенных шаблонов. Результатом анализа является множество лексических объектов. В состав шаблонов могут входить не только условия создания лексических объектов, но и действия, выполняемые при создании этих объектов – методы. Среди методов – занесение в свойства объекта сопоставленных фрагменту шаблона значений из анализируемого текста.

К недостаткам систем, основанных на сопоставлении с образцом, следует отнести их сугубо синтаксическую направленность и, как следствие этого, слабые выразительные средства для представления семантических аспектов анализа текста.

В системе контекстного программирования возможности сопоставления с образцом реализуются путем использования терминальных шаблонов, задающих морфологические правила (образцы) для выделения лексем из текста. Лексемы, в свою очередь, являются структурными частями предложений понятийной модели, выражающими синтаксические шаблоны для допустимых фраз. В свою очередь семантическое описание предложений осуществляется на предметном языке, а прагматический уровень представляется множеством именованных семантических описаний, или аспектов.

В описываемой системе результатом грамматического разбора фразы может является не только пассивное заполнение некоторой структуры данных (фрейма, объекта), но и выполнение некоторых активных действий, связанных, например, с порождением или уничтожением сущностей, принадлежащих любым описанным в модели понятиям, т.е. возможно накопление фактов, которое осуществляется в терминах моделируемой предметной области.

4.3.7. Унификационные формализмы

Стремление к построению формализмов, которые обладали бы нужной степенью декларативности (в отличие от ATN), с одной стороны, и использование языков програм-

мирования на базе Пролога, с другой стороны, привели к появлению грамматик, основанных на логическом исчислении в форме атрибут-значение [301].

Синтаксис описания контекстно-свободных правил в этом формализме расширен введением атрибутов. Это означает, что нетерминальные символы могут иметь некоторую внутреннюю структуру. Иногда в качестве дополнения структуру, аналогичную структуре нетерминальных знаков, приписывают и правилам грамматики [247].

Многие унификационные формализмы в различной степени дополнены механизмом логического программирования, основанного на ограничениях [181]. Структурой данных, используемой в этих языках, является набор атрибутов, значения которых, в свою очередь, также может быть набором атрибутов. Благодаря этому описание грамматики может включать унификационные операции, ведущие к вызову механизма логического вывода и некоторых других действий. Например, унификационный механизм может включать определение переменных, представление отрицания и полной дизъюнкции, и т.п. Эти описания компилируются в текст на языке Пролог, включающий базовые операциям типизированной атрибутно-значной логики.

Использование унификационного формализма возможно и в контекстной технологии. В этом случае для связи различных частей предложения понятийной модели применяется атрибутная память. С каждым терминальным и нетерминальным знаком, а также с предложением в целом связывается атрибутная память, имеющая линейную организацию. Это позволяет текст, выражающий сущность некоторого понятия, сопроводить множеством атрибутов, вычисляемых и изменяемых по мере необходимости.

Если в тексте, выраженном некоторым предложением, встретится фраза, выражающая некоторое понятие в форме, определяемой другим предложением, то возможно получение значения атрибутов сущности этого понятия, которые сформированных при грамматическом разборе последнего предложения. Это позволяет извлекать атрибуты элементов предложения и формировать атрибуты понятия, выражаемого этим предложением.

Заметим, что система контекстного программирования не имеет специализированных языковых средств для доступа к атрибутной памяти. Доступ к атрибутной памяти реализован через вызовы сервисов системы программирования (см. параграф 4.6.1). В итоге, оперирование атрибутной памятью осуществляется на предметном языке в текстах компиляции, которые могут появляться после произвольного элемента предложения – для определения его атрибутов, и после самого предложения – для вычисления атрибутов понятия, выраженного этим предложением.

4.3.8. Логические грамматики

Логическими называются грамматики, имеющие возможность логической интерпретации правил. Первые логические грамматики были использованы В.Б. Борщёвым и М.А. Хомяковым [25] (1973 г.), и независимо от них А. Кольмероэ [274] (1975 г.) – разработчик языка Пролог. Однако ближайшими предшественниками этого класса были атрибутные грамматики Д. Кнута [303] (1968 г.) и грамматики А. Ван Вейнгаардена, использованные для определения языка Алгол-68 [182] (1969 г.).

Одним из известных типов логических грамматик являются грамматики DCG (Definite Clause Grammars), разработанные Ф. Перейрой и Д. Уорреном [321] (1980 г.), формализм которых неизменно включается во все развитые системы логического программирования. Эти грамматики, в отличие от атрибутных, не требуют задания порядка вычисления атрибутов, т.к. построение семантической структуры текста с помощью DCG описывается в процедурном виде, например, на языке Пролог [290].

Выразим логическую грамматику обобщенными правилами замены вида: $a \to b$, где a и b – конечные строки терминальных и нетерминальных знаков. Строка a имеет некоторую структуру и представляется как $a(x_1 x_2 \ldots)$, где x_i – глобальная переменная или структура a_i такого же вида. Строка b состоит из элементов, каждый из которых может быть структурой a_i, терминальной строкой в квадратных скобках $[t_1 t_2 \ldots]$, а также списком процедур p_j, заключенным в фигурные скобки $\{p_1 p_2 \ldots\}$. Процедуры используются для выражения дополнительных условий, которые в должны выполнятся при применении правила.

Правило замены $a \to b$ интерпретируется как импликация $a :\!- b$ (a верно, если b верно). Структура $a(x_1 x_2 \ldots)$ описывает слово или часть слова языка и определяется правой частью b: первый аргумент x_1 является частью слова, выраженной первым элементом b, второй аргумент x_2 – следующим элементом, и т.д.

Запрос к системе на распознавание текста имеет вид $s(x, a)$, где s – аксиома грамматики – некоторая ранее определенная структура, первый аргумент x – переменная, принимающая значение распознанной части текста, второй аргумент a – структура части текста, оставшегося нераспознанной. Если a выразить пустым термом $[]$, то произойдет распознавание текста, порождаемого логической грамматикой.

Система контекстного программирования позволяет выполнить грамматический разбор текста, используемый для логических грамматик. Каждое правило понятийной модели является продукцией контекстно-свободной грамматики, которая дополнена описа-

ниями процесса грамматического разбора (текст в угловых скобках), компиляции (текст в квадратных скобках) и исполнения (текст в фигурных скобках). Предложения, состоящее из последовательности понятий (нетерминальных знаков) и лексем (перечислимых множеств строк), определяет структуру выражения понятия в тексте и соответствует структуре в терминологии предикатных грамматик. Список процедур может быть выражен в виде текста компиляции, в котором описывается проверка дополнительных (контекстных) условий, необходимых для применения предложения. Результатом распознавания предложения является выполнение всех текстов компиляции и генерация кода вызова императива времени исполнения. Как в процессе компиляции, так и при исполнении императива возможно создание произвольных сущностей и добавление их систему, в том числе и присваивание атрибутам понятия выражающих их строк.

Однако в отличие от логических грамматик, в контекстной технологии поддерживается встроенные в формализм обобщение и ассоциация нетерминальных понятий, что выражается в использовании понятийной структуры предметной области при разнесенном грамматическом разборе. При этом происходит отождествление нетерминальных знаков грамматики с учетом их дифференциации, а также каждый такой знак рассматривается как состоящий из одного или нескольких понятий, задаваемых их интеграцией.

В последнем случае имеет существенное отличие нетерминального знака контекстной технологии от структуры логической грамматики, где структура нетерминального знака задает только его форму выражения в тексте, в то время как в контекстной технологии форма выражения нетерминального понятия определяется последовательностью элементов (понятий и лексем) одного или нескольких предложений понятийной модели и не зависит от его внутренней структуры. Это позволяет добиться больших выразительных возможностей при описании языков и выполнить это описание в более свободной (естественной) форме.

4.3.9. Программирование в ограничениях

Известен подход к грамматическому разбору, основанный не на описании правил вывода порождающей грамматики, а на задании множества ограничений на текст или фрагменты текста (licensing rules), которые в явном виде друг с другом не связаны.

Математические методы решения задач с ограничениями описаны в работе [333]. Представители этого направления связывают популярность таких грамматик с тем, что правила контекстно-свободных и контекстно-зависимых грамматик описывают структурные свойства лингвистических конструкций, в то время как ограничения задают более общие условия, определяющие эти конструкции. В частности, это приводит к большей не-

зависимости правил и возможности описания в грамматике глубинных свойств лексических единиц.

Аналогичный подход используется и в области программирования, где для решения плохо структурированных задач применяются методы программирования в ограничениях (constraint programming) [201].

Программирование в ограничениях является максимально декларативным и основано на описании модели задачи, а не алгоритма ее решения. Модель специфицируется в виде неупорядоченной совокупности отношений, которые соответствуют связям, существующим между параметрами задачи. Эти отношения могут иметь вид уравнений, неравенств, логических выражений и т.п.

В связи с тем, что в системе контекстного программирования отсутствуют начальные ограничения на формы выражения конструкций проблемного языка, то не видится никаких препятствий в реализации и чисто декларативного подхода к программированию, основанного на описании ограничений. В качестве примера можно указать реализацию языка логического программирования (см. Приложение 2), основанного на исчислении предикатов первого порядка. В этом примере показано, как с помощью методологии понятийного анализа осуществляется постановка задачи, а средствами контекстной технологии осуществляется реализация системы программирования, являющейся выражением некоторого специфического подхода к организации и обработке данных, описываемого исчислением предикатов первого порядка.

4.3.10. Функциональные грамматики

В подходах к грамматическому разбору, рассмотренных выше, основное внимание обращается на правила описания синтаксической структуры текста. В противовес этому направлению функциональный подход рассматривает синтаксис языка лишь как средство реализации коммуникативных целей. Поэтому при функциональном подходе уделяет большее внимание семантике, а синтаксические категории описываются с точки зрения их функций для выражения некоторого значения, или смысла.

При построении функциональной грамматики основную роль играет направление от семантики – от содержания к средствам выражения. Процесс грамматического разбора основан на взаимодействия слов в предложении, в результате которого происходит их связывание друг с другом, выбор текущей альтернативы связывания, вычисление обобщенного семантического типа связи и сборка предложения в единую конструкцию, полностью определяется наборами участвующих в этом процессе семантико-грамматических типов 294.

Очевидно, без привлечения в том или ином виде обобщенных семантических знаний о языке правильный синтаксический анализ невозможен. Для этого используется развитая иерархическая классификация семантических категорий, представленных лексемами, выражающая некоторую модель предметной области (окружающей среды), высказывания о которой предполагается анализировать.

Например, в одной из последних работ [220], посвященных функциональному подходу, анализ текста разбивается на три этапа: анализ отдельного слова, анализ предложения и анализ текста в целом. На первом этапе слова предложения преобразуются в набор термов, готовых к взаимодействию друг с другом. На втором этапе осуществляется их взаимодействие, в процессе которого происходит выбор лексемы из словаря и приписывание ее терму. Если по какой-либо причине связывание лексем не осуществилось, процесс повторяется. Завершается второй этап связыванием выбранных лексем в единую структуру, описывающую семантику анализируемого предложения. Аналогично представляется третий этап, на котором в качестве связываемых выступают структуры распознанных предложений.

Синтаксический анализатор в рассматриваемой реализации решает две основные задачи: правильную привязку терма лексеме и связывание выбранных лексем в единую структуру, с учетом тех ограничений, которые заданных для каждой лексемы в словаре. Единственной действующей операцией является операция связывания лексем – при анализе предложения, или связывание распознанных структур – при анализе текста в целом.

Подход, обозначенный функциональными грамматиками, может быть реализован в системе контекстного программирования, в процессе приписывания терму некоторого лексического значения, которое осуществляется с учетом текущего контекста, и в процессе структурного распознавания предложения. В этом случае в качестве иерархической классификации семантических категорий выступает понятийная модель предметной области, а текущий контекст определяет условия, при котором такое приписывание становится возможным. Итоговое связывание лексем в единую структуру осуществляется на этапе структурного распознавания предложения, по завершения которого имеем распознанное предложение, семантика которого описана и представлена одним или несколькими императивами. Распознанное предложение, в отличие от функционального подхода, активно, что выражается в исполнении императива, определяемого текущей прагматикой.

Связывание распознанных предложений можно представить как побочный эффект исполнения соответствующих императивов, где для семантического контроля текста предусмотрена реализация некоторой концепции связности, возможно зависящая от рассматриваемой предметной области.

4.3.11. Система контекстного программирования

Подытожим результаты сравнительного анализа системы контекстного программирования с другими известными подходами к организации обработки данных. В отличие от описанных выше технологий грамматического разбора текста и описания его семантики, система контекстного программирования использует понятийную структуру предметной области и механизм разнесенного грамматического разбора. Благодаря этому принципиальными отличиями контекстной технологии от рассмотренных аналогов является:

– контекстная интерпретация фрагментов текста;

– обобщение и ассоциация нетерминальных знаков;

– описание семантики на предметном языке;

– расширяемость протоязыка и предметного языка;

– множество целевых платформ и пополняемость набора базовых примитивов;

– возможность реализации всех анализируемых технологий обработки текстов.

Существенное отличие системы контекстного программирования заключается в возможности определения семантики предметного языка на самом предметном языке и использование для этого открытого множества базовых семантических категорий.

4.4. Архитектура системы программирования

Рассмотрим архитектурные особенности реализации системы контекстного программирования, которые следуют из методологии понятийного анализа и определяются разработанной контекстной технологией обработки данных.

4.4.1. Компиляция императивов

Семантика предложений понятийной модели определяется их описаниями и задается в виде императивов – именованных последовательностей действий, выраженных текстом text на определяемом языке (рис. 4.7).

```
sentence    →  [aspect] syntax [parse [compile]] semantic
semantic    →  pragmatic
               pragmatic semantic
pragmatic   →  [aspect] '{' [text] '}'
```

Рис. 4.7. Определение семантики предложений

Если при определении императива некоторого предложения будут распознаны другие предложения, то необходимо некоторым образом организовать выполнение этих императивов при вызове императива определяемого предложения.

Пример 4.4. Семантика предложения Boolean 'imp' Boolean → Boolean из примера «Исчисление высказываний» предыдущей главы может быть описана низкоуровневыми средствами на языке ассемблера процессора Intel [243]:

```
Boolean 'imp' Boolean
    [ asm{ pop eax; not eax; pop edx; or eax, edx; push eax } ]
```

Однако ранее семантика этого предложения определена средствами проблемного языка, который к тому времени уже был достаточен для такого определения:

```
Boolean `a` 'imp' Boolean `b`
    { not a or b }
```

Следовательно, при построении императива этого предложения необходимо предусмотреть вызов императивов тех предложений, которые распознаны при грамматическом разборе 'not a or b'. Предположим, имеются императивы, описывающие семантику этих предложений:

```
'not' Boolean { … }                 #\ Логическое отрацание
Boolean 'or' Boolean { … }          #\ Дизъюнкция
```

Представив императив рассматриваемого предложения в виде вызовов распознанных предложений, условно обозначенных как (not) и (or), и в порядке их распознавания, имеем:

```
Boolean `a` 'imp' Boolean `b`       #\ Импликация
    { a (not) b (or) }
```

Открывающаяся фигурная скобка извлекает из стека две сущности Boolean и присваивает им идентификаторы a и b в порядке их перечисления в предложении. Первое распознанное предложение 'not' Boolean → Boolean обрабатывает операнд a, который перед вызовом его императива заносится в стек, а в процессе его исполнения операнд извлекается из стека, инвертируется и полученный результат опять заносится в стек.

Перед вызовом второго предложения Boolean 'or' Boolean → Boolean операнд b записывается на вершину стека, где к этому времени уже находился результат, созданный предложением 'not' Boolean → Boolean. После завершения императива второго предложения, который извлекает два операнда с вершины стека и выполняет операции дизъюнкции, на вершине стека имеем искомый результат: not a or b.

Как и следовало ожидать, первое и второе определения семантики предложения Disjunction 'imp' Disjunction оказались эквивалентными. ♦

Таким образом, во время грамматического разбора текста семантических определений предложений необходимо создавать их компилированное представление, состоящее

из последовательности вызовов императивов тех предложений, которые распознаны при грамматическом разборе и в порядке распознания. Сам императив, в свою очередь, можно трактовать как некоторую единицу вызова, требующую для своего выполнения как передачи параметров, так и возврата результата.

Формальными параметрами императива являются понятия, которые встречаются при описании синтаксиса предложения, а фактическими параметрами – сущности этих понятий. Так как автоматом, эквивалентным по своим порождающим и распознающим возможностям контекстно-свободной грамматике, является магазинный (стековый) автомат [205], то для организации выполнения императивов целесообразно использовать целевую платформу со стековой организацией вычислений.

4.4.2. Прямая подстановка

Как видно из предыдущего примера, вызов императивов сопровождается достаточно большими накладными расходами. Для повышения эффективности генерируемого кода вместо вызова императива используется прямая подстановка кода, составляющего тело небольших определений синтаксиса.

Для задания прямой подстановки применяется текст компиляции compile, задаваемый при определении синтаксиса предложений и заключаемый в квадратные скобки (рис. 4.8).

```
sentence    →  [aspect] syntax semantic
syntax      →  item [parse] [compile]
               item [parse] [compile] syntax
item        →  notion [alias]
               lexeme [alias]
compile     →  [aspect] '[' [text] ']'
```

Рис. 4.8. Текст компиляции

Очевидно, текст компиляции подлежит такому же грамматическому разбору, как и описание семантики предложений. Генерация кода прямой подстановки должна задаваться этим текстом и заключаться в тело императива, реализующего описанные действия.

Пример 4.5. Переопределим предложения примера 4.4:

```
'not' Boolean
        [ asm{ pop eax; not eax; push eax } ] {}
Boolean 'or' Boolean
        [ asm{ pop eax; pop edx; or eax, edx; push eax } ] {}
Boolean `a` 'imp' `b` Boolean
       { not a or b }
```

где аспект asm определен для записи ассемблерных команд непосредственно в область кода формируемой единицы вызова, а пустые фигурные скобки используются как разделить и символизируют об отсутствии явного описания семантики у этого предложения.

Тогда для императива последнего предложения будет сгенерирован код, эквивалентный

```
asm{ … push eax; not eax; push eax; …
     pop eax; pop edx; or eax, edx; push eax … }
```

где вместо многоточия записываются команды, обеспечивающие взаимную стыковку команд прямой подстановки. ♦

4.4.3. Процедурный вызов

Стек синтаксического анализатора (стек контекста) хранит идентификаторы понятий. В свою очередь, стек времени исполнения (стек операндов) оперирует сущностями, соответствующими этим понятиям (рис. 4.9).

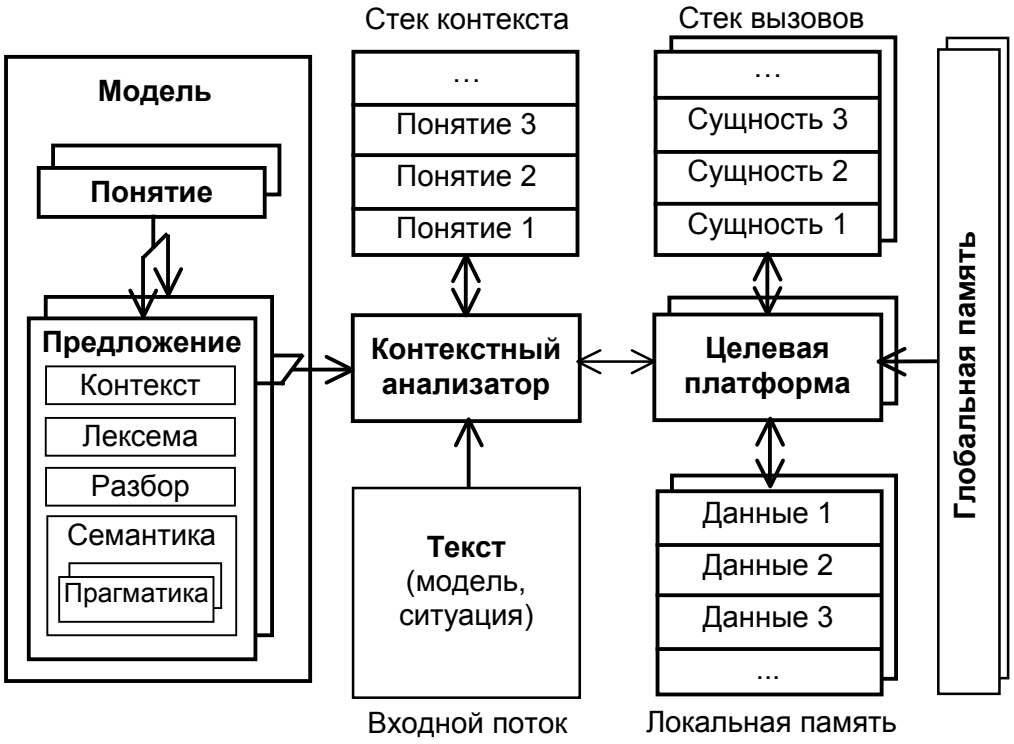

Рис. 4.9. Архитектура системы контекстного программирования

Следовательно, при выполнении любого императива, сущности, описанные в виде понятий из синтаксиса предложения, должны располагаться в стеке операндов в том же порядке, что и идентификаторы понятий в стеке синтаксического анализатора. Последнее позволяет рассматривать синтаксис предложения как описание структуры процедурного вызовов императива, а также как соглашение по передаче параметров и возврату резуль-

тата. Напомним, что синтаксис предложения определяет правила выражения соответствующего понятия в тексте программы. Следовательно, результатом выполнения любого императива является сущность понятия – результата предложения.

Как показано в 4.4.1, прологом процедурного вызова является сохранение адреса возврата из процедуры в стеке адресов возврата, извлечение из стека операндов сущностей–аргументов и их сохранение в локальной памяти процедуры. Порядок и количество аргументов определяется синтаксисом предложения. Для обеспечения реентерабельности [216] императивов локальная память для процедур выделяется в стеке адресов возврата (рис. 4.9).

Доступ к сущностям-аргументам из тела процедуры осуществляется путем прямого обращения к ячейкам локальной памяти. В случае необходимости локальная память может использоваться для создания других объектов локальной видимости. Объекты с глобальной видимостью создаются в глобальной памяти на основе выделения и освобождения участков памяти требуемого объема, для чего используются специальные команды виртуальной машины.

Эпилог процедурного вызова состоит из сохранения сущности-результата в стеке операндов, освобождения локальной памяти и возврата из процедуры по адресу, извлекаемому из стека адресов возврата и сохраненному там при вызове.

4.4.4. Аксиома

Текст в квадратных скобках интерпретируется как действия, которые выполняются в процессе распознавания предложения или после его распознавания (рис. 4.8). Однако этот текст выражается на языке, определяемом понятийной моделью.

Однако в самом начале, когда понятийная модель не содержит ни одного предложения, как определить семантику первого предложения? Очевидно, для этого необходимо предусмотреть специальные средства и связанные с этими средствами соглашения. Такие средства и соглашения называются ***аксиомой*** и являются по своей сути предопределенными (изначально подразумевающимися).

Определим предложение, которое будем использовать для записи в область кода какого-либо числа, например байта:

```
() ()
        '#' "[0-9A-F][0-9A-F]" {}
```

где первая строка задает «пустое» понятие, которое не является ни обобщением, ни агрегацией других понятий (или обобщает и агрегирует само себя), а вторая строка определяет предложение, выражающее «пустое» понятие (или не выражающее никакого понятия) и состоящее из двух лексем: терма '#' и шаблона "[0-9A-F][0-9A-F]".

Терм обеспечивает однозначное опознание предложения в тексте, а шаблон гарантирует наличие после знака # двух шестнадцатеричных цифр. В итоге имеем декларацию предложения, позволяющего выразить в тексте значение одного байта, закодированного в шестнадцатеричной системе счисления. Так как это предложение первое, то не существует средств для задания его семантики. А значит, не имеется возможности и для задания семантики любых других предложений понятийной модели, сколько бы дополнительных предложений мы ни декларировали в программе.

Для разрешения описанной проблемы будем использовать следующее минимально необходимое соглашение (аксиому): ***«пустые» квадратные скобки реализуют запись в область кода того значения, которое задается элементом предложения, после которого эти скобки указаны***.

С учетом сформулированной аксиомы переопределим первое предложение так:

```
'#' "[0-9A-F][0-9A-F]" [] {}
```

Теперь, после такого определения и с учетом аксиомы, получаем предложение с семантикой: «запись байта в область кода». Этого предложения уже достаточно для задания семантики всех других предложений понятийной модели.

Пример 4.6. Определим семантику предложений из примера 4.5.

```
'not' Boolen
        [ #58 #F7 #D0 #50 ] {}
Boolen 'or' Boolen
        [ #58 #5A #0B #C2 #50 ] {}
Boolen `a` 'imp' `b` Boolen
       { not a or b }
```

где, например, текст '#58 #F7 #D0 #50' эквивалентен ранее используемому тексту прямой подстановки 'code{ pop eax; not eax; push eax }'. ♦

Очевидно, запись в область кода команды процессора является частной интерпретацией аксиомы. Для генерации некоторого промежуточного (текстового) представления возможно, например, такое определение:

```
'`' "[0-9A-Za-z\s.,]+" [] '`' {}
```

которое реализует запись мнемонического обозначения команды виртуальной машины или строки целевого языка программирования. В рассматриваемом случае текст, подлежащий записи, заключается в обратные одинарные кавычки и может содержать пробелы (задается метасимволом \s) и знаки пунктуации (точка и запятая).

Пример 4.7. Определим семантику еще одним способом – текстом на целевом языке программирования. Для этого рассмотрим самый простой случай – использование язы-

ка ассемблера. Если аксиома системы определена как показано выше, то семантику предложений из примера 4.5 можно определить так:

```
'not' Boolen
        [ `pop eax` `not eax` `push eax` ] {}
Boolen 'or' Boolen
        [ `pop eax` `pop edx` `or eax, edx` `push eax` ] {}
Boolen `a` 'imp' `b` Boolen
       { not a or b } ♦
```

Более подробно особенности обработки и интерпретации императивов будут рассмотрены в 4.5.2. Заметим, что соглашение об аксиоме является единственным соглашением, необходимым и достаточным для реализации модели произвольной сложности в рамках описанного формализма.

4.4.5. Фазы

В процессе своего функционирования система контекстного программирования может находиться в одной из трех фаз (рис. 4.10):

– в фазе разбора;

– в фазе компиляции;

– в фазе выполнения.

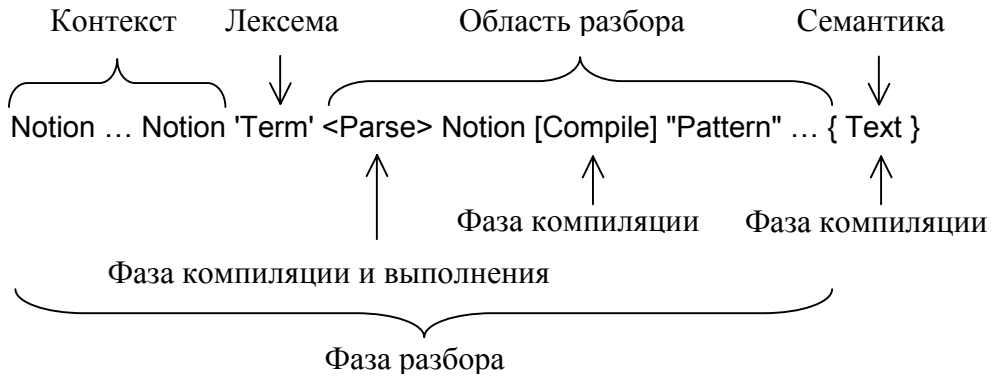

Рис. 4.10. Фазы работы системы программирования

В ***фазе разбора*** выполняется грамматический разбор текста (лексический, синтаксический и семантический анализ), заданного на протоязыке контекстной технологии.

Если в при разборе в структуре анализируемого предложения обнаружится текст разбора parse, подлежащий компиляции и непосредственному исполнению в этой фазе, система сначала компилирует этот текст, для чего переходит в фазу компиляции, а затем выполняет порожденный компилятором код, переходя в фазу исполнения. Текст разбора служит для расширения возможностей протоязыка при грамматическом разборе предложений.

Если в структуре предложения встретится текст компиляции compile, то этот текст компилируется, а порожденный им код сохраняется в структуре анализируемого предложения для исполнения в фазе компиляции.

В ***фазе компиляции*** происходит грамматический разбор текста, заданного на определенном в понятийной модели проблемном языке, осуществляемый путем контекстного сопоставления и структурного распознавания предложения, находящегося в текущей позиции входного потока. При этом используется внутреннее представление предложений, ранее обработанных и сохраненных в понятийной модели.

Если в структуре распознаваемого предложения встретится откомпилированный в фазе разбора текст компиляции compile, то код, порожденный этим текстом, выполняется, для чего система переходит в фазу исполнения. Так как на действия, описываемые текстом компиляции compile, не накладывается никаких ограничений, то побочным эффектом его исполнения может быть, в том числе, и прямая генерация кода. Другой частный результат исполнения кода компиляции – принудительное прекращение структурного распознавания предложения, что может быть следствием проверки более сложных контекстных условий, чем те, которые задаются средствами протоязыка, или, что тоже самое, реализуется проверка контекстных условий определяемого проблемного языка.

В итоге получаем следующий важный вывод: в процессе определения проблемного языка и одновременно с ним создается ***специализированный компилятор*** для этого языка.

В ***фазе выполнения*** исполняется код, порожденный описанием решения прикладной задачи (ситуационная часть situation) или код, порожденный описанием семантики предложений (императивы конструкции semantic). Код ситуационной части выполняется непосредственно после компиляции, а код императивов – в местах его вызова.

Таким образом, текст, заключенный в квадратные скобки, следует рассматривать как подлежащий компиляции во время грамматического разбора предложения и подлежащий исполнению – во время его компиляции. Отсюда и произошло его название – текст компиляции. Текст в фигурных скобках подлежит компиляции во время грамматического разбора предложения, а соответствующий ему код исполняется во время вызова императива, когда исполняется код, при порождении которого было распознано предложение, содержащее этот императив. Текст в угловых скобках может быть текстом разбора или текстом ситуационной части. Как текст разбора, так и текст ситуации исполняются непосредственно после свой компиляции. Поэтому и скобки, в которые заключаются эти тексты, выбраны одинаковыми.

4.5. Организация обработки данных

Структурно система контекстного программирования может быть представлена состоящей из следующих частей (рис. 4.11):

– входной поток Stream;

– лексический анализатор Token с препроцессором Text и словарем макроопределений Vocabulary;

– синтаксический анализатор Parser;

– семантический анализатор Analyzer;

– виртуальные машины Engine;

– понятийная модель Cognition.

4.5.1. Входной поток

Входной поток Stream организован в виде файлов, содержащих текст обрабатываемой программы. Результатом работы системы контекстного программирования является понятийная модель предметной области, представленная в компилированном виде. Если во входном файле содержится текст ситуационной части, то после его компиляции в код целевой платформы происходит выполнение этого кода.

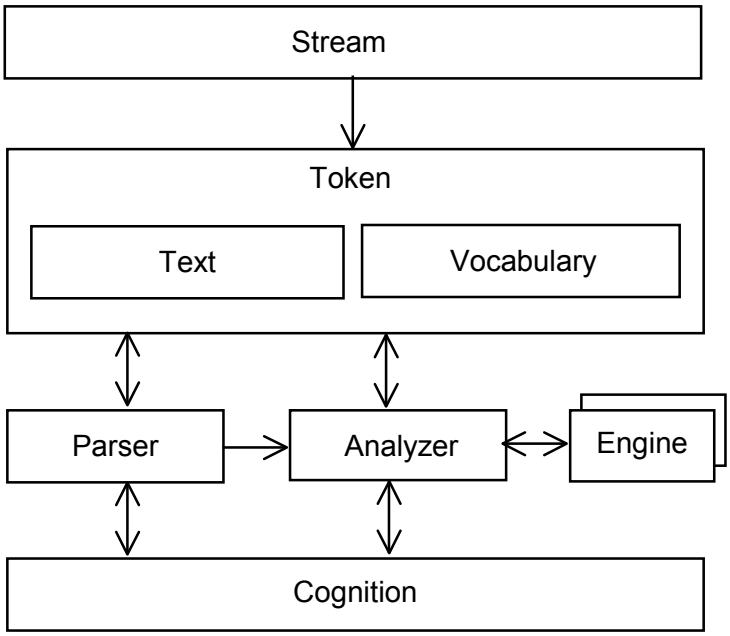

Рис. 4.11. Структура системы контекстного программирования

4.5.2. Виртуальная машина

Для исполнения кода, генерируемого семантическим анализатором, используется одна или несколько целевых вычислительных платформ. Целевая вычислительная платформа представляется в виде некоторого интерфейса, декларируемого средствами протоя-

зыка. Со стороны системы программирования целевая платформа представляется в виде одной или нескольких виртуальных машин, исполняющих некоторый набор команд (операторов).

На вход виртуальной машины подается код, порожденный семантическим анализатором, организованный в виде последовательности байтов. Этот код является ***единицей вызова*** виртуальной машины и соответствует императиву некоторого предложения понятийной модели. В зависимости от типа целевой платформы каждая единица вызова может представляться:

– последовательностью команд непосредственного исполнения;

– последовательностью интерпретируемых операторов;

– текстом на входном языке другой системы программирования.

В последних двух случаях виртуальная машина генерирует ***исполняемый*** код, который в виде идентификатора образа памяти Image возвращается системе программирования для сохранения в понятийной модели. Тем самым реализуется однократное преобразование ***промежуточного кода*** в исполняемый. При повторном выполнении той же единицы вызова не требуется ее повторная компиляция.

Так как виртуальных машин может быть несколько, то может существовать и несколько образов одного и того же промежуточного кода. Более того, имеется возможность выражать промежуточный код в рамках одной программы различными средствами, для различных типов виртуальных машин. На рис. 4.12 схематически показана структура интерфейса виртуальной машины.

Метод Open служит для инициализации экземпляра виртуальной машины. При вызове метода ему передаются в качестве параметров адрес метода обратного вызова Callback, необходимый размер локальной (Stack) и глобальной (Memory) памяти.

Метод обратного вызова Callback используется для активации сервисов системы контекстного программирования, которые реализуются специальными командами виртуальной машины, например программными прерываниями.

Возвращаемое значение метода Open – идентификатор Handle созданного экземпляра виртуальной машины, для которой выделены необходимые ресурсы, в частности, локальная и глобальная память.

Метод Image необходим для компиляции промежуточного кода в образ памяти виртуальной машины, содержащий соответствующие этому промежуточному коду последовательность исполняемых команд целевой платформы. Параметрами метода является идентификатор экземпляра виртуальной машины Handle, собственно сам промежуточный код Code, объем необходимой для его выполнения статически распределяемой памяти

Locals и объем аргументов-сущностей Arguments из стека времени исполнения, которые извлекаются из этого стека перед выполнением единицы вызова.

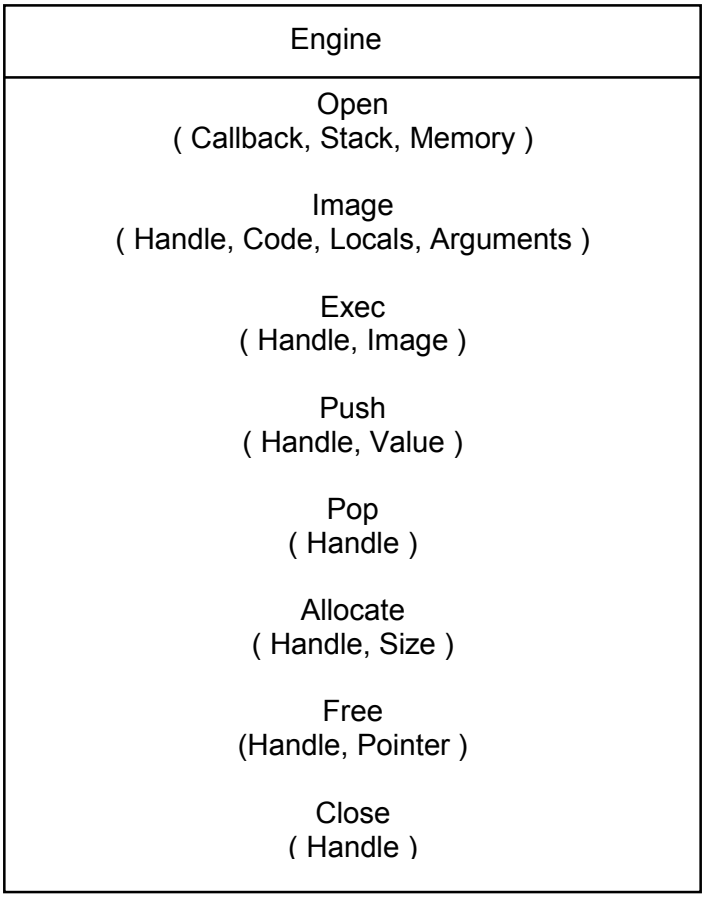

Рис. 4.12. Интерфейс виртуальной машины

Метод Exec необходим для выполнения единицы вызова, представленной в виде образа памяти Image, т.е. скомпилированной в исполняемый код целевой платформы. Методы Push и Pop служат для организации доступа к стеку времени исполнения виртуальной машины, а Allocate и Free – для доступа к глобальной памяти.

По завершению работы с экземпляром виртуальной машиной Handle вызывается метод Close, который уничтожает идентифицируемый экземпляр и освобождает выделенные для него ресурсы.

Заметим, что организация интерфейса Engine позволяет реализовать параллельные вычисления путем создания нескольких экземпляров виртуальных машин одного или различных типов.

4.5.3. Лексический анализатор

Лексический анализатор в контекстной технологии отличается по своей организации от лексических анализаторов других известных систем программирования. В частно-

сти, в нем реализовано контекстное выделение лексем. Последнее достигается следующими средствами (рис. 4.13):

– методами Backup–Restore для сохранения-восстановления текущего состояния входного потока Stream и связанных с этим указателем состоянием препроцессора Text и словаря макроопределений Vocabulary;

– методом Match для извлечения терминального представления Term искомой лексемы Lexeme, заданной некоторым регулярным выражением;

– методами Extract–Recover для извлечения текущей лексемы из входного потока и возврата лексемы во входной поток.

На рис. 4.13 указаны не только методы интерфейса Token, но и те сущности, которые агрегированы в интерфейс: входной поток Stream, препроцессор Text и словарь макроопределений Vocabulary.

Препроцессор Text обрабатывает стандартные типы макрооператоров:

– define – определение макроподстановки, возможно с параметрами;

– undef – удаление макроподстановки;

– if-then-else-end – условный макрооператор.

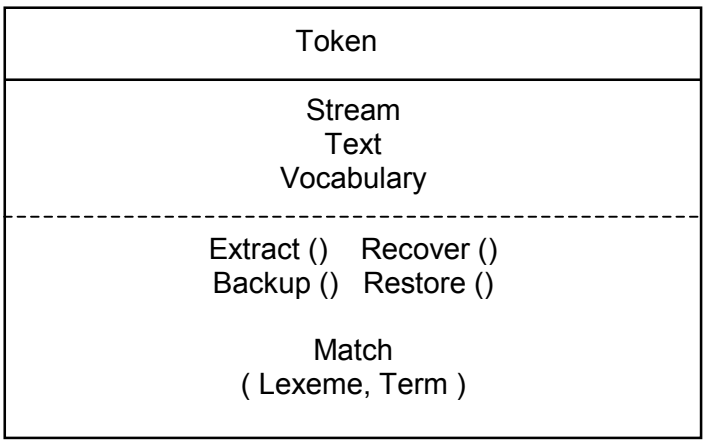

Рис. 4.13. Лексический анализатор

Особенностью описываемой реализации лексического анализатора Token является его способность к сохранению и восстановлению не только указателя во входном потоке, но и состояния препроцессора Text и словаря макроопределений Vocabulary.

4.5.4. Синтаксический анализатор

Основное назначение синтаксического анализатора – грамматический разбор той части входного текста, которая описывается грамматикой протоязыка контекстной технологии (см. Главу 2). Входными данными синтаксического анализатора Parser (рис. 4.14, метод Parse) является понятийная модель предметной области Cognition (неопределенная или определенная частично) и имя файла File с текстом программы.

В синтаксический анализатор агрегируются следующие сущности:

– Notion – текущее понятие понятийной модели Cognition;

– Aspect – обрабатываемый аспект текущего предложения;

– Code – формируемый код императива, компиляции или исполнения;

– Token – лексический анализатор, открытый для файла File;

– Dictionary – словарь алиасов понятий и элементов предложения.

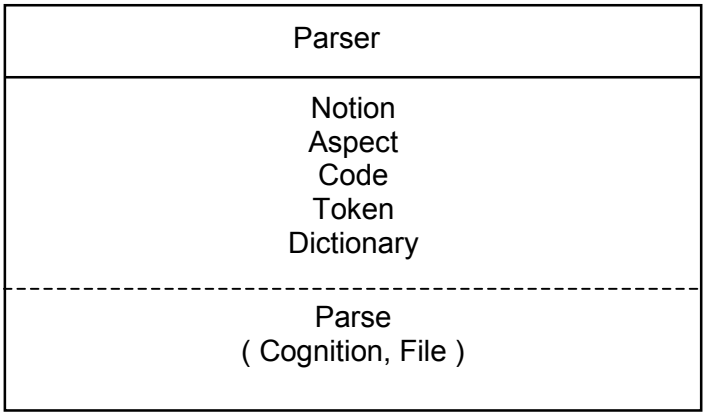

Рис. 4.14. Синтаксический анализатор

4.5.5. Семантический анализатор

Синтаксис входного текста, с одной стороны, частично определяется грамматикой протоязыка и обрабатывается синтаксическим анализатором Token. С другой стороны, синтаксис другой части текста задается описанием предложений, которые:

– определены ранее и содержатся в понятийной модели Cognition;

– определяются в текущем обрабатываемом тексте.

Проблемный язык, создаваемый в процессе обработки входного текста, используется для задания текста text (рис. 4.15), являющегося описанием:

– parse – действий, которые необходимо выполнить во время синтаксического анализа предложения;

– compile – действий, которые система программирования осуществляет всякий раз, когда будет распознано предложение, содержащее эту конструкцию и в порядке этого распознавания;

– pragmatic – прагматики понятия, представленной в виде одного или нескольких именованных императивов (***aspect*** – имя определяемого императива или семантический аспект предложения);

– situation – ситуационной части программы, в которой декларируются исходные данные и осуществляется решение некоторой прикладной задачи.

```
parse       →   '<' [text] '>'
compile     →   '[' [text] ']'
pragmatic   →   [aspect] '{' [text] '}'
situation   →   [aspect] '<' [text] '>'
```

Рис. 4.15. Предложения протоязыка, требующие семантического анализа

Очевидно, обработку конструкции text необходимо осуществлять иначе, чем текст, описываемый протоязыком, и использовать для этого методы, отличающиеся от методов синтаксического анализа. В системе контекстного программирования обработка конструкции text осуществляется семантическим анализатором Analyzer (рис. 4.16).

Основным методом семантического анализатора является метод Analyze, предназначенный для обработки конструкций text из входного потока и генерации промежуточного кода, соответствующего этому тексту. В качестве параметров методу передается:

– понятийная модель Cognition, в текущий момент дополняемая синтаксическим анализатором Parser;

– понятие Notion, при обработке которого встретилась конструкция text; или неопределенное понятие, если обрабатывается текст ситуации situation;

– текущий семантический аспект Aspect, распознанный синтаксическим анализатором;

– словарь алиасов понятий и элементов предложения Dictionary, построенный синтаксическим анализатором Parser при анализе текущего предложения;

– лексический анализатор Token, посредством которого семантический анализатор по мере необходимости будет извлекать лексемы из входного потока;

– единица вызова Code, для формирования которой запускается семантический анализатор.

Так как в системе контекстного программирования целевая вычислительная платформа непосредственно связана с семантическим анализатором (рис. 4.11), то для исполнения кода, порожденного текстом text конструкций parse и compile, необходим метод Exec. Метод Exec вызывается синтаксическим анализатором Parser для выполнения этого кода во время синтаксического анализа. Последнее позволяет реализовать достаточные сложные и заранее неопределяемые контекстные условия как для предложений понятийной модели (конструкция parse), так и для любого элемента анализируемого предложения (конструкция compile). Параметрами описываемого метода являются:

– единица вызова Code, подлежащая выполнению;

– контекстный элемент Item из состава предложения (понятие Notion или лексема Lexeme) – для конструкции compile; или само предложение Sentence – для конструкции parse;

– семантический аспект предложения Aspect – если предложение определено только для использования в этом аспекте, или аспект по умолчанию – если применение предложения не ограниченно конкретным аспектом.

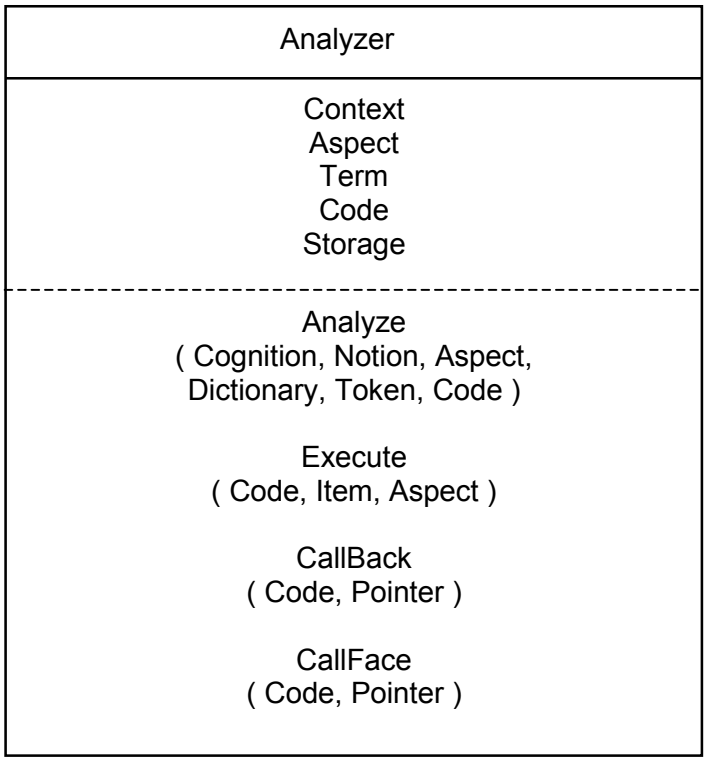

Рис. 4.16. Семантический анализатор

Методы обратного вызова Callback являются внешними точками входа семантического анализатора и вызываются виртуальной машиной для активации обратных сервисов системы контекстного программирования. Для извлечения данных, являющихся параметрами и непосредственно следующих за вызовом методу Callback передается соответствующий указатель Pointer.

Метод прямого вызова Callface предназначен для активации интерфейсных (прямых) сервисов, реализация которых зависит от используемого разработчиком стиля программирования. Интерфейсные сервисы определяются в виде кода компиляции, привязываемого к основным интерфейсным точкам системы контекстного программирования, которые определяют порядок создания, использования и уничтожения сущностей понятийной модели; генерацию кода; особенности выполнения ситуационной и императивной частей модели. Если соответствующий код не задан, система контекстного программирования использует его стандартную реализацию.

В семантический анализатор Analyzer агрегируются следующие сущности:

– Context – текущий контекст семантического анализатора;

– Aspect – текущий (динамический) аспект семантического анализа;

– Term – текущий терм из входного потока;

– Code – текущая исполняемая единица вызова;

– Storage – атрибутная память для взаимосвязи различных секций кода компиляции compile.

4.6. Реализация компилятора

Имеются ряд функций системы программирования, которые определяются методологией контекстного программирования и реализацией этой методологии в виде той или иной технологии. В частности, системные сервисы служат для выполнения таких функций и, тем самым, определяют особенности реализации системы в целом.

4.6.1. Сервисы обратного вызова

Сервисы обратного вызова необходимы для задания действий, которые кодируются в единицах вызова Code в виде специальных команд целевой платформы и непосредственно реализуются системой программирования. Система контекстного программирования поддерживает следующие сервисы обратного вызова:

– ***сервис компиляции*** для записи в область текущей единицы вызова Code данных различных типов и организации самого процесса формирования кода, а также для изменения входного потока, подлежащего обработке, например, задание входного потока в виде строки[67];

– ***информационный сервис*** для получения различных данных: из понятийной модели Cognition (свойства сущностей понятийной модели, элементы понятийной структуры), из описателя текущего понятия Notion (имя понятия, его идентификатор, обобщаемые или агрегируемые им понятия и др.), из обрабатываемого предложения Sentence (синтаксис предложения, его элементы, свойства предложения, и др.) или его текущего элемента Item (имя и тип элемента, терм из входного потока, которым выражена лексема, целочисленная интерпретация этого терма и др.);

[67] Перенаправление входного потока, осуществляемое таким образом, что анализируемым текстом становится некоторая произвольная строка, позволяет, в частности, реализовать метод трансформационных грамматик [270], при котором грамматический разбор текста разбивается на две части: разбор поверхностной и глубинной структуры фразы. Поверхностная структура выражает внешнюю форму представления текста, в то время как глубинная структура отражает его смысловую организацию.

– ***сервис состояний*** синтаксического и семантического анализаторов, например, их текущий синтаксический или семантический аспект Aspect; изменение состояний анализаторов, и др.;

– ***сервис контекстных условий***, служащий для определения предложений, описывающих временные конструкции локального уровня, например, временные переменные, передачу сообщений о невозможности успешного завершения анализа текста из-за невыполнения того или иного контекстного условия;

– ***сервис атрибутной памяти***, предназначенный для создания ячеек для хранения данных, полученных при выполнении текстов компиляции предыдущих элементов предложения (связь по данным между различными текстами компиляции).

4.6.2. Сервисы прямого вызова

Сервисы прямого вызова предназначены для переопределения действий, выполняемых системой программирования в интерфейсных точках. В отличие от сервисов обратного вызова, которые связаны с технологией контекстного программирования и не могут быть переопределены, сервисы прямого вызова переопределяются в программе по мере необходимости. Реализация этих сервисов зависит от используемых разработчиком парадигм и стилей программирования.

Пользователям системы программирования предоставляется возможно определить нестандартным образом реализацию основных стадий ***жизненного цикла*** понятий в виде тех действия, которые выполняются всякий раз, когда сущность, принадлежащая этому понятию изменяется:

– декларируется (создается ее описание в таблице идентификаторов);

– создается (выделяются ресурсы, необходимые для представления сущности);

– извлекается из сущности-агрегата (реализуется абстракция декомпозиции);

– преобразуется перед записью в стек времени исполнения для передачи единице вызова в качестве параметра (если соответствующее преобразование не задано явно);

– реплицируется на вершине стека исполнения перед возвратом из единицы вызова в качестве результата;

– уничтожается при выходе из зоны видимости.

Другая группа сервисов прямого вызова позволяет переопределить процесс ***генерации*** кода. При этом предоставляется возможность задать действия, которые требуется выполнять для записи в область кода Code:

– данных различных типов;

– вызова другой единицы вызова;

– обращение к сущности, переданной в качестве аргумента.

Третья группа интерфейсных сервисов используется для спецификации действий, выполняемых перед и после ***активации*** единиц вызова, являющихся императивами предложения, и единиц вызова, построенных для выполнения ситуационной части программы.

Переопределение действий, выполняемых в интерфейсных точках, описывается детализированными правила протоязыка контекстной технологии (рис. 4.17). По сравнению с ранее определенными конструкциями протоязыка, конструкции на рис. 4.17 включают такие нетерминальные знаки как:

– existor – задает текст компиляции, выполняемый во время создания constructor и уничтожения destructor сущности определяемого понятия ***notion***;

– declarator – переопределяет форму декларации сущности и соответствующий текст компиляции (по умолчанию создается декларатор вида: '***notion***' "[A-Za-z][A-Za-z0-9]+", где '***notion***' – терм с именем определяемого понятия, а шаблон определяет правила именования сущностей определяемого понятия);

– alterator – задает текст компиляции, выполняемый во время репликации replicator или конвертации convertor сущности при ее возврате из единицы вызова или передаче единице вызова в качестве параметра.

essences → differenciation [existor] ***notion*** [declarator] [integration [alterator]]
existor → [constructor] [parse [destructor]]
declarator → term [pattern] [compile]
alterator → [replicator] [parse [convertor]]
differenciation→ '(' [notions] ')'
integration → '(' [notions] ')'
notions → notion [declarator]
notion [declarator] notions

Рис. 4.17. Переопределение сервисов прямого вызова

Декларатор, заданный при описании интегрированных понятий в конструкции integration, своим текстом компиляции определяет порядок извлечения агрегированных сущностей из сущности-агрегата.

Для задания интерфейсов генерации и активации используются те же конструкции, что и для переопределения жизненного цикла понятий, но примененные при описании «пустого» понятия.

4.6.3. Области видимости

Как правило, проверку контекстных условий связывают с созданием и использованием сущностей, имеющих различную семантическую роль [133]. Задачей контекстного анализа является установление правильности использования таких сущностей. Наиболее

часто решаемой задачей является определение существования сущности и проверка соответствия ее использования заданному для нее контексту [208].

В процессе обработки входного потока система контекстного программирования создает и уничтожает сущности различных понятий, причем особенности их жизненного цикла могут быть определены в тексте программы в виде соответствующих кодов компиляции. Последнее эквивалентно динамическому созданию и удалению в процессе обработки текста некоторых синтаксических правил и их семантических интерпретаций.

Для обозначения участков текста, в которых применимы те или иные динамически создаваемые синтаксические правила и их динамическая изменяемая семантика, определяют области действия и области видимости.

Область действия сущности – это фрагмент текста, в котором действуют синтаксические правила выражения этой сущности и ее динамическая семантическая роль.

Область видимости отличается от области действия тем, что в области действия сущность существует, в то время как в области видимости он может быть использована (указана, выражена). Как правило, в область видимости включаются те фрагменты текста, в которых отсутствует определение других сущностей, имеющих тот же синтаксис, но различающихся семантическими ролями.

Область действия сущностей, объявленных при описании синтаксиса предложения, – во всех далее следующих определениях (императивах) и внутри вызываемых из них единиц вызова. Однако область видимости сущностей – только текст определения (императива) текущего предложения.

4.6.4. Динамическая семантика

В языках программирования различают статическую и динамическую семантику [208]. Семантика, описываемая при задании правил вывода грамматики языка, называется ***статической***, а семантика получаемой в результате компиляции программы последовательности команд – ***динамической***.

Однако разделение семантики проблемного языка на статическую и динамическую в классическом выше смысле не представляется возможным по причине отсутствия в системе контекстного программирования явного разделения компиляции описания языка и компиляцией текста, написанного на этом языке. По этой причине под ***динамической семантикой*** будем понимать временное создание в фазе компиляции вспомогательных предложений, которые удаляются при выходе из области видимости компилируемого фрагмента текста.

Сущности, с которыми оперирует система контекстного программирования, являются, по своей сути, данными, выражающими соответствующие этим сущностям понятия.

Сущности как глобальной, так и локальной видимости могут использоваться внутри других единиц вызова. В этом случае сущности передаются в качесстве опреандов, например, через стек операндов, а синтаксис и семантика их использования определяются описанием соответствующих предложений.

Для описания сущностей, создаваемых во время компиляции текста, написанного на проблемном языке, необходимо предусмотреть механизм динамического описания их семантики. Семантическая роль таких сущностей определяется понятием, к которому эти сущности принадлежат, а синтаксис их выражения задается некоторым предложением, в частном случае, выражаемом терминальной строкой –бесконтекстным именем сущности.

Таким образом, для задания синтаксиса и семантики динамически создавемых сущностей необходимо предусмотреть механизм добавления и удаления предложений, состоящих, например, из терма-имени и понятия-результата. Такие предложения в языках программирования соответсвуют описаниям переменных, создаваемых в процессе компиляции текста программы и сохраняемых в таблице идентификаторов компилятора.

В системе контекстного программирования переменные создаются во время объявления сущности и используются синтаксическим анализатором при грамматическом разборе текста в виде предложения, временно добавленного в список предложений соответствующего понятия. После уничтожения сущности это предложение должно быть удалено.

Хранение предложений для временных сущностей-переменных в системе контекстного программирования организовано в понятийной модели, например, в специальном словаре динамических сущностей, который выполняет ту же роль, что и таблица идентификаторов у компиляторов языков программирования. В этом случае, предложения словаря используются при грамматическом разборе текста и имеют наивысший приоритет по отношению к другим предложениям модели.

Если в области действия одной сущности создается другая сущность с тем же синтаксисом и различающейся семантической ролью, необходимо добавить еще одно предложение с большим приоритетом, чем первое (расположенное после первого). Последнее позволяет создать область действия для второй сущности-переменной путем ограничения области видимости первой.

4.6.5. Глобальная и локальная память

Появление контекстных условий в языках программирования вызвано тем, что формализм контекстно-свободных грамматик и соответствующий ему механизм порождения и распознавания текстов магазинным автоматом недостаточны для реализации общих алгоритмических моделей.

Универсальная алгоритмическая модель в современном понимании эквивалентна произвольной грамматике и требует такого же универсального устройства для своей реализации, как, например, машина Тьюринга. Машина Тьюринга имеет произвольный доступ к запоминающему устройству, в то время как магазинный автомат реализован на основе стека, где непосредственно доступна только его вершина. Следовательно, для проверки контекстных условий необходимо предусмотреть средства для семантически интерпретируемого доступа к произвольным ячейкам запоминающих устройств.

Для реализации возможностей универсальной алгоритмической модели в системе контекстного программирования предусмотрена глобальная память (рис. 4.9). Глобальная память предназначена для создания сущностей с глобальной видимостью. Выделение, использование и освобождение этой памяти осуществляется специальными командами виртуальной машины. Область действия глобальных идентификаторов простирается от места их создания до места уничтожения, а область видимости ограничена скобками, где эти объекты созданы.

Интерфейс виртуальной машины (рис. 4.12) содержит методы выделения Allocate и освобождения Free некоторого участка глобальной памяти, или блока. При выделении памяти создается объект, называемый ссылкой. Один и тот же блок может быть выделен несколько раз, т.е. иметь несколько ссылок. Для учета количества ссылок, каждый блок содержит счетчик. Метод Free уменьшает счетчик, а при его равенстве нулю освобождает участок памяти, занятый блоком. Такой механизм позволяет реализовать современные методы управления памятью и «сборки мусора».

На этой основе может быть создан механизм создания и уничтожения сущностей сложной структуры. Если сущность помещается в одной или нескольких ячейках стека операндов, то такая сущность является простой и передается единицам вызова по значению, через стек операндов. Если сущность требует большого объема памяти, то она рассматривается как сложная и для ее размещения используется глобальная память. Передача сложных сущностей осуществляется по ссылке. Для доступа к частям сложной сущности может использоваться тот же механизм ссылок. В этом случае ссылка принадлежит блоку, но указывает на некоторый участок глобальной памяти.

Локальная память предназначена для создания сущностей с локальной видимостью, ограничиваемой скобками (фигурными, квадратными, угловыми). Локальная область памяти выделяется при входе в единицу вызова, а при выходе – освобождается. Выделение и освобождение локальной памяти являются интерфейсными точками и могут быть переопределены в процессе программирования.

4.6.6. Знаки пунктуации

Для обеспечения правильной (однозначной) интерпретации текстов необходимо предусмотреть средства их структуризации. Как показано в 4.2, грамматический разбор реализуется «жадным» алгоритмом, который при структурном распознавании предложения пытается отождествить с искомым понятием наибольшую часть входного потока.

Для управления «жадностью» грамматического разбора и повышения его эффективности в системе контекстного программирования используются дополнительные средства структуризации текста программы – знаки пунктуации. ***Знаки пунктуации*** служат для задания границ предложений и являются своего рода точками останова структурного распознавания при просмотре текста программы вперед.

Заметим, что в протоязыке контекстной технологии не предусмотрено специальных возможностей для определения знаков пунктуации. Как видно из следующего примера, такие средства и не требуются.

Пример 4.8. Рассмотрим следующие предложения, выражающие «пустое» понятие:

```
() ()
        ',' {}
        '.' [ … ] {}
```

где задается два знака пунктуации: точка и запятая.

Несмотря на то, что первое предложение не содержит определения семантики, его семантическая роль – остановка структурного распознавания предложения, после которого записана запятая. Действительно, для фрагмента понятийной модели

```
'not' Boolean [ ... ] {}
Boolean 'and' Boolean [ ... ] {}
Boolean 'or' Boolean [ ... ] {}
Boolean `a` 'imp' `b` Boolean { ... }
```

текст ситуационной части '<not a or b and c>' и '<not a or b, and c>' будет интерпретирован по разному: в первом случае как $\overline{a} \vee (b \,\&\, c)$, во втором – как $(\overline{a} \vee b) \,\&\, c$, где использованы знаки операций отрицания, дизъюнкции и конъюнкции соответственно.

В свою очередь, предложение «точка» содержит некоторое определение семантики (показано многоточием втексте времени компиляции). Например, в рассматриваемом случае возможно описание сементики этого предложения таким образом, что появление точки в тексте программы будет интерпретировано путем вызова соответствующего сервиса системы программирования как выполнение той части кода, которая уже сгенерирована семантическим анализатором.

Тогда ситуация '<not a or b. and c>' будет интерпретирована как разбор и вычисление 'not a or b' при «пустом» контексте, а затем разбор и выполнение 'and c' при начальном контексте Boolean и с учетом результатам выполнения первой части, сохраненного на вершине стека операндов.

Таким образом, в отличие от запятой, которая только разграничивает предложения, точка используется для указания системе программирования на необходимость выполнения уже откомпилированного текста. ♦

4.7. Заключительные замечания

Контекстная технология основана на понятийном анализе и позволяет объективировать (формализовать) описание предметной области в более устойчивых формах, чем это предполагает ее объектно-ориентированный прототип.

Понятийный анализ как предельная форма абстракции познавательной деятельности позволяет ввести в использование и определить систему понятий, через которую знания о некоторой предметной области выражаются наиболее полно и компактно. Для этого найдены средства задания синтаксиса понятий, их семантической интерпретации, а также средства для определения связи понятий между собой, что позволяет задать различия между понятием как категорией и его языковым выражением.

Известные методы грамматического разбора не могут быть использованы в контекстной технологии в виду того, что на вид контекстно-свободных грамматик, описывающих тексты программ на проблемном языке, не накладывается никаких ограничений, которые необходимы в других методах. В разработанном методе разнесенного грамматического разбора определение применимости предложения разделяется на две части – на контекстное сопоставление, осуществляемое при просмотре анализируемого текста назад, и структурное распознавание, выполняемое при просмотре вперед.

Разнесенный грамматический разбор позволяет выполнять анализ текста, порожденного, в том числе и неоднозначными грамматиками. Эффективность метода в этом случае напрямую зависит от неоднозначности описания понятийной модели. Ввиду того, что целью создания понятийной модели является адекватное решаемым задачам описание некоторой предметной области, следует ожидать небольшого числа неоднозначностей. С другой стороны, введение в понятийную модель неоднозначностей служит улучшению выразительных качеств проблемного языка, создаваемого в процессе декомпозиции предметной области.

В заключение укажем, что понятийная модель, представленная на протоязыке контекстной технологии, позволяет не только объективировать знания о предметной области,

но и методом разнесенного грамматического разбора произвести обработку этих знаний и синтезировать эффективную процедуру решения прикладной задачи путем дискретной обработки данных.

Исходя из представлений о прагматике предусмотрена возможность различной семантической интерпретации понятий, которая осуществляется как с учетом формы их языкового выражения, так и с учетом контекста использования. Последнее позволяет учесть познающего субъекта, его цели и задачи при формировании и использовании определяемых понятийных моделей.

Формируемая при понятийном анализе модель, а также ее семантическая интерпретация, позволяют создавать базы знаний для различных предметных областей, которые могут как пополняться, так использоваться для создания других баз знаний в качестве их составных частей.

Вычислительный эксперимент, выполненный с системой контекстного программирования, подтвердил обоснованность базовых принципов, использованных в основе понятийного анализа и контекстной технологии.

Выводы к Главе 4

1. Обоснованы базовые принципы контекстной технологии обработки данных и показано, что для представления понятийной модели предметной области достаточно использования формализма контекстно-свободных грамматик, дополненного средствами для задания контекстов нетерминальных понятий языка.

2. Показано, что выразительные возможности определяемого в контекстной технологии проблемного языка эквивалентны контекстным языкам по классификации Хомского.

3. Разработан метод разнесенного грамматического разбора и показано, что он позволяет выполнять эффективный, по сравнении с другими известными методами, анализ текста, порожденного контекстными и неоднозначными грамматиками.

4. Обоснована архитектура системы контекстного программирования и показана возможность реализации таких принципов, как компиляция текста на проблемном языке в процессе его определения, прямая подстановка кода и вложенный процедурный вызов как средства повышения эффективности обработки данных, единственность и достаточность аксиомы для объявления базовых семантических категорий, необходимое чередование фаз грамматического разбора, компиляции и выполнения как в процессе обработки понятийной модели, так и при решении прикладной задачи.

5. Разработана организация системы контекстного программирования путем ее декомпозиции на такие структурные части, как входной поток, лексический, синтаксический и семантический анализаторы, целевые вычислительные платформы и понятийную модель.

6. Реализованы средства альтернативного определения семантики понятийной модели средствами различных уровней: от команд процессора целевой вычислительной платформы до текстов целевой системы программирования.

7. Разработан рабочий прототип системы контекстного программирования и выполнено его экспериментальное исследование, в результате которого найдены необходимые сервисы системы контекстного программирования, способы описания различных парадигм (стилей) программирования, правила задания динамического синтаксиса и динамической семантики, методы доступа к глобальной и локальной памяти, способы структуризации текста программы.

Глава 5.
Алгебраическая декомпозиция

Существует класс задач, для эффективного решения которых не применимы известные методологии анализа предметной области и соответствующие им технологии программирования, в том числе и описанный ранее понятийный подход и основанная на этом подходе контекстная технология обработки данных. Решение таких задач основывается на дискретной обработки данных, для описания которой использование высокоуровневых моделей не представляется возможным ввиду отсутствия содержательной интерпретации выполняемого преобразования данных. Например, такие задачи возникают в цифровой обработке сигналов, проектировании дискретных устройств, первичной обработке измерительных данных, и в других областях, характерной особенностью которых является представление исходных и результирующих данных в виде таблиц (векторов, двумерных и многомерных массивов и других однородных структур данных).

В рассматриваемом случае, для описания дискретной обработки дискретная функция, заданная в табличном виде, декомпозируется и представляется в виде композиции функций меньшей размерности. Так, функция f от n переменных $X=\{x_0,x_1,\ldots,x_{n-1}\}$ на каждом шаге представляется как функция от других функций g и h, $f(X)=g(X',h(X''))$, где X', X'' – подмножества X [338]. Полученное в результате описание исходной функции, заданное с точностью до элементарных функций (операций) над входными переменными, реализуется в виде дискретного устройства или программы для вычислительного средства.

Настоящая глава посвящена исследованию алгебраической декомпозиции дискретных функций в наиболее общей постановке задачи, когда с целью получения эффективных описаний дискретной обработки данных переменные и функции принимают значения на произвольных конечных множествах, а выбор алгебраических операций не ограничен каким-либо их подмножеством.

5.1. Содержательная постановка задачи

Поставим задачу синтеза формульного представления дискретной функции. Для этого рассмотрим алгебраическое разложение функции $f(X)$, которое осуществим по функциям θ_i одной или нескольких переменных X':

$$f(X)=\sum\theta_i(X')\times a_i(X''), \tag{5.1}$$

где a_i – функции (коэффициенты разложения), зависящие от оставшейся части переменных X''. На следующем шаге разложению подвергнем функции a_i, возможно, в другой алгебраической системе и по другим функциям. Описанный процесс будем повторять до тех пор, пока в качестве коэффициентов не будут получены константы.

Частным видом алгебраической декомпозиции является решающее разложение:

$$f(X) = \sum \theta_i(x) \times a_i(X \setminus x), \tag{5.2}$$

где θ_i – система решающих функций, a_i – остаточные функции, \ – знак разности множеств. В случае, когда $X' = X$ и $X'' = \varnothing$, имеем спектральную форму,

$$f(X) = \sum \theta_i(X) \times a_i, \tag{5.3}$$

где θ_i – система спектральных функций, a_i – коэффициенты разложения.

При алгебраической декомпозиции тип базиса $\Omega = \{+, \times\}$ определяется образующими его алгебраическими операциями. Для булевых функций используется классический базис Буля [265] с операциями конъюнкции и дизъюнкции, базис Жегалкина [75] с операциями неэквиваленции и конъюнкции, арифметический базис [161, 167]. В многозначной логике применяются базисы с операциями максимума и минимума [255], сложения по модулю k и минимума [281], операцией арифметического сложения и поразрядными логические операциями [71], арифметическими операциями на кольце целых чисел [355], операциями полей Галуа [272, 324].

Суть подхода заключается в объединении алгебраических методов, разделительной декомпозиции и ортогональных разложений в широком спектре алгебраических систем. После разделения переменных на два непересекающихся подмножества, исходная функция разлагается по системе частичных функций, зависящих от первого подмножества переменных, в результате чего получаются частичные функции (многомерные коэффициенты разложения), зависящие от второго подмножества. После представления последних формулами, синтезируется искомая форма исходной функции. Классы операций, образующих базис Ω и соответствующий ему вид частичных функций θ_i из (5.1), получим исходя из существования методов нахождения коэффициентов a_i при известных (заданных) значениях функции f.

Основные результаты настоящей главы опубликованы в работах [59, 63].

5.2. Основные понятия и определения

Для дискретного кодирования данных выберем в качестве универсального множество целых чисел $N_0 = \{..., -1, 0, 1, ...\}$. Когда необходимо задать конечное множество из k

элементов, будем использовать подмножество N_k этого множества, $N_k = \{0, 1, ..., k-1\}$, где $k > 0$ – количество элементов в N_k.

5.2.1. Дискретные функции

Функция f одной переменной x, заданная на множестве N_k и принимающая значения на множестве N_{k_f}, есть отображение N_k в N_{k_f}, такое что каждый элемент x области определения N_k связан (соответствует) не более чем с одним элементом $y = f(x)$ области значений N_{k_f}. Число k назовем ***значностью переменной***, а k_f – ***значностью функции***. Функция является ***дискретной***, если ее область определения и область значений – конечные множества.

Пусть функция f задана на множестве $N_{k_0} \times N_{k_1} \times \ldots \times N_{k_{n-1}}$ и принимает значения на множестве N_{k_f}, где $\times$ – декартово произведение множеств. В этом случае функция зависит от n переменных x_0, x_1, …, x_{n-1} со значностями k_0, k_1, …, k_{n-1} соответственно, т.е. $y = f(x_0, x_1, \ldots, x_{n-1})$, где $y \in N_{k_f}$, а $x_j \in N_{k_j}$ ($j = \overline{0, n-1}$).

Произвольная дискретная функция f задается вектором значений $\mathbf{F} = [f_0\ f_1 \ldots f_{m-1}]$ длины m и вектором значностей переменных $\mathbf{K} = [k_0\ k_1 \ldots k_{n-1}]$ длины n, $m = k_0 k_1 \ldots k_{n-1}$, где для обозначения векторов использованы квадратные скобки. Если функция задана только вектором $\mathbf{F}$, то существует множество векторов $\mathbf{K}$, которые могут быть использованы для доопределения функции. Количество таких векторов равно количеству представлений числа m в виде произведения целых чисел, больших единицы. Функцию, заданную только вектором значений длины m, будем называть m-***функцией***.

Для задания дискретной функции используем ее таблицу истинности с числом строк $m = k_0 k_1 \ldots k_{n-1}$ (табл. 5.1).

Пример 5.1. Пусть на множестве $N_3 = \{0, 1, 2\}$ задана переменная x' значности $k' = 3$, а на множестве $N_2 = \{0, 1\}$ – переменная x'' значности k''. Представим функцию f значности $k_f = 4$ зависящей от этих переменных. Так как значность функции равна 4, то она принимает значения на множестве $N_4 = \{0, 1, 2, 3\}$. Определим f таблицей истинности с числом строк $m = k'k'' = 6$ (табл. 5.2). ♦

Таблица 5.1. Таблица истинности

x	x_{n-1}	…	x_1	x_0	f
0	0	…	0	0	f_0
1	0	…	0	1	f_1
…	…	…	…	…	…
k_0-1	0	…	0	k_0-1	f_{k_0-1}
…	…	…	…	…	…
i	i_{n-1}	…	i_1	i_0	f_i
…	…	…	…	…	…
$m-1$	$k_{n-1}-1$	…	k_1-1	k_0-1	f_{m-1}

5.2.2. Системы счисления

Установим взаимно однозначное соответствие между значением переменной $x \in N_m$ и значениями переменных $x_j \in N_{k_j}$ путем представления числа x в n-разрядной позиционной системе счисления с основаниями, определяемыми значностями переменных.

Таблица 5.2. Дискретная функция

x	x''	x'	f
0	0	0	3
1	0	1	0
2	0	2	1
3	1	0	2
4	1	1	2
5	1	2	1

Представлением числа x в n-разрядной ***позиционной системе счисления*** с основаниями k_0, k_1, …, k_{n-1} называется упорядоченное множество из n чисел или цифр $(x_0, x_1, \ldots, x_{n-1})_{k_0\, k_1 \ldots k_{n-1}}$, таких что

$$x = \sum_{j=0}^{n-1} x_j w_j, \quad w_j = \prod_{t=0}^{j-1} k_t,$$

где $x_j \in \{0, 1, \ldots, k_j - 1\}$ – j-я цифра числа x с весом w_j, причем цифра x_0 имеет наименьший вес w_0, равный единице.

Для прямого и обратного преобразования числа x в его представление в системе счисления с различными основаниями цифр $(x_0, x_1, \ldots, x_{n-1})_{k_0\, k_1 \ldots k_{n-1}}$ используются следующие рекуррентные правила:

$$x(0) = x, \qquad \left.\begin{array}{r} x_t = x(t) \bmod k_t \\ x(t+1) = x(t) \ \text{div} \ k_t \end{array}\right\} \ (t = \overline{0, n-2}),\ x_{n-1} = x(n-1); \tag{5.4}$$

$$x(n-1) = x_{n-1}, \quad x(t-1) = x(t)k_{t-1} + x_{t-1} \ (t = \overline{n-1, 1}),\ x = x(0), \tag{5.5}$$

где $x(t)$ – временная переменная, изменяющая свое значение на каждой итерации t, mod обозначает остаток, а div – целую часть от деления первого операнда на второй.

Пусть функция заданна вектором значений $\mathbf{F}$ и вектором значностей переменных $\mathbf{K}$. Тогда выражение

$$\mathbf{F}(x) = \mathbf{F}(x_0, x_1, \ldots, x_{n-1})_{k_0\, k_1 \ldots k_{n-1}},$$

равно значению функции, когда переменные принимают значения x_0, x_1, …, x_{n-1}, а переменная x, равная номеру элемента вектора $\mathbf{F}$, представлена в позиционной системе счисления с основаниями $\mathbf{K}$. В этом случае выражение $x = (x_0, x_1, \ldots, x_{n-1})$ читается как последовательность цифр числа x в этой системе счисления, причем в этой записи первая цифра имеет наименьший вес.

Пример 5.2. Функцию f из примера 5.1 можно представить зависящей от одной переменной x со значностью $k = 6$, равной произведению значностей переменных x' и x''. Связь между переменной x и переменными x', x'' устанавливается формулами (5.4) и (5.5), $x = (x''\, x')_{2\,3}$. В частности, если $x' = 2$ и $x'' = 1$, то $x = (1\,2)_{2\,3} = 3 \cdot 1 + 2 = 5$. ♦

Если учесть, что дискретная функций определяется не только вектором значений, но значностями переменных, то один и тот же вектор длины m на разных наборах таких значностей образует различные функции[68]. Отсюда получаем следующую теорема.

Теорема 5.1. *Количество дискретных функций значности k_f и длиной вектора значений m равно M_f,*

$$M_f(k_f, m) = M_\chi(m)\, k_f^m, \tag{5.6}$$

где $M_\chi(m)$ – количество представлений числа m в виде произведения натуральных чисел, больших 1.

---

[68] Действительно, из анализа табл. 5.2 следует, что функции $f(x)$ и $f(x', x'')$ различные, так зависят от разного числа переменных. Эти функции задают различные зависимости, также, например, как функции $\sin(x)$ и $\sin(x + y)$ в теории функций действительных переменных. Такая же картина наблюдается и в том случае, когда число переменных у функций одинаково, но значности переменных, взятые с точностью до их перестановки, различны.

Доказательство. При фиксированном m имеем k_f^m различных характеристических векторов. Каждый характеристический вектор, в свою очередь, определяет $M_\chi(m)$ функций, различающихся значностями переменных. В итоге общее количество дискретных функций (m-функций) равно M_f и вычисляется по (5.6). ♦

Из выражения (5.6) видно, что количество функций многозначной логики превосходит количество функций в k-значной логике. Это позволяет одну и ту же дискретную обработку данных (m-функцию) представить большим числом способов, отличающихся различным разбиением массива входных данных на части (переменные).

5.2.3. Дискретные операции

При дискретной обработке преобразование входных данных в выходные происходит путем выполнения унарных, бинарных, тернарных и других операций.

Операцией называется функция, существенно зависящая от своих переменных. Местность операции определяется числом операндов (переменных), участвующих в формировании результата операции. Очевидно, если результат r-местной операции не зависит от одного из операндов, такую операцию следует рассматривать как $(r-1)$-местную.

Унарная операция может быть определена вектором, а бинарная – матрицей. Вектор $[y_0\, y_1 \ldots y_{k_0-1}]$ длины k_0, обозначаемый также как $[y_i \,|\, i=\overline{0,k_0-1}]$ или $[y_i]$, задает унарную операцию, а единственный операнд этой операции имеет значность k_0. Матрица размерности $k_0 \times k_1$, обозначаемая как $[y_{ij} \,|\, i=\overline{0,k_0-1},\ j=\overline{0,k_1-1}]$ или $[y_{ij}]$, является матрицей бинарной операции, если она содержит хотя бы одну строку (столбец), отличную от остальных строк (столбцов). В этом случае первый операнд имеет значность k_0, а второй – k_1.

Пусть унарная операция $\triangleright$ и бинарная операции $\circ$ заданы вектором $[y_i]$ и матрицей $[y_{ij}]$. Обозначим применение унарной операции $\triangleright$ к переменной x как $\triangleright x = [y_i](x) = y_x$, а бинарной операции $\circ$ к переменным x и z как $x \circ z = [y_{ij}](x,z) = x[y_{ij}]z = y_{xz}$.

Теорема 5.2. *Количество дискретных операций над n переменными со значности k_j ($j=\overline{0,n-1}$) равно M_o,*

$$M_o(k,k_0,k_1,\ldots,k_{n-1}) = \sum_{i=0}^{n-1}(-1)^i \sum_{j=1}^{C_n^i} k^{k_{j_0}k_{j_1}\ldots k_{j_{n-i}}}, \qquad (5.7)$$

где C_n^i *– число сочетаний из* n *по* i*;* k_{j_0}, k_{j_0}, ..., $k_{j_{n-i}}$ *пробегают все возможные сочетаний из* n *элементов* k_0, k_1, ..., k_{n-1} *по* $n-i$.

Доказательство. Произвольная константа является нульместной операцией. Количество таких операций k.

Произвольный вектор длиной k_0 задает унарную (одноместную) операцию значности k, если элементы этого вектора не одно и то же число. Количество таких операций $k^{k_0} - k$.

Произвольная $(k_1 \times k_0)$-матрица определяет бинарную (двуместную) операцию, если все строки и столбцы этой матрицы не одинаковы. Количество таких матриц $k^{k_0 k_1} - k^{k_0} - k^{k_1} + k$.

Тернарная операция представляется характеристическим вектором длиной $k_0 k_1 k_2$ или трехмерной таблицей, такой что составляющие ее двумерные таблицы не одинаковы. Число таких трехмерных таблиц равно $k^{k_0 k_1 k_2} - k^{k_0 k_1} - k^{k_0 k_2} - k^{k_1 k_2} + k^{k_0} + k^{k_1} + k^{k_2} - k$.

Далее индукцией по числу переменных убеждаемся в справедливости формулы (5.7). ♦

В k-значной логике формула (5.7) принимает более простой вид:

$$M_o(k,n) = \sum_{i=0}^{n-1} (-1)^i C_n^i k^{k^{n-i}} . \qquad (5.8)$$

5.2.4. Разделительная декомпозиция

Под ***декомпозицией дискретной функции*** будем понимать представление функции формулой, связывающей переменные и операции над ними со значением функции.

Универсальной декомпозиционной схемой является представление некоторой функции f в виде композиции функций θ и a_j ($j = \overline{1,m}$, $m \geq 1$) [279]:

$$f(X) = \theta(X', a_1(X''), a_2(X''), \ldots, a_m(X'')),$$

где X' и X'' – подмножества переменных, получаемые в результате разделения множества X. Если $X' \cap X'' \neq \varnothing$, то декомпозиция называется ***пересекающейся***, в противном случае – ***непересекающейся***.

Все известные декомпозиции (см. 1.2) основаны на ***разделении*** переменных. При разделении переменных множество $X = \{x_0, x_1, \ldots, x_{n-1}\}$ разбивается на два непересекающихся подмножества $X' = \{x'_0, x'_1, \ldots, x'_{n'-1}\}$ и $X'' = \{x''_0, x''_1, \ldots, x''_{n''-1}\}$ с числом переменных n' и n''. Определим значности подмножеств X' и X'' так:

$$k' = \prod_{(x_j \in X')} k_j = \prod_{j=0}^{n'-1} k'_j, \quad k'' = \prod_{(x_j \in X'')} k_j = \prod_{j=0}^{n''-1} k''_j.$$

Это позволяет рассматривать функцию зависящей от двух фиктивных переменных x' и x'' со значностями k' и k'' соответственно, где $k'k'' = m$.

Пример 5.3. Пусть функция определена векторами

$$\mathbf{F} = [3\,2\,2\,1\,0\,2\,3\,1\,2\,1\,0\,3\,1\,0\,3\,0\,2\,0\,0\,2\,0\,2\,2\,0], \quad \mathbf{K} = [2\,2\,3\,2],$$

т.е. зависит от переменных $X = \{x_0, x_1, x_2, x_3\}$ со значностями $k_0 = 2$, $k_1 = 2$, $k_2 = 3$ и $k_3 = 2$. Разобьем переменные на два подмножества $X' = \{x_1, x_3\}$ и $X'' = \{x_0, x_2\}$. Тогда $k' = k_1 k_3 = 4$ и $k'' = k_0 k_2 = 6$. Представим функцию в виде двумерной таблицы (табл. 5.3).

Таблица 5.3. Двумерная таблица функции

		x_0	0	1	0	1	0	1
		x_2	0	0	1	1	2	2
x_1	x_3	$x' \setminus x''$	0	1	2	3	4	5
0	0	0	3	2	0	2	2	0
1	0	1	2	1	3	1	1	3
0	1	2	1	0	2	0	0	2
1	1	3	3	2	0	2	2	0

При построении таблицы пересчет переменной $x \in N_{24}$ в переменные $x' \in N_4$ и $x'' \in N_6$ производился с использованием системы счисления с основаниями $\mathbf{K}' = [2\,2]$ и $\mathbf{K}'' = [2\,3]$ так,

$$\mathbf{F}(x)_{24} = \mathbf{F}(x_0, x_1, x_2, x_3)_{2232} = \mathbf{T}[(x_0, x_2)_{22}\ \ (x_1, x_3)_{23}] = \mathbf{T}[c\,r] = t_{rc},$$

где $\mathbf{T}$ – матрица функции (двумерная таблица), t_{rc} – элемент этой матрицы, а r (c) – ее строка (столбец). В частности:

$$t_{23} = \mathbf{T}[3\,2] = \mathbf{T}[(1,1)_{22}\ \ (0,1)_{23}] = \mathbf{F}(1,0,1,1)_{2232} = \mathbf{F}(17) = 0. \ \blacklozenge$$

Количество способов, которыми множество из n элементов может быть разделено на два непересекающихся подмножества, очевидно, равно $2^{n-1} - 1$.

При построении двумерной таблицы используются рекуррентные формулы (5.4) и (5.5), для которых необходима некоторая упорядоченность переменных в множествах X' и X''. Перестановка переменных x'_i и x'_j из множества X' при фиксированном порядке переменных из множества X'' приводит к перестановке строк результирующей таблицы, так как каждое число $x' = (x'_0, x'_1, \ldots, x'_{n'-1})$, или номер строки, имеет уже другое значение при неизменных значениях переменных, его составляющих. Аналогично, перестановка

переменных из множества X'' порождает перестановку столбцов в двумерной таблице функции.

Пример 5.4. Покажем, как изменится двумерная таблица функции из примера 5.3 при перестановке переменных x_0 и x_2 из множества X'' (табл. 5.4).

Из сравнения табл. 5.3 и табл. 5.4 видно, что перестановка переменных привела к перестановке столбцов. ♦

Таблица 5.4. Перестановка переменных

		x_2	0	1	2	0	1	2
		x_0	0	0	0	1	1	1
x_1	x_3	$x' \setminus x''$	0	1	2	3	4	5
0	0	0	3	0	2	2	2	0
1	0	1	2	3	1	1	1	3
0	1	2	1	2	0	0	0	2
1	1	3	3	0	2	2	2	0

5.2.5. Алгебраическая декомпозиция

Решающая (5.2) и спектральная (5.3) декомпозиции являются ***алгебраическими*** и выполняются в некоторой алгебре, образованной двумя операциями – сложением и умножением. В общем случае при алгебраической декомпозиции (5.1) произвольная m-функция значности $k_f \le k$ представляется разложением по некоторой системе k'-функций θ_i $(i = \overline{0, k'-1})$,

$$f(X) = \sum_{i=0}^{k'-1} \theta_i(X') \times a_i(X''), \tag{5.9}$$

где a_i – ***коэффициенты*** разложения, или некоторые k''-функции. Функции a_i и θ_i будем называть ***частичными*** для функции f.

Найдем условия, при которых разложение (5.9) существует. Для этого разделим множество переменных X на два непересекающихся множества X', X'' и представим m-функцию f значности k_f зависящей от двух переменных x' и x'' со значностями k' и k'', т.е. при $m = k'k''$. Если m – простое число (не представляется в виде произведения двух чисел), увеличим m и произвольным образом доопределим функцию в образовавшихся точках.

При подстановке в формулу (5.9) значений переменной $x' = \overline{0, k'-1}$ получаем систему алгебраических уравнений

$$\begin{cases} f(0,x'') & = & \theta_0(0)\times a_0(x'') & +\ldots+ & \theta_{k'-1}(0)\times a_{k'-1}(x''); \\ f(0,x'') & = & \theta_0(1)\times a_0(x'') & +\ldots+ & \theta_{k'-1}(1)\times a_{k'-1}(x''); \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ f(k'-1,x'') & = & \theta_0(k'-1)\times a_0(x'') & +\ldots+ & \theta_{k'-1}(k'-1)\times a_{k'-1}(x''), \end{cases} \tag{5.10}$$

из которой требуется найти a_i $(i=\overline{0,k'-1})$.

Определение 5.17. *Система функций* θ_i *называется* ***фундаментальной****, если сложение и умножение функций для всех значений переменных коммутативно, ассоциативно и выполняется закон дистрибутивности умножения функций относительно их сложения, т.е. при* $*\in\{+,\times\}$*, для любых* i, j, t *и всех* x, y, z

$$\theta_i(x)*\theta_j(y)=\theta_j(y)*\theta_i(x);$$
$$\theta_i(x)*\{\theta_j(y)*\theta_t(z)\}=\{\theta_i(x)*\theta_j(y)\}*\theta_t(z);$$
$$\theta_i(x)\times\{\theta_j(y)+\theta_t(z)\}=\{\theta_i(x)\times\theta_j(y)\}+\{\theta_i(x)\times\theta_t(z)\}.$$

По своей сути фундаментальные функции являются всевозможными гомоморфизмами множества N_k в свое подмножество, на котором сложение и умножение коммутативны и ассоциативны, а умножение дистрибутивно относительно сложения.

Определение 5.18. *Матрица прямого преобразования* $\mathbf{D}$ *называется* ***невырожденной****, если система уравнений (5.10) разрешима относительно* a_i *единственным образом.*

В матричном виде система (5.10) представляется так:

$$\mathbf{F}=\mathbf{D}\times\mathbf{A} \quad (\mathbf{A}=\mathbf{Q}\times\mathbf{F}), \tag{5.11}$$

где $\mathbf{F}$ и $\mathbf{A}$ – матрицы функции и коэффициентов размерности $k'\times k''$, $\mathbf{D}$ и $\mathbf{Q}$ – квадратная матрица прямого и обратного преобразования размерности $k'\times k'$ с элементами $d_{ij}=\theta_i(j)$ и $q_{ij}=\vartheta_i(j)$. Заметим, что столбцы матрицы $\mathbf{D}$ являются векторами частичных функций θ_i, а строки матрицы $\mathbf{A}$ – векторами частичных функций a_i.

Определение 5.19. *Преобразование (5.11) называется* ***ортогональным****, если для каждой невырожденной матрицы* $\mathbf{D}$ *существует такая матрица* $\mathbf{Q}$*, что в алгебре* R *имеет место выражение* $\mathbf{A}=\mathbf{Q}\times\mathbf{F}$.

Определение 5.20. *Построение матрицы* $\mathbf{Q}$ *по заданной матрице* $\mathbf{D}$ *будем называть* ***обращением*** $\mathbf{D}$ *и обозначать унарной матричной операцией* μ*:* $\mathbf{Q}=\mu\mathbf{D}$.

Так как столбцы $\mathbf{D}$ являются векторами частичных функций θ_i, то столбцы $\mathbf{Q}$ можно рассматривать как вектора функций ϑ_i, ортогональных θ_i в указанном смысле.

Существует несколько классов алгебр, позволяющих разрешить систему (5.10) относительно неизвестных функций $a_i(x'')$.

5.3. Образующие алгебры

Пусть имеется алгебра $R = \langle N_k, +, \times \rangle$, заданная на множестве N_k двумя бинарными операциями, условно называемыми сложением $+$ и умножением $\times$. При $k = 0$ алгебра определена на счетном множестве N_0, в противном случае – на конечном множестве из k элементов N_k.

Исследуем такие алгебры R, в которых разложение (5.9) возможно. Для этого операции сложения и умножения выберем таким образом, чтобы для заданной системы частичных функций θ_i система уравнений (5.10), представленная как

$$\sum_{i=0}^{k'-1} d_{ji} \times a_i(X'') = f_j(X'') \quad (j = \overline{0, k'-1}), \tag{5.12}$$

имела единственное решение относительно неизвестных функций-коэффициентов $a_i(X'')$, где $d_{ji} = \theta_i(j)$. Такие алгебры будем называть ***образующими***[69].

5.3.1. Алгебра логики

Рассмотрим алгебру $R_L = \langle N_k, +, \times \rangle$, в которой потребуем существование таких элементов σ и $\tau \neq \sigma$, что

$$a + \sigma = a, \quad \sigma + a = a, \quad \sigma \times a = \sigma, \ \tau \times a = a$$

для всех $a \in N_k$. Элемент σ назовем ***нулем***, а τ – ***единицей*** алгебры R_L.

Определение 5.21. *Матрицей* ***перестановок*** $\mathbf{P}_{k'}$ *порядка* k' *называется квадратная матрица, состоящая из нулей* σ *и единиц* τ*, и имеющая в каждой строке (столбце) ровно по одной единице.*

Теорема 5.3. *Произвольная функция* f *представима в алгебре логики* R_L *разложением (5.9), если матрица прямого преобразования* $\mathbf{D}$ *является матрицей перестановок.*

Тогда $\mathbf{Q} = \mathbf{D}^T$ *и* $\mathbf{Q} \times \mathbf{D} = \mathbf{E}$*, где* $\mathbf{E}$ *– единичная матрица,* T *– операция транспонирования.*

Доказательство. Из условия теоремы следует, что система уравнений (5.12) принимает вид

$$f(i, x'') = a_j(x'') \quad (i = \overline{0, k'-1}), \tag{5.13}$$

---

[69] Известно, что разложение произвольной функции в виде (5.9) возможно в полях и целостных кольцах (коммутативных кольцах с единицей и без делителей нуля). Следовательно, перечисленные алгебры являются образующими.

а j таково, что $d_{ij} = \tau$. Переупорядочим строки системы уравнений (5.13) в соответствии с естественной нумерацией коэффициентов a_j. Тогда $a_j(x'') = f(i, x'')$ ($j = \overline{0, k'-1}$), а i таково, что $q_{ji} = d_{ij} = \tau$. Образовав из получившихся элементов q_{ji} матрицу $\mathbf{Q}$, находим $\mathbf{Q} = \mathbf{D}^T$ и $\mathbf{Q} \times \mathbf{D} = \mathbf{E}$. ♦

Пример 5.5. Продемонстрируем идею доказательства теоремы 5.3 на примере. Пусть задана матрица перестановок

$$\mathbf{D} = \begin{bmatrix} \sigma & \tau & \sigma \\ \sigma & \sigma & \tau \\ \tau & \sigma & \sigma \end{bmatrix}.$$

Тогда система уравнений (5.12) для этой матрицы имеет вид:

$$\begin{cases} f(0, x'') &= \sigma \times a_0(x'') + \tau \times a_1(x'') + \sigma \times a_2(x''); \\ f(1, x'') &= \sigma \times a_0(x'') + \sigma \times a_1(x'') + \tau \times a_2(x''); \\ f(2, x'') &= \tau \times a_0(x'') + \sigma \times a_1(x'') + \sigma \times a_2(x''). \end{cases}$$

После тождественных преобразований этой системы на основе свойств специальных элементов алгебры σ и τ получаем:

$$\begin{cases} f(0, x'') &= a_1(x''); \\ f(1, x'') &= a_2(x''); \\ f(2, x'') &= a_0(x''). \end{cases}$$

Переставим строки полученной системы в порядке возрастания индексов функций-коэффициентов $a_j(x'')$,

$$\begin{cases} a_0(x'') &= f(2, x''); \\ a_1(x'') &= f(0, x''); \\ a_2(x'') &= f(1, x''). \end{cases}$$

Представим правые части уравнений в виде линейных комбинаций функций $f(i, x'')$ с коэффициентами, равными σ и τ,

$$\begin{cases} a_0(x'') &= \sigma \times f(0, x'') + \sigma \times f(1, x'') + \tau \times f(2, x''); \\ a_1(x'') &= \tau \times f(0, x'') + \sigma \times f(1, x'') + \sigma \times f(2, x''); \\ a_2(x'') &= \sigma \times f(0, x'') + \tau \times f(1, x'') + \sigma \times f(2, x''), \end{cases}$$

и составим из полученных коэффициентов матрицу $\mathbf{Q}$,

$$\mathbf{Q} = \begin{bmatrix} \sigma & \sigma & \tau \\ \tau & \sigma & \sigma \\ \sigma & \tau & \sigma \end{bmatrix}.$$

Находим, что $\mathbf{Q} = \mathbf{D}^T$. Вычислим произведение матриц $\mathbf{Q}$ и $\mathbf{D}$ в алгебре логики,

$$\mathbf{Q}\times\mathbf{D}=\begin{bmatrix}\sigma&\sigma&\tau\\ \tau&\sigma&\sigma\\ \sigma&\tau&\sigma\end{bmatrix}\times\begin{bmatrix}\sigma&\tau&\sigma\\ \sigma&\sigma&\tau\\ \tau&\sigma&\sigma\end{bmatrix}=\begin{bmatrix}\tau&\sigma&\sigma\\ \sigma&\tau&\sigma\\ \sigma&\sigma&\tau\end{bmatrix}=\mathbf{E}.\ \blacklozenge$$

Произвольную матрицу $\mathbf{P}_{k'}$ можно задать перестановкой $\alpha=(\alpha_0,\alpha_1,\ldots,\alpha_{k'-1})$ элементов $\alpha_i\in N_{k'}$. Матрица $\mathbf{P}_{k'}$ является невырожденной и задает фундаментальную систему функций. Количество таких систем в алгебре R_L (или количество различных представлений функции f) ровно $M_L=k'!$.

Пример 5.6. Пусть операции сложения и умножения – любые из операций, задаваемые матрицами $+$ и $\times$ с элементами $s_{ij}=i+j$ и $p_{ij}=i\times j$ соответственно:

$$+=\begin{bmatrix}0&1&2&3\\ 1&*&*&*\\ 2&*&*&*\\ 3&*&*&*\end{bmatrix},\quad \times=\begin{bmatrix}0&0&0&0\\ *&*&*&*\\ *&*&*&*\\ 0&1&2&3\end{bmatrix},$$

где $*$ – безразличное значение[70], т.е. в месте $*$ может находиться произвольный элемент N_4. Видно, что $\sigma=0$ и $\tau=3$.

Теперь рассмотрим функцию f из табл. 5.2. Тогда из формулы (5.9) следует

$$f(x',x'')=\theta_0(x')\times a_0(x'')+\theta_1(x')\times a_1(x'')+\theta_2(x')\times a_2(x'').$$

Задав перестановку $\alpha=(2\,0\,1)$ элементов N_3 получим

$$\mathbf{D}=\begin{bmatrix}\theta_0(0)&\theta_1(0)&\theta_2(0)\\ \theta_0(1)&\theta_1(1)&\theta_2(1)\\ \theta_0(2)&\theta_1(2)&\theta_2(2)\end{bmatrix}=\begin{bmatrix}0&0&3\\ 3&0&0\\ 0&3&0\end{bmatrix},\quad \mathbf{Q}=\mathbf{D}^T=\begin{bmatrix}0&3&0\\ 0&0&3\\ 3&0&0\end{bmatrix},$$

$$\mathbf{A}=\mathbf{Q}\times\begin{bmatrix}f(0,x'')\\ f(1,x'')\\ f(2,x'')\end{bmatrix}=\begin{bmatrix}0&3&0\\ 0&0&3\\ 3&0&0\end{bmatrix}\times\begin{bmatrix}3&2\\ 0&2\\ 1&1\end{bmatrix}=\begin{bmatrix}0&2\\ 1&1\\ 3&2\end{bmatrix},\quad \mathbf{A}=\begin{bmatrix}a_0(x'')\\ a_0(x'')\\ a_0(x'')\end{bmatrix},$$

$$f(x',x'')=\begin{bmatrix}0\\3\\0\end{bmatrix}(x')\times\begin{bmatrix}0\\2\end{bmatrix}(x'')+\begin{bmatrix}0\\0\\3\end{bmatrix}(x')\times\begin{bmatrix}1\\1\end{bmatrix}(x'')+\begin{bmatrix}3\\0\\0\end{bmatrix}(x')\times\begin{bmatrix}3\\2\end{bmatrix}(x''),$$

где $[y_i](x)=y_x$ – унарная операция, задаваемая характеристическим вектором $[y_i]$. $\blacklozenge$

[70] В разложениях безразличные значения операций, помеченные знаком *, никогда не используются. Это гарантируется видом частичных функций, которые не допускают никаких других операндов (номеров строк и столбцов входа в матрицу операций), кроме определяемых в условии теоремы существованием коммутативного нуля по сложению, который является и левым нулем по умножению, а также левой единицы по умножению.

Заметим, что в алгебре логики R_L на множестве $B=\{\sigma,\tau\}$ с точностью до автоморфизма существуют две алгебраические подсистемы: булева алгебра $B_L=\langle B,\&,\vee\rangle$ (если $\tau+\tau=\tau$) или поле Жегалкина $P_L=\langle B,\oplus,\&\rangle$ (если $\tau+\tau=\sigma$), где $\&$, $\vee$ и $\oplus$ – соответственно обобщенные булевы операции конъюнкции, дизъюнкции и неэквиваленции, задаваемые с точностью до выбора элементов для нуля и единицы.

Можно показать, что если R_L включает B_L или P_L, то произвольная матрица из нулей и единиц образует в R_L фундаментальную систему функций.

Для представления функций в алгебре R_L в качестве сложения могут использоваться такие операции как сложение по модулю k, максимум, поразрядная дизъюнкция и неэквиваленция (исключающее ИЛИ); а в качестве умножения – умножение по модулю k, минимум, поразрядная конъюнкция. Для обеспечения замкнутости на множестве N_k поразрядных операций k выбирается равным степени 2 (степени простого числа).

5.3.2. Мультипликативная алгебра

Определим алгебру $R_M=\langle N_k,+,\times\rangle$, у которой операции удовлетворяют условиям алгебры R_L, а умножение дополнительно образует группу $G_M=\langle N_k\setminus\{\sigma\},\times\rangle$ на множестве N_k за исключением нулевого элемента σ. Последнее означает, что для любого элемента $a\in G_M$ существует обратный ему элемент a^{-1}, такой что $a\times a^{-1}=\tau$.

Определение 5.22. ***Мономиальной*** *матрицей* $\mathbf{M}_{k'}$ *порядка* k' *называется квадратная матрица, получаемая перестановкой строк (столбцов) диагональной матрицы с элементами* $d_i\neq\sigma$, $d_i\in N_k$.

Теорема 5.4. *Произвольная функция* f *представима в мультипликативной алгебре* R_M *разложением (5.2), если матрица прямого преобразования* $\mathbf{D}$ *является мономиальной.*

Тогда $\mathbf{Q}=\widetilde{\mathbf{D}}^T$ *и* $\mathbf{Q}\times\mathbf{D}=\mathbf{E}$, *где* $\mathbf{E}$ *– единичная матрица, а* $\widetilde{\mathbf{D}}$ *получается из* $\mathbf{D}$ *заменой ненулевых элементов на обратные в группе* G_M.

Доказательство. Из (5.12) получаем

$$f(i,x'')=d_{ij}\times a_j(x'')\quad(i=\overline{0,k'-1}),\tag{5.14}$$

где $d_{ij}\neq\sigma$. Используя свойства операции умножения, найдем a_j и переупорядочим строки (5.14) в соответствии с нумерацией коэффициентов a_j, что в итоге дает

$$a_j(x'')=q_{ji}\times f(i,x'')\quad(j=\overline{0,k'-1}),$$

где $q_{ji} = d_{ij}^{-1}$. Образовав из получившихся элементов q_{ji} матрицу $\mathbf{Q}$, находим $\mathbf{Q} = \widetilde{\mathbf{D}}^T$ и $\mathbf{Q} \times \mathbf{D} = \mathbf{E}$. ♦

Нетрудно показать, что в мультипликативной алгебре количество представлений произвольной функции равно $M_M = k'!(k-1)^{k'}$, а система спектральных функций, задаваемая мономиальной матрицей, фундаментальна, если группа G_M коммутативна.

Пример 5.7. Представим операции сложения и умножения матрицами

$$+ = \begin{bmatrix} 0 & 1 & 2 & 3 \\ 1 & * & * & * \\ 2 & * & * & * \\ 3 & * & * & * \end{bmatrix}, \quad \times = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 2 & 3 & 1 \\ 0 & 3 & 1 & 2 \\ 0 & 1 & 2 & 3 \end{bmatrix},$$

где, как и в примере 5.6, безразличные (неиспользуемые) элементы матрицы операции сложения обозначены *. Для функции из табл. 5.2 имеем

$$\mathbf{D} = \begin{bmatrix} 0 & 0 & 1 \\ 3 & 0 & 0 \\ 0 & 2 & 0 \end{bmatrix}, \quad \widetilde{\mathbf{D}}^T = \begin{bmatrix} 0 & 3 & 0 \\ 0 & 0 & 1 \\ 2 & 0 & 0 \end{bmatrix},$$

$$\mathbf{A} = \begin{bmatrix} 0 & 3 & 0 \\ 0 & 0 & 1 \\ 2 & 0 & 0 \end{bmatrix} \times \begin{bmatrix} 3 & 2 \\ 0 & 2 \\ 1 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 2 \\ 2 & 2 \\ 2 & 1 \end{bmatrix},$$

$$f(x', x'') = \begin{bmatrix} 0 \\ 3 \\ 0 \end{bmatrix}(x') \times \begin{bmatrix} 0 \\ 2 \end{bmatrix}(x'') + \begin{bmatrix} 0 \\ 0 \\ 2 \end{bmatrix}(x') \times \begin{bmatrix} 2 \\ 2 \end{bmatrix}(x'') + \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}(x') \times \begin{bmatrix} 2 \\ 1 \end{bmatrix}(x'').\ \blacklozenge$$

В мультипликативной алгебре с точностью до изоморфизма применим базис, включающий операции сложения (или максимум) и умножения по модулю k, где k – простое число.

5.3.3. Аддитивная алгебра

Пусть $R_A = \langle N_k, +, \times \rangle$, где операции сложения и умножения удовлетворяют условиям алгебры R_L, а сложение таково, что образует коммутативную группу $G_A = \langle N_k, + \rangle$, т.е. для любого элемента $a \in G_A$ существует противоположный ему элемент $-a$, такой что $a + (-a) = a - a = \sigma$, где знаком – обозначена операция, обратная операции сложения.

В дальнейшем нам потребуется следующие определения и лемма.

Определение 5.23. *Порядком элемента a группы G_A называется такое минимальное натуральное число $c_a > 0$, для которого циклическая сумма*

$$c_a \cdot a = \underbrace{a + a + \ldots + a}_{c_a} = \sigma,$$

где σ *– нейтральный (нулевой) элемент группы* G_A.

Для определенности положим $0 \cdot a = \sigma$ и $-\lambda \cdot a = \lambda \cdot (-a)$ для любого $a \in G_A$ и $\lambda \in Z$, Z – множество целых чисел.

Определение 5.24*. Циклическим порядком группы* G_A *будем называть минимальное значение порядков всех ее элементов кроме нейтрального.*

Заметим, что циклический порядок группы G_A равен наименьшему простому числу (кроме единицы), являющемуся делителем ее порядка k, т.е. порядку наименьшей циклической подгруппы группы G_A [148, с. 19].

Лемма 5.5*. В произвольной группе* G_A *уравнение вида* $\lambda \cdot a = b$, *где* $\lambda \in Z$, $a, b \in G_A$, *имеет единственное решение тогда и только тогда, когда циклический порядок группы больше* $|\lambda|$.

Доказательство. Для доказательства леммы достаточно показать, что для всех $a \in N_k$ и при фиксированном $|\lambda|$ элементы $\lambda \cdot a$ различны.

Предположим, что это не так. Тогда имеются такие элементы $a' \neq a''$, что $\lambda \cdot a' = \lambda \cdot a''$. Отсюда следует $\lambda \cdot (a' - a'') = \sigma$. Последнее означает, что элементы a' и a'' совпадают. Приходим к противоречию, которое доказывает лемму. ♦

Определение 5.25*.* ***Биполярной*** *матрицей* $\mathbf{L}_{k'}$ *порядка* k' *называется квадратная матрица, состоящая из нулей* σ, *единиц* τ *и минус единиц* $-\tau$ *алгебры* R_A.

Определение 5.26*. Матрицей,* ***сопряженной*** *биполярной матрице* $\mathbf{L}_{k'}$ *называется такая матрица* $\mathbf{L}^*_{k'}$ *порядка* k', *которая получается из* $\mathbf{L}_{k'}$ *заменой нулей, единиц и минус единиц алгебры* R_A *на нули, единицы и минус единицы кольца целых чисел* $R_Z = \langle Z, +, \times \rangle$.

Рассмотрим в R_A систему спектральных функций, принимающих значения на множестве $T = \{-\tau, \sigma, \tau\}$. Тогда (5.12) преобразуется в систему уравнений

$$f(i, x'') = \sum_{(d_{ij} \neq \sigma)} \pm a_j(x'') \quad (i = \overline{0, k'-1}), \tag{5.15}$$

Очевидно, что эквивалентными преобразованиями уравнений (5.15) в группе G_A являются перестановка строк, замена строки на строку из противоположных элементов, замена строки на ее сумму с произвольной строкой. Отсюда следует, что a_j принадлежат линейному пространству над кольцом целых чисел R_Z, у которого сложение элементов

задается групповой операцией, а умножение элемента a на число λ есть циклическая сумма $\lambda \cdot a$.

Из линейной алгебры известно (формулы Крамара), что решением системы (5.15) являются такие a_j, которые удовлетворяют уравнениям

$$\Delta \cdot a_j = \sum_{i=0}^{k'-1} \Delta_{ij} \cdot f(i, x'') = \Delta_j \quad (j = \overline{0, k'-1}) \tag{5.16}$$

где Δ – определитель матрицы $\mathbf{D}^*$, сопряженной матрице $\mathbf{D}$; Δ_{ij} – алгебраическое дополнение элемента d_{ij}^* матрицы $\mathbf{D}^*$; Δ_j – определитель, получаемый из $\mathbf{D}^*$ заменой ее j-го столбца вектором свободных членов.

Тогда в силу леммы 5.5 система (5.15) имеет решение, если $|\Delta| < c$, где c – циклический порядок группы G_A. Нами доказана

Теорема 5.6. *Произвольная функция f представима в аддитивной алгебре R_A разложением (5.2), если матрица прямого преобразования $\mathbf{D}$ является биполярной и модуль определителя сопряженной ей матрицы $\mathbf{D}^*$ не равен нулю и меньше циклического порядка группы G_A.*

Тогда $\Delta \cdot \mathbf{A} = \overline{\mathbf{D}}^T \cdot \mathbf{F}$, где $\overline{\mathbf{D}}$ – матрица алгебраических дополнений $\mathbf{D}^*$, вычисленная в кольце целых чисел R_Z.

Если G_A циклическая группа, то разложение (5.9) ортогонально, когда Δ является делителем числа, кратного ее порядку k. Тогда уравнение вида $\Delta \cdot a = b$ имеет единственное решение, и на кольце целых чисел может быть вычислена матрица $\mathbf{Q} = \delta \cdot \overline{\mathbf{D}}^T$, где $\delta = qk/\Delta$, q – коэффициент кратности.

Когда $\Delta = 1$, то для любой коммутативной группы G_A имеем $\mathbf{Q} = \overline{\mathbf{D}}^T$. Так как R_A является частным случаем алгебры логики R_L, то для фундаментальности произвольной матрицы из нулей и единиц необходимо потребовать, чтобы $\tau + \tau = \sigma$ или $\tau + \tau = \tau$; а все матрицы, у которых модуль определителя сопряженной матрицы больше нуля и меньше циклического порядка группы, были ортогональны.

Заметим, что операция умножения из R_A применяется только при вычислениях функции. Для получения коэффициентов разложения используется операция циклической суммы, задаваемая группой G_A.

Пример 5.8. Определим операции сложения и умножения алгебры R_A матрицами

$$+ = \begin{bmatrix} 0 & 1 & 2 & 3 \\ 1 & 0 & 3 & 2 \\ 2 & 3 & 0 & 1 \\ 3 & 2 & 1 & 0 \end{bmatrix}, \quad \times = \begin{bmatrix} 0 & 0 & 0 & 0 \\ * & * & * & * \\ * & * & * & * \\ 0 & 1 & 2 & 3 \end{bmatrix},$$

где, как и ранее, неиспользуемые значения операции умножения обозначены *.

Циклический порядок группы G_A равен 2. Для функции из табл. 5.2 получим

$$\mathbf{D} = \begin{bmatrix} 3 & 3 & 0 \\ 3 & 0 & 3 \\ 0 & 3 & 0 \end{bmatrix}, \quad \mathbf{D}^* = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}, \quad \overline{\mathbf{D}}^T = \begin{bmatrix} -1 & 0 & 1 \\ 0 & 0 & -1 \\ 1 & -1 & -1 \end{bmatrix}, \quad \Delta = -1,$$

$$\Delta \cdot \mathbf{A} = \begin{bmatrix} -1 & 0 & 1 \\ 0 & 0 & -1 \\ 1 & -1 & -1 \end{bmatrix} \times \begin{bmatrix} 3 & 2 \\ 0 & 2 \\ 1 & 1 \end{bmatrix} = \begin{bmatrix} 2 & 3 \\ 1 & 1 \\ 2 & 1 \end{bmatrix}, \quad \mathbf{A} = -1 \cdot \begin{bmatrix} 2 & 3 \\ 1 & 1 \\ 2 & 1 \end{bmatrix} = \begin{bmatrix} 2 & 3 \\ 1 & 1 \\ 2 & 1 \end{bmatrix},$$

$$f(x', x'') = \begin{bmatrix} 3 \\ 3 \\ 0 \end{bmatrix}(x') \times \begin{bmatrix} 2 \\ 3 \end{bmatrix}(x'') + \begin{bmatrix} 3 \\ 0 \\ 3 \end{bmatrix}(x') \times \begin{bmatrix} 1 \\ 1 \end{bmatrix}(x'') + \begin{bmatrix} 0 \\ 3 \\ 0 \end{bmatrix}(x') \times \begin{bmatrix} 2 \\ 1 \end{bmatrix}(x'') . \blacklozenge$$

В рамках аддитивной алгебры возможно использование таких образующих операций как сумма по модулю k (поразрядная неэквиваленция) и умножение по модулю k (минимум, поразрядная конъюнкция). Как и ранее, для поразрядных операций k должно быть равно степени 2 (степени простого числа).

5.3.4. Конечное поле

Пусть $R_F = \langle N_k, +, \times \rangle$, где операции сложения и умножения образуют поле на множестве N_k и тем самым удовлетворяют условиям алгебр R_L, R_M и R_A одновременно. Известна следующая теорема [254].

Теорема 5.7*. Произвольная функция f представима в конечном поле R_F разложением (5.2), если и только если матрица прямого преобразования $\mathbf{D}$ состоит из элементов N_k и ее определитель в поле R_F отличен от нуля. Тогда $\mathbf{Q} = \mathbf{D}^{-1}$ и $\mathbf{Q} \times \mathbf{D} = \mathbf{E}$, где $\mathbf{D}^{-1}$ – матрица, обратная $\mathbf{D}$.*

Также известно, что конечное поле R_F, задаваемое с точностью до изоморфизма, существует при $k = p^q$, где p – простое число, q – натуральное число [148, с. 66], а количество обращаемых матриц размерности $k' \times k'$ (количество спектральных представлений функции) в таком поле равно M_F [5, с. 228],

$$M_F = k^{k'(k'-1)/2} \prod_{i=1}^{k'} (k^i - 1) . \qquad (5.17)$$

При $k=2$ из (5.17) получаем количество представлений функции M_A в аддитивной алгебре. Заметим, что в конечном поле любая матрица с элементами из N_k является фундаментальной, а все невырожденные матрицы – ортогональны.

Пример 5.9. На множестве N_4 операции

$$+=\begin{bmatrix}0&1&2&3\\1&0&3&2\\2&3&0&1\\3&2&1&0\end{bmatrix},\quad \times=\begin{bmatrix}0&0&0&0\\0&2&3&1\\0&3&1&2\\0&1&2&3\end{bmatrix}$$

образуют поле. Для ранее определенной функции имеем

$$\mathbf{D}=\begin{bmatrix}1&0&2\\3&1&0\\0&2&3\end{bmatrix},\quad \mathbf{D}^{-1}=\begin{bmatrix}1&1&3\\3&1&2\\2&3&2\end{bmatrix},$$

$$\mathbf{A}=\begin{bmatrix}1&1&3\\3&1&2\\2&3&2\end{bmatrix}\times\begin{bmatrix}3&2\\0&2\\1&1\end{bmatrix}=\begin{bmatrix}0&1\\0&2\\1&0\end{bmatrix},$$

$$f(x',x'')=\begin{bmatrix}1\\3\\0\end{bmatrix}(x')\times\begin{bmatrix}0\\1\end{bmatrix}(x'')+\begin{bmatrix}0\\1\\2\end{bmatrix}(x')\times\begin{bmatrix}0\\2\end{bmatrix}(x'')+\begin{bmatrix}2\\0\\3\end{bmatrix}(x')\times\begin{bmatrix}1\\0\end{bmatrix}(x'').\ \blacklozenge$$

Разложение дискретных функций в конечных полях (полях Галуа) получило широкое распространение на практике [272, 324].

5.3.5. Целостное кольцо

Теорема 5.7 справедлива на целостном кольце R_R (коммутативном кольце с единицей и без делителей нуля)[71]. При конечном числе элементов кольцо R_R изоморфно конечному полю [148, с. 25] и было рассмотрено выше.

Пусть $R_R=\langle Z,+,\cdot\rangle$, где Z – множество целых чисел. Тогда решение системы (5.12) записывается в виде $\dot{a}_j=\Delta\cdot a_j=\Delta_j$ ($j=\overline{0,k'-1}$). По построению $f\in N_{k_f}\subset Z$. Отсюда определитель системы Δ является делителем спектральной суммы и

$$f(x',x'')=\frac{1}{\Delta}\sum_{i=0}^{k'-1}\theta_i(x')\cdot\dot{a}_i(x''). \qquad (5.18)$$

[71] Для существования разложения (5.2) на кольцах требование наличия единицы (нейтрального элемента по умножению) не является обязательным. В общем случае теорема (5.7) справедлива просто на коммутативных кольцах без делителей нуля. Последнее следует из рассуждений, приведенных далее.

Определим разрядность вычислительного устройства, требуемую для вычисления функции f. Пусть заданы произвольные матрицы $\mathbf{D}$, $\mathbf{F}$ и известно, что элементы $\mathbf{D}$ по модулю меньше или равны некоторому числу d, а значения функции меньше $k_f \in N$. Тогда из известной формулы для определителя матрицы находим, что $|\Delta_j|$ не превосходит $(k_f - 1)k'd^{k'-1}$, а текущее значение модуля спектральной суммы при вычислении выражения (5.18) меньше M_R,

$$M_R(d, k', k_f) = \frac{1}{2}k'd(k_f - 1)k'd^{k'-1} = \frac{1}{2}(k_f - 1)\,k'^2 d^{k'}.$$

Таким образом, при p-ичном кодировании данных вычислительному средству требуется оперировать не более чем r разрядами, $r =]\log_p 2N_R[$, где $]x[$ – наименьшее целое, превосходящее x.

Пример 5.10. На кольце целых чисел для функции из примера 5.1 имеем

$$D = \begin{bmatrix} 3 & -2 & 1 \\ 1 & 0 & 2 \\ 4 & -1 & 3 \end{bmatrix},\ \Delta = -5,\ A' = \begin{bmatrix} 2 & 5 & -4 \\ 5 & 5 & -5 \\ -1 & -5 & 2 \end{bmatrix} \times \begin{bmatrix} 3 & 2 \\ 0 & 2 \\ 1 & 1 \end{bmatrix} = \begin{bmatrix} 2 & 10 \\ 10 & 15 \\ -1 & -10 \end{bmatrix},$$

$$f(x', x'') = -\frac{1}{5} \cdot \left(\begin{bmatrix} 3 \\ 1 \\ 4 \end{bmatrix}(x') \cdot \begin{bmatrix} 2 \\ 10 \end{bmatrix}(x'') + \begin{bmatrix} -2 \\ 0 \\ -1 \end{bmatrix}(x') \cdot \begin{bmatrix} 10 \\ 15 \end{bmatrix}(x'') + \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}(x') \cdot \begin{bmatrix} -1 \\ -10 \end{bmatrix}(x'') \right). \blacklozenge$$

На кольце целых чисел известно представление дискретных функций в арифметических формах [347, 355].

5.3.6. Сигнатуры образующих алгебр

Систематизируем результаты, полученные при исследовании образующих алгебр, и найдем свойства их сигнатур.

Элемент a называется ***инвертирующим*** (слева и справа) на множестве $N_a^r \subseteq N_k$, если уравнения $a + x = b$ и $x + a = b$ имеют единственные решения относительно $x \in N_k$ для всех $b \in N_a^r$. Очевидно, что для каждого элемента a может быть получено его инвертирующее множество N_a^r.

Аналогично, элемент a называется ***обращающим*** (слева) на множестве N_a^i, если уравнение $a \times x = b$ имеет единственное решение относительно x для всех $b \in N_a^i$.

Элемент a называется ***вырожденным*** (слева) на множестве N_a, если элемент $x = a \times b$ может быть найден (выражен, определен) при любом наперед неизвестном

$b \in N_a$. Такие элементы b назовем ***свободными***. В частном случае вырожденный элемент a называется ***нейтральным***, если $a \times b = b$, ***константным*** – если $a \times b = c$ и c не зависит от b, ***нулеобразующим*** – если $a \times b = 0$ для всех b, и т.д.

Элемент a называется ***неделимым*** (слева) на множестве N_a, если уравнение $a = b \times x$ не имеет решений относительно x при любых $b \in N_a$.

Пример 5.11. Поясним введенные определения на примере операций, заданных на множестве N_4 матрицами (таблицами Кэли):

$$+ = \begin{bmatrix} 0 & 1 & 2 & 3 \\ 1 & 0 & 3 & 2 \\ 2 & 3 & 0 & 1 \\ 3 & 2 & 1 & 0 \end{bmatrix}, \quad \times = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 2 & 3 \\ 0 & 2 & 3 & 1 \\ 0 & 3 & 1 & 2 \end{bmatrix},$$

Относительно операции сложения все элементы множества N_4 являются инвертирующими слева (справа) на множестве N_4, так как уравнение $a + x = b$ ($x + a = b$) имеет единственное решение для всех $x \in N_4$, что видно из матрицы сложения, которая не имеет повторяющихся элементов в каждой своей строке (столбце).

Относительно операции умножения элементы $a \in \{1, 2, 3\}$ являются обращающими слева (справа) на множестве $\{1, 2, 3\}$, так как уравнение $a \times x = b$ ($x \times a = b$) имеет единственное решение для всех $x \in \{1, 2, 3\}$, что также видно из матрицы операции.

Относительно операции умножения элементы 0 являются вырожденным слева (справа) на множестве N_4, так как $0 \times x = 0$ ($x \times 0 = 0$) для всех $x \in N_4$. В свою очередь элемент 1 вырожден на множестве $\{1, 2, 3\}$ иным образом: $1 \times x = x$ ($x \times 1 = x$). В рассматриваемом примере элемент 0 – нулеобразующий, а 1 – нейтральный.

И наконец, элемент 0 является неделимым слева (справа) на множестве $\{1, 2, 3\}$, так как уравнение $a = b \times x$ ($a = x \times b$) не имеет решения относительно x при любых $b \in \{1, 2, 3\}$. Последнее видно из матрицы умножения, которая не содержит элемента 0 в позициях с номерами строк (столбцов) 1, 2, 3. ♦

В табл. 5.5 определены сигнатуры образующих алгебр, где знаком \ обозначена операция разности множеств, ϕ – нулеобразующий элемент, такой, что $\phi \times a = \sigma$ для всех a. При отсутствии указания на множества, на которых элементы являются инвертирующими, обращающими, вырожденными или неделимыми, подразумевается все множество N_k – носитель образующей алгебры.

Из табл. 5.5 находим, что операция сложения в аддитивной и фундаментальной алгебре образует коммутативную (абелеву) группу на множестве N_k с нейтральным эле-

ментом σ, называемым ***нулем***. Фундаментальная алгебра, в свою очередь, включает поле ($k > 0$) или коммутативное кольцо без делителей нуля ($k = 0$). В фундаментальной алгебре элемент τ становится нейтральным по умножению или ***единицей***, а ϕ вырождается в σ – нейтральный элемент по сложению. В остальных алгебрах элементы ϕ и σ могут различаться.

Таблица 5.5. Сигнатуры образующих алгебр

Алгебра	Операция сложения	Операция умножения
Логики R_L	$\sigma + \sigma = \sigma$, σ – инвертирующий	ϕ – вырожденный ($\phi \times a = \sigma$), τ – обращающий
Мультипликативная R_M		ϕ – вырожденный ($\phi \times a = \sigma$), $N_k \setminus \{\sigma\}$ – обращающие
Аддитивная R_A	N_k – инвертирующие, ассоциативность, коммутативность	ϕ – вырожденный, τ – обращающий
Фундаментальная R_F		ϕ – неделимый на $N_k \setminus \{\sigma\}$, ассоциативность, коммутативность, дистрибутивность по сложению

В аддитивной алгебре возможно различное вырождение элементов ϕ и τ, но в любом случае требуется, чтобы хотя бы одно вырождение существенным образом зависело от свободного элемента. Частным случаем аддитивной алгебры являются униполярная и биполярная алгебры. ***Униполярная*** алгебра имеет традиционное вырождение элементов: $\phi \times x = \sigma$, $\tau \times x = x$. В ***биполярной*** алгебре элемент ϕ вырождается иначе: $\phi \times x = -x$, где $-x$ – элемент, противоположный x, такой что $-x + x = \sigma$.

Для существования матрицы $\mathbf{Q}$ и использования матричного аппарата обратного дискретного преобразования (5.11) на операции образующих алгебр накладывают дополнительные ограничения. В алгебре логики элемент σ должен быть равен ϕ и являться нейтральным по сложению, а элемент τ – нейтральным по умножению. В мультипликативной алгебре умножение должно иметь обратную операцию на множестве $N_k \setminus \sigma$. В аддитивной и фундаментальной алгебрах в дополнительных требованиях нет необходимости.

Отношения между алгебрами показаны на рис. 5.18.

5.4. Функциональная полнота

По своей сути разложение (5.9) позволяет свести вычисление произвольной m-функции f к вычислению функций θ_i и a_i меньшей сложности (с меньшей длиной вектора), а теоремы разложения определяют условия, при которых базис $\Omega = \{+, \times, \{\theta_i\}, < a >\}$ обладает функциональной полнотой, где $\{\theta_i\}$ – фиксированный набор k'-функций, а $< a >$ – класс, включающий все k''-функции. В предельном случае, когда $k' = m$, получаем спектральное представление, у которого класс $< a >$ состоит из констант образующей алгебры.

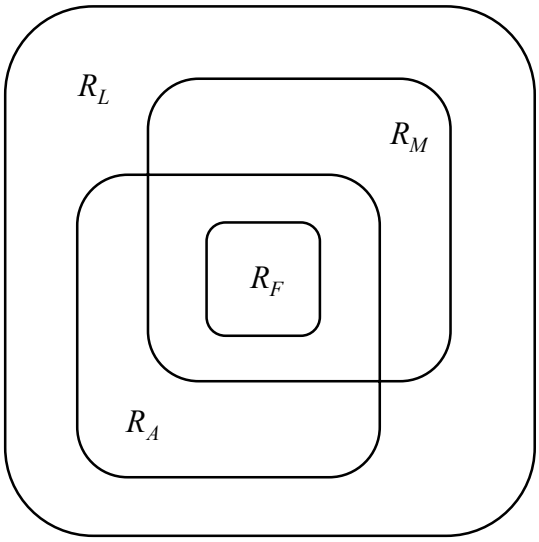

Рис. 5.18. Классы образующих алгебр

Как показано ранее, существуют четыре основных типа образующих алгебр, которые ввиду произвольности частичных функций $< a >$ порождают четыре различных класса частичных функций $\{\theta_i\}$ соответственно (табл. 5.6).[72]

[72] В связи с тем, что теоремы разложения определяют только необходимые условия представления функций, может оказаться, что в образующих алгебрах, кроме фундаментальной, существуют системы частичных функций, которые не являются, например, унимодальными двузначными, но могут быть использованы для разложений в алгебре логики. Это возможно по причине принадлежности одного типа образующих алгебра другому их типу, например, любое конечное поле является одновременно алгеброй логики, мультипликативной и аддитивной алгеброй. Так как теоремы разложения описывают не конкретную алгебру, а целый их класс, то и классы функций будем рассматривать обобщенно, с точностью до специфицированных свойств специальных элементов. Таким образом, будем предполагать, что значения операций, помеченных знаком *, никогда не используются. Последнее означает некоторое необходимое ограничение классов функций, приведенных в классификации.

Таблица 5.6. Образующие алгебры

Класс функций	Двузначные	Многозначные
Унимодальные	Алгебры логики	Мультипликативные алгебры
Мультимодальные	Аддитивные алгебры	Фундаментальные алгебры

Унимодальными называются функции, принимающие значения, отличные от нулеобразующего элемента ϕ, только в одной точке области определения. Если эти значения одинаковы, то функции унимодальные двузначные, в противном случае – унимодальные многозначные.

К ***мультимодальным*** относятся функции, имеющие произвольные значения в любой точке области определения. Если множество значений включает только два элемента, то такие функции называются двузначными, если более двух – многозначными.

В свою очередь мультимодальные функции разделяются на два подкласса: ***униполярные мультимодальные*** и ***биполярные мультимодальные***, соответствующих униполярному и биполярному вырождению элементов. Биполярные функции используются для декомпозиции в биполярной аддитивной алгебре и кольце целых чисел, а униполярные – в униполярной аддитивной алгебре и конечном поле.

Действительно, для разрешимости системы уравнений (5.12) на коэффициенты d_{ji} должны быть наложены определенные ограничения. В алгебре логики матрица прямого преобразования $\mathbf{D}$ является ***матрицей перестановок***, в каждой строке и столбце которой имеется только один элемент, отличный от ϕ и равный τ. В мультипликативной алгебре – ***мономиальной матрицей***, содержащей в каждой строке и в каждом столбце по одному элементу, отличному от ϕ. В аддитивной алгебре – ***логической матрицей***, состоящей из ϕ и τ $(-\tau)$ с ненулевым определителем, по модулю меньшим циклического порядка группы по сложению. В фундаментальных алгебрах: ***обращаемой матрицей*** – в конечном поле и ***матрицей с ненулевым определителем*** – в коммутативном кольце без делителей нуля.

Классы частичных функций, порождаемые образующими алгебрами, находятся в отношении включения друг в друга (рис. 5.19). Самый обширный класс S_F представлен мультимодальными многозначными функциями и включает в себя два пересекающиеся класса: класс мультимодальных двузначных функций S_A и класс унимодальных многозначных функций S_M. Пересечение последних образует класс унимодальных двузначных функций S_L.

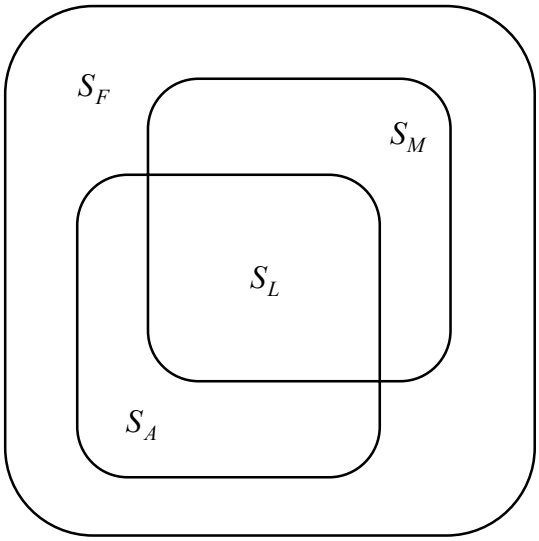

Рис. 5.19. Классы спектральных функций

Типичные представители распространенных на практике систем приведены в табл. 5.7. В классе унимодальных используются следующие функции: дизъюнктивные [265], конъюнктивные [340], литеральные [255] и интервальные [314]. Из мультимодальных применяются функции Жегалкина [75], Рида-Малера [318, 328], теоретико-числовые [225, 327] и полиномиальные [159]. Бимодальные функции представлены функциями Радемахера [326], Уолша [362], Хаара [293] и Виленкина-Крестенсона [42, 271].

Таблица 5.7. Системы частичных функций

Класс функций	Двузначные	Многозначные
Унимодальные	Дизъюнктивная Конъюнктивная	Литеральная Интервальная
Мультимодальные униполярные	Жегалкина Рида-Маллера	Теоретико-числовые Полиномиальные
Мультимодальные биполярные	Радемахера Уолша	Хаара Виленкина-Крестенсона

Для демонстрации прикладных аспектов теории алгебраической декомпозиции в Приложении 4 рассмотрены задачи нахождения различных классов частичных функций, которые имеют эффективное (компактное) представление в различных образующих алгебрах, а также реализация образующих алгебр на современных вычислительных средствах.

5.5. Синтез формул

Традиционно синтез формул частичных функций не производится, в разложение подставляются уже готовые выражения, известные для небольшого числа базисов. Для

каждой такой системы дается символьное обозначение функций и приводятся аналитические выражения, определяющие эти функции через другую известную систему или в виде формул в некотором базисе операций. В последнем случае используются операции, которые позволяют наиболее просто выразить спектральные функции в аналитической форме. Иногда для упрощения выражений вводятся специальные операции, которые не использовались ранее.

Покажем на примере синтез формульного представления функции, который осуществим в аддитивной алгебре.

Пример 5.12. Пусть операции сложения и умножения R_A заданы матрицами

$$+ = \begin{bmatrix} 0 & 1 & 2 & 3 \\ 1 & 2 & 3 & 0 \\ 2 & 3 & 0 & 1 \\ 3 & 0 & 1 & 2 \end{bmatrix}, \quad \times = \begin{bmatrix} 0 & 1 & 2 & 3 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix},$$

где + – сумма по модулю 4, образующая абелеву группу на множестве N_4 с нулем $\sigma = 0$, а $\times$ – операция логического сдвига ($a \times b$ есть сдвиг b на a двоичных разрядов влево) с нулеобразующим элементом $\phi = 3$ ($\phi \times a = 0$ для всех $a \in N_4$) и обращающим элементом $\tau = 0$ ($\tau \times a = a$ для всех $a \in N_4$). Циклический порядок группы равен 2, так как $a + a = \sigma$ при $a = 2$.

Рассмотрим функцию из примера 5.3 при разделении переменных $X' = \{x_1, x_3\}$ и $X'' = \{x_0, x_2\}$. Запишем ее разложение по некоторой системе спектральных функций

$$f(X) = \theta_0(X') \times a_0(X'') + \theta_1(X') \times a_1(X'') + \theta_2(X') \times a_2(X'') + \theta_3(X') \times a_3(X'').$$

Зададим систему спектральных функций логической матрицей $\mathbf{D}$ и найдем в кольце целых чисел определитель Δ и алгебраические дополнения $\overline{\mathbf{D}}$ сопряженной матрицы $\widetilde{\mathbf{D}}$, получаемой из $\mathbf{D}$ заменой τ на единицу, а ϕ на нуль кольца целых чисел:

$$\mathbf{D} = \begin{bmatrix} \tau & \phi & \phi & \phi \\ \tau & \tau & \phi & \phi \\ \tau & \phi & \tau & \phi \\ \tau & \phi & \phi & \tau \end{bmatrix}, \quad \widetilde{\mathbf{D}} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{bmatrix}, \quad \Delta = 1, \quad \overline{\mathbf{D}} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ -1 & 1 & 0 & 0 \\ -1 & 0 & 1 & 0 \\ -1 & 0 & 0 & 1 \end{bmatrix}.$$

Циклический порядок группы больше Δ. Следовательно, система функций $\mathbf{D}$ обеспечивает функциональную полноту базиса в алгебре R_A. Тогда

$$\Delta \cdot \mathbf{A} = \overline{\mathbf{D}} \cdot \mathbf{F} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ -1 & 1 & 0 & 0 \\ -1 & 0 & 1 & 0 \\ -1 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 3 & 2 & 0 & 2 & 2 & 0 \\ 2 & 1 & 3 & 1 & 1 & 3 \\ 1 & 0 & 2 & 0 & 0 & 2 \\ 3 & 2 & 0 & 2 & 2 & 0 \end{bmatrix},$$

где $\cdot$ – операция ***циклической суммы***, определяемая как $n \cdot a = a + a + \ldots + a$ (n раз), причем $(-n) \cdot a = n \cdot (-a)$ и $0 \cdot a = \sigma$, $\mathbf{A}$ – матрица коэффициентов, $\mathbf{F}$ – двумерная таблица функции (табл. 5.3).

Умножая матрицы $\overline{\mathbf{D}}$ и $\mathbf{F}$ относительно операции сложения и циклической суммы, находим $\Delta \cdot \mathbf{A}$, а затем и $\mathbf{A}$ путем решения системы уравнений $\Delta \cdot \mathbf{A} = \mathbf{B}$, где $\mathbf{B} = \overline{\mathbf{D}} \cdot \mathbf{F}$,

$$\Delta \cdot \mathbf{A} = \begin{bmatrix} 3 & 2 & 0 & 2 & 2 & 0 \\ 3 & 3 & 3 & 3 & 3 & 3 \\ 2 & 2 & 2 & 2 & 2 & 2 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}, \quad \mathbf{A} = \begin{bmatrix} 3 & 2 & 0 & 2 & 2 & 0 \\ 3 & 3 & 3 & 3 & 3 & 3 \\ 2 & 2 & 2 & 2 & 2 & 2 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}.$$

В рассматриваемом случае $\Delta \cdot \mathbf{A} = \mathbf{A}$, т.к. $\Delta = 1$. В итоге имеем

$$f(X) = \begin{bmatrix} \tau \\ \tau \\ \tau \\ \tau \end{bmatrix} (X') \times \begin{bmatrix} 3 \\ 2 \\ 0 \\ 2 \\ 2 \\ 0 \end{bmatrix} (X'') + \begin{bmatrix} \phi \\ \tau \\ \phi \\ \phi \end{bmatrix} (X') \times \begin{bmatrix} 3 \\ 3 \\ 3 \\ 3 \\ 3 \\ 3 \end{bmatrix} (X'') + \begin{bmatrix} \phi \\ \phi \\ \tau \\ \phi \end{bmatrix} (X') \times \begin{bmatrix} 2 \\ 2 \\ 2 \\ 2 \\ 2 \\ 2 \end{bmatrix} (X'') + \begin{bmatrix} \phi \\ \phi \\ \phi \\ \tau \end{bmatrix} (X') \times \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} (X''),$$

и после тождественных преобразований получим

$$f(X) = \begin{bmatrix} 3 \\ 2 \\ 0 \\ 2 \\ 2 \\ 0 \end{bmatrix} (X'') + \begin{bmatrix} 3 \\ 0 \\ 3 \\ 3 \end{bmatrix} (X') \times 3 + \begin{bmatrix} 3 \\ 3 \\ 0 \\ 3 \end{bmatrix} (X') \times 2 + \begin{bmatrix} 3 \\ 3 \\ 3 \\ 0 \end{bmatrix} (X') \times 0 = \begin{bmatrix} 3 \\ 2 \\ 0 \\ 2 \\ 2 \\ 0 \end{bmatrix} (x_0, x_2) + \begin{bmatrix} 0 \\ 3 \\ 2 \\ 0 \end{bmatrix} (x_1, x_3).$$

Теперь представим функции, присутствующие в выражении и зависящие от двух переменных в виде операций $\circ_0$ и $\circ_1$,

$$f(X) = x_0 \begin{bmatrix} 3 & 0 & 2 \\ 2 & 2 & 0 \end{bmatrix} x_2 + x_1 \begin{bmatrix} 0 & 2 \\ 3 & 0 \end{bmatrix} x_3 = x_0 \circ_0 x_2 + x_1 \circ_1 x_3 . \blacklozenge$$

Из примера 5.12 видно, что при алгебраической декомпозиции по виду таблицы функции можно оценить эффективность ее формульного представления. Представление более компактно, если при прочих равных условиях матрица $\mathbf{F}$ содержит большее число одинаковых или линейно-зависимых строк. Очевидно, определение линейной зависимости осуществляться в каждой из образующих алгебр по-разному.

5.6. Заключительные замечания

Основные теоретические результаты, представленные в настоящей главе, опубликованы в работах [59, 63]. Алгебраическая декомпозиция дискретных функций, выпол-

няемая в мультипликативной и аддитивной алгебре, и ее применение в цифровой обработке сигналов и логической обработке данных, рассмотрены в работах [60, 61]. Асимптотические оценки для сложности представления дискретных функций даны в работах [55, 56]. Эмпирические формулы для конструирования операций конечных полей описаны в работе [58].

Синтезу полиномиальных и неполиномиальных представлений дискретных функций в различных образующих алгебрах посвящены работы [59, 358]. Различные расширения спектральных базисов исследованы в работах [50, 70, 71]. Обобщенная методика синтеза полиномиальных форм рассмотрена в работах [51, 54]. Синтезу частных случаев полиномиальных представлений (мультипликативных и булевых форм) посвящены работы [53, 357].

Применению различных образующих алгебр и синтезируемых для них спектральных функций в цифровой обработке сигналов посвящена работа [358]. Новые возможности обработки сигналов в аддитивной алгебре описаны в работах [60, 359]. Синтез Хаароподобных систем сигналов на основе неполиномиальной аналитической конструкции исследован в работе [62]. Использование мультипликативной алгебры при логической обработке данных показано в работах [72, 73, 74].

Выводы к Главе 5

1. Рассмотрена проблема функциональной декомпозиции дискретных функций и найдена классификация методов декомпозиции на основе анализа регулярности синтезируемых формул. Показано, что известные методы декомпозиции строятся на основе разделения переменных, причем в общих методах решение о разделении переменных принимается на каждом шаге декомпозиции, в то время как в частных случаях используется их фиксированное разделение.

2. Обобщена теория алгебраической декомпозиции, при которой разделение переменных выполняется на каждом шаге декомпозиции, а объединение фрагментов синтезируемой формулы осуществляется в специальных образующих алгебрах. Последнее позволяет снизить трудоемкость синтеза путем использования на каждом шаге различных образующих алгебр и выполнять независимую от других шагов минимизацию синтезируемой формулы.

3. Приведена классификация образующих алгебр по типам используемых в них операций. Исследованы свойства таких алгебр и найдены требования, налагаемые на сигнатуру и специальные элементы носителя. Приведены необходимые и достаточные условия существования разложений в каждой из алгебр.

4. Исходя из требований функциональной полноты, исследованы свойства подформул, объединяемых при алгебраической декомпозиции в единое выражение, на основе чего дана классификация частичных функций, которые могут быть использованы при синтезе и отличаются различными требованиями к вычислительным средствам, используемым для реализации дискретной обработки данных.

5. Предложена методика многоступенчатого синтеза формул, выполняемая при алгебраической декомпозиции, на каждом шаге которой исходя из свойств декомпозируемой функции принимается решение о разделении переменных, выборе образующей алгебры и вида частичных функций. Последнее позволяет, при прочих равных условиях, учесть нетривиальные свойства декомпозируемой функции и получить ее более компактное (эффективное) формульное представление.

Глава 6.
Алгебраический синтез

Исследование сложности представления булевых функций предпринято в работах [141, 155, 249, 256]. Основной их результат: почти все функции реализуются со сложностью, близкой к максимальной. Для максимального количества операций, необходимых для реализации произвольной булевой функции, найдены асимптотические оценки и показано существование декомпозиций, им удовлетворяющих. Однако выдвинуто предположение о том, что нахождение минимальных формул сопряжено с полным или почти полным перебором возможных решений, что практически осуществимо только для функций небольшой размерности.

Таким образом, следует отличать синтез формульных представлений дискретных функций по методикам, удовлетворяющим асимптотическим оценкам, от их минимизации, которая выполняется иным образом (возможно, по другим методикам). Для закрепления этого факта в терминологическом аппарате определим и будем различать два понятия: алгебраический синтез и минимизация.

Под ***алгебраическим синтезом*** будем понимать получение такого представления дискретных функций, которое содержит количество операций, не превосходящее максимально необходимого для подавляющего большинства функций той же размерности. В свою очередь ***минимизацию*** определим как нахождение такой формы представления функции, которая включает наименьшее из возможных число операций. Очевидно, что количество операций, получаемых при минимизации, не будет превосходить количество операций, получаемых при алгебраическом синтезе. Заметим, что для большинства функций результат их минимизации и алгебраического синтеза по количеству операций будет совпадать.

Так как найдены только асимптотические оценки максимальной сложности представления функций, то и эффективность методики синтеза может будет установлена при анализе поведения используемых в ней алгоритмов в асимптотической области. Однако, если удастся определить максимально необходимое число операций для представления произвольной функции при конечной ее размерности, то появится возможность определения эффективности алгоритмов и при конечной размерности задачи.

Настоящая глава посвящена алгебраическому синтезу формульных представлений дискретных функций и исследованию эффективности получаемых при этом математических моделей дискретной обработки данных как в асимптотической области, так и при конечной размерности задачи.

6.1. Содержательная постановка задачи

Для сравнения различных декомпозиционных схем введем количественные характеристики, определяющие эффективность получаемых представлений. Для этого рассмотрим разложение произвольной m-функции значности $k_f \le k$, где k – значность аналитической конструкции синтезируемых формул, при ее алгебраической декомпозиции по некоторой системе k'-функций θ_i ($i = \overline{0, k'-1}$),

$$f(X) = \sum_{i=0}^{k'-1} \theta_i(X') \times a_i(X''), \tag{6.1}$$

где множество переменных X разделено на два непересекающихся множества X' и X'' с числом элементов n' и n''. После исключения из (6.1) выражений вида $\phi \times a_i(X'') = \sigma$ получим

$$f(X) = \sum_{i=0}^{M-1} \theta_i(X') \times a_i(X''), \tag{6.2}$$

где M – количество слагаемых, $M \le k'$, a_i – коэффициенты разложения (k''-функции).

Сложность разложения функции определим количеством слагаемых M. Очевидно, чем меньше сложность разложения, тем более эффективна декомпозиция. Под ***сложностью представления функции*** L будем понимать количество операций, необходимых для вычисления функции по ее формуле.

Рассмотрим разложение (6.2), записанное в матричном виде, $\mathbf{D} \times \mathbf{A} = \mathbf{T}$, где размерность матрицы $\mathbf{D}$ равна $k' \times k'$, а матриц $\mathbf{A}$ и $\mathbf{T}$ – $k' \times k''$ (рис. 6.1).

$$\begin{bmatrix} \tau & \phi & \phi & \phi & \phi & \phi \\ \phi & \tau & \phi & \phi & \phi & \phi \\ \phi & \phi & \tau & \phi & \phi & \phi \\ \phi & \phi & \phi & \tau & \phi & \phi \\ * & * & * & * & \tau & \phi \\ * & * & * & * & \phi & \tau \end{bmatrix} \times \begin{bmatrix} + & + & + & + \\ + & + & + & + \\ + & + & + & + \\ + & + & + & + \\ \phi & \phi & \phi & \phi \\ \phi & \phi & \phi & \phi \end{bmatrix} = \begin{bmatrix} + & + & + & + \\ + & + & + & + \\ + & + & + & + \\ + & + & + & + \\ \sim & \sim & \sim & \sim \\ \sim & \sim & \sim & \sim \end{bmatrix}$$

Рис. 6.1. Решающая декомпозиция: τ – единица алгебры; ϕ – нулеобразующий элемент; $*$ – свободные элементы; $+$ – элементы линейно-независимых строк; $\sim$ – элементы линейно-зависимых строк

Видно, что количество членов разложения, необходимое для представления произвольной функции, не может превосходить k''. Последнее следует из фундаментальности частичных функций θ_i. Действительно, выбрав в качестве базисных k'' линейно независимых строк матрицы функции, только $k' - k''$ строк можно представить в виде линейных комбинаций базисных, а соответствующие коэффициенты обнулить.

Очевидно, сложность разложения функции не будет зависеть от выбора линейно-независимых строк, и найти разложение с меньшей сложностью, чем число линейно-независимых строк в матрице функции, не представляется возможным. В свою очередь, перестановка переменных внутри множеств не влияет на линейную зависимость строк (столбцов), следовательно, разделение переменных следует выполнять с точностью до их порядка.

Из приведенных рассуждений видно, что алгебраическая декомпозиция близка к преобразованию Карунена-Лоэва [8, с. 182], в котором разложение осуществляется по собственным векторам ковариационной матрицы функции, и, тем самым, обеспечивается наилучшее (оптимальное) приближение функции в среднеквадратическом смысле. Однако в отличие от преобразования Карунена-Лоэва, на практике возникает необходимость в минимизации не только сложности разложения, но и сложности представления функции. Поэтому поставим задачу синтеза формульных представлений с наименьшей сложностью. Заметим, что при разложении по Карунену-Лоэву такая задача не ставится.

При оптимальном разложении по Карунену-Лоэву спектральные функции вычисляются с некоторой степенью свободы, а количество этих функций равно числу ненулевых собственных значений ковариационной матрицы. В итоге преобразование Карунена-Лоэва гарантирует получение разложения функции с наименьшей сложностью. Воспользуемся оставшимися степенями свободы в задании спектральных функций путем выбора такого их подмножества, которое при минимальной сложности разложения обеспечивает и минимальную сложность представления функции.

Для выражения синтезируемых частичных функций исследуем предельно общие аналитические конструкции формул и выявим те их свойства, которые существенны при синтезе. Для этого рассмотрим аналитические конструкции, задаваемые в базисе произвольных операций и имеющие произвольный порядок вхождения переменных и произвольную расстановку скобок.

Основные результаты, представленные в настоящей главе, опубликованы в работах [63, 69].

6.2. Аналитические конструкции

При декомпозиции дискретных функций используются различные аналитические конструкции, задающие строение синтезируемых формул. Основное их назначение – упорядочить процесс синтеза формул, сделать его регулярным и предсказуемым.

Наибольшее применение получили такие аналитические конструкции формул, как дизъюнктивная и конъюнктивная, а также их предельный случай – совершенная дизъюнк-

тивная и совершенная конъюнктивная нормальные формы [340]. Известны также аналитические конструкции формул, представленные полиномом Жегалкина [75], формой Рида-Малера [318, 328], секвенциальной формой [102], арифметическим и арифметико-логическим полиномом [71, 161], литеральной [255] и интервальной [314] формой, формами Радемахера [326], Уолша [362], Хаара [293], Виленкина-Крестенсона [42, 271] и др.

По своей сути каждая аналитическая конструкция является некоторой декомпозиционной схемой, в рамках которой ищется формульное представление декомпозируемой функции. Многообразие аналитических конструкций является следствием попыток синтеза эффективных представлений для ограниченных классов функций. Очевидно, что для каждой из известных форм могут быть найдены функции, имеющие в этих формах наиболее компактное представление, так же как и функции, представляемые с наибольшей сложностью. Поставим целью обобщить известные аналитические конструкции формул и синтезировать конструкцию, имеющую максимальную выразительную способность.

6.2.1. Классы формул

Рассмотрим предельно общую аналитическую конструкцию, состоящую из всех возможных унарных и бинарных операций с произвольными значностями как своих операндов, так и результата операции и произвольной расстановкой скобок, задающих порядок вычисления. Будем предполагать, что переменные, входящие в аналитическую конструкцию, также могут иметь произвольную значность. Ввиду того, что основной целью синтеза является аппаратурная или программно-аппаратная реализация дискретных функций по получаемым при синтезе формульным представлениям, ограничим диапазон возможных значностей переменных и операций некоторым фиксированным числом k, которое назовем ***значностью аналитической конструкции***.

Ограничение значности аналитической конструкции диктуется практическими соображения, связанными с возможностью выполнения многозначных операций и доступа к многозначным данным (переменным) на вычислительном средстве, для которого синтезируется формульное представление.

Очевидно, чем больше значность аналитической конструкции, тем при прочих равных условиях, может быть получено более компактное представление. Однако значность операций должна согласовываться со значностью переменных, которая зачастую является фиксированной и определяется из содержательных представлений о решаемой задаче.

Произвольную формулу будем характеризовать количеством переменных и количеством операций, из которых она состоит. Это количественные характеристики формулы. К качественным характеристикам отнесем наличие скобок, изменяющих естественный

порядок вычисления, и повторную входимость переменных, определяющую частоту обращений к одной и той же переменной при вычислениях.

По входимости переменных все формулы будем разделять на ***повторные***, у которых одна и та же переменная может присутствовать более одного раза, и ***бесповторные***, – когда каждая переменная встречается один раз или не встречается ни разу.

Под ***бесскобочными*** формулами будем понимать такие формулы, в которых операции над переменными выполняются слева направо в естественном порядке их записи при приоритете унарных операций над бинарными. В таких формулах скобки могут быть опущены.

В ***скобочных*** формулах порядок вычислений отличается от естественного. Это приводит к необходимости сохранять промежуточные результаты вычисления всех или части подформул, заключенных в скобки.

Таким образом, следует различать четыре класса формул: повторные скобочные, повторные бесскобочные, бесповторные скобочные и бесповторные бесскобочные. Рассмотрим особенности применения при алгебраической декомпозиции формул из различных классов.

6.2.2. Повторная входимость

Необходимость разделения формул на повторные и бесповторные вызвана тем, что от этого существенным образом зависит сложность их программной или аппаратной реализации.

При аппаратной реализации бесповторных формул трассировка данных от места хранения переменной до места ее обработки осуществляется к единственному входу логического элемента. В то время как для повторных формул такая трассировка требуется для многих элементов.

При программной реализации повторные формулы приводят к многократному обращению к ячейкам памяти, где хранятся переменные. Вычисление же бесповторной формулы приводит к единственной выборке данных из входных ячеек.

При алгебраической декомпозиции имеется возможность выбора как различных образующих алгебр, так и частичных функций с несущественной зависимостью от части переменных. Последнее позволяет синтезировать формулы как с повторными вхождениями переменных, так и с отсутствием вхождения всех или части переменных.

Следовательно, вхождение в аналитическую конструкцию одной и той же переменной более одного раза приводит к принципиальной неотличимости синтезируемых формул от выражений, получаемых в результате алгебраической декомпозиции. Если же разрешить повторные вхождения переменных, то для представления частичных функций в

виде формул необходим такой же или более общий аппарат функциональной декомпозиции. Это приводит к невозможности завершить декомпозицию, если, конечно, формулы не заданы заранее.

Таким образом, повторные вхождения переменных в аналитическую конструкцию формул не только не улучшают выразительные свойства алгебраической декомпозиции, но и приводят к некоторой избыточности. Не нарушая общности, потребуем отсутствия повторных вхождений одной и той же переменной в аналитические конструкции частичных функций.

6.2.3. Расстановка скобок

При реализации скобочных формул порядок вычислений отличается от естественного. Аппаратная реализация бесскобочных формул осуществляется в виде каскадных (линейных) схем. В то время, как реализация скобочных формул приводит к схеме общего вида. В свою очередь вычисление скобочных формул требует использования промежуточных ячеек памяти. В противоположность этому, бесскобочная форма может быть вычислена сразу, без сохранения промежуточных результатов.

Исследуем возможность использования бесповторных формул с различной расстановкой скобок при алгебраической декомпозиции. Рассмотрим наиболее общую бесповторную аналитическую конструкцию, заданную с точностью до расстановки скобок и порядка вхождения переменных,

$$\theta(X) = \triangleleft_0\ x_0 \circ_0 \triangleleft_1\ x_1 \circ_1 \dots \circ_{n-2} \triangleleft_{n-1}\ x_{n-1}, \qquad (6.3)$$

где $\triangleleft_j$ – произвольные унарные или степенные операции, $\circ_j$ – произвольные бинарные или соединительные операции, x_{i_j} – переменные из множества X.

Выполним тождественные структурные преобразования конструкции формулы (6.3). С учетом произвольной расстановки скобок возможны три типа вхождения переменных в формулу:

$$\theta(X) = \dots\ (\triangleleft_s\ x_s \circ \triangleleft_t\ x_t)\ \dots\ (\triangleleft_s\ x_s \circ (\dots))\ \dots\ ((\dots) \circ \triangleleft_t\ x_t) \dots .$$

Рассмотрим первый тип, как наиболее общий, включающий в виде частных случаев два оставшихся. На рис. 6.2 показано вычисление фрагмента формулы $\triangleleft_s\ x_s \circ \triangleleft_t\ x_t$, откуда видно, что значение переменной x_s определяет строку операции $\triangleleft_s$, из которой извлекается значение, задающее строку матрицы $\circ$, а столбец последней получается аналогично и задается переменной x_t и матрицей ее операции $\triangleleft_t$.

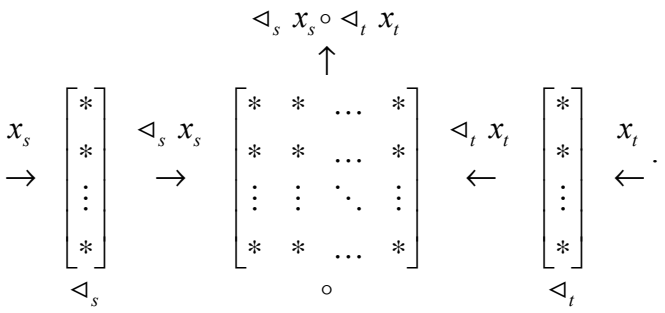

Рис. 6.2. Исходный фрагмент бесповторной формулы

Из рис. 6.2 видно, что рассматриваемый фрагмент формулы может быть эквивалентно преобразован в фрагмент $x_s \bullet x_t$, содержащий некоторую бинарную операцию $\bullet$ (рис. 6.3). Такое преобразование осуществляется путем копирования и перестановки строк матрицы $\circ$, задаваемое матрицами унарных операций $\triangleleft_s$ и $\triangleleft_t$.

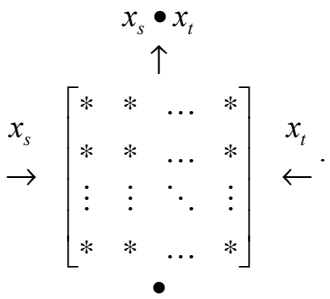

Рис. 6.3. Результат тождественных преобразований

Если окажется, что операция $\bullet$ содержит одинаковые строки (столбцы), то она несущественным образом зависит от левого (правого) операнда и этот операнд может быть исключен из формулы. В итоге тождественных преобразований получаем минимальную формулу функции

$$\theta(X) = x_{i_0} \bullet_1 x_{i_2} \bullet_2 \ldots \bullet_{n'-2} x_{i_{n'-1}},$$

заданную с точностью до расстановки скобок, и из которой исключены унарные операции и несущественные переменные.

Пример 6.1. Пусть дискретная функция задана формулой

$$f(x_0, x_1) = \triangleleft_0 x_0 \circ_0 (\triangleleft_1 x_1 \circ_1 \triangleleft_2 x_2),$$

где унарные и бинарная операции определены матрицами

$$\triangleleft_0 = \begin{bmatrix} 1 \\ 2 \\ 0 \end{bmatrix}, \ \triangleleft_1 = \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}, \ \triangleleft_2 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \ \circ_0 = \begin{bmatrix} 2 & 0 & 0 \\ 0 & 2 & 1 \\ 1 & 1 & 2 \end{bmatrix}, \ \circ_1 = \begin{bmatrix} 0 & 1 \\ 1 & 0 \\ 2 & 0 \end{bmatrix}.$$

Выполним тождественные преобразования этой формулы и получим ее минимальную (каноническую) форму. Исходная конструкция формулы имеет вид

$$x_0\begin{bmatrix}1\\2\\0\end{bmatrix}\begin{bmatrix}2&0&0\\0&2&1\\1&1&2\end{bmatrix}\left(x_1\begin{bmatrix}1\\0\\1\end{bmatrix}\begin{bmatrix}0&1\\1&0\\2&0\end{bmatrix}\begin{bmatrix}1\\0\end{bmatrix}x_2\right),$$

откуда находим, что переменные x_0 и x_1 имеют значность 3, а переменная x_2 – значность 2.

Унарная операция $\triangleleft_1$ эквивалентна такому преобразованию бинарной операции $\circ_1$, при котором на первое месте стоит строка 1, на втором месте – строка 0, а третье место копируется строка 1 исходной матрицы. Вторая унарная операция эквивалентна перестановке столбцов. В результате тождественных преобразований получаем

$$x_0\begin{bmatrix}1\\2\\0\end{bmatrix}\begin{bmatrix}2&0&0\\0&2&1\\1&1&2\end{bmatrix}\left(x_1\begin{bmatrix}1&0\\0&1\\1&0\end{bmatrix}\begin{bmatrix}1\\0\end{bmatrix}x_2\right),$$

$$x_0\begin{bmatrix}1\\2\\0\end{bmatrix}\begin{bmatrix}2&0&0\\0&2&1\\1&1&2\end{bmatrix}\left(x_1\begin{bmatrix}0&1\\1&0\\0&1\end{bmatrix}x_2\right).$$

Далее анализируем вторую бинарную операцию (в круглых скобках) и находим ее значность, равную 2. Следовательно, в первой бинарной используются только первые два столбца,

$$x_0\begin{bmatrix}1\\2\\0\end{bmatrix}\begin{bmatrix}2&0\\0&2\\1&1\end{bmatrix}\left(x_1\begin{bmatrix}0&1\\1&0\\0&1\end{bmatrix}x_2\right).$$

В свою очередь унарная операция $\triangleleft_0$ вызывает перестановку строк (120) первой бинарной операции,

$$x_0\begin{bmatrix}0&2\\1&1\\2&0\end{bmatrix}\left(x_1\begin{bmatrix}0&1\\1&0\\0&1\end{bmatrix}x_2\right).$$

В итоге имеем $f(x_0,x_1)=x_0\bullet_0(x_1\bullet_1 x_2)$, где

$$\bullet_0=\begin{bmatrix}0&2\\1&1\\2&0\end{bmatrix},\ \bullet_1=\begin{bmatrix}0&1\\1&0\\0&1\end{bmatrix}.$$

В результате получена минимальная формула исходной функции в классе произвольных бинарных операций. ♦

Таким образом, конструкция (6.3) может быть подвергнута тождественным преобразованиям, в результате которых получается конструкция вида

$$\theta(X) = x_{i_0} \circ_0 x_{i_1} \circ_1 x_{i_2} \circ_2 \ldots \circ_{n'-2} x_{i_{n'-1}}, \qquad (6.4)$$

также заданная с точностью до расстановки скобок, но уже не содержащая унарных операций и несущественных переменных ($n' \le n$). Сравним выразительные возможности скобочных и бесскобочных аналитических конструкций вида (6.4).

6.2.4. Бесскобочная конструкция

Подсчитаем количество различных функций N_θ, которое порождается конструкцией (6.4) при естественном порядке ее вычисления. Будем предполагать, что левый операнд соединительных операций $\circ_j$ имеет значность предыдущей операции, а правый операнд – значность соответствующей переменной k_{i_j}.

Для начала рассмотрим всевозможные бинарные операции от переменных со значностями k_0, k_1 и найдем количество порождаемых ими функций $N_o(k,k_0,k_1)$ значности k. Общее число функций, очевидно, равно $k^{k_0 k_1}$ и

$$\sum_{j=0}^{k} C_k^j N_o(j,k_0,k_1) = k^{k_0 k_1},$$

где множитель C_k^j – число сочетаний из k элементов по j, который учитывает функции значности j, принимающие значения не на всем множестве N_k, а на различных его подмножествах из j элементов. Отсюда получаем рекуррентное выражение для вычисления $N_o(k,k_0,k_1)$,

$$N_o(0,k_0,k_1) = 0, \quad N_o(k,k_0,k_1) = k^{k_0 k_1} - \sum_{j=0}^{k-1} C_k^j N_o(j,k_0,k_1). \qquad (6.5)$$

Аналогично получаем $N_o^{\rightarrow}(k,k_0,k_1)$ – количество функций значности k, порождаемых всевозможными бинарными операциями при их ***более чем существенной зависимости*** от левой переменной, т.е. когда матрицы операций не содержат одинаковых строк,

$$N_o^{\rightarrow}(1,k_0,k_1) = 0, \quad N_o^{\rightarrow}(k,k_0,k_1) = \prod_{i=0}^{k_0-1} (k^{k_1} - i) - \sum_{j=1}^{k-1} C_k^j N_o^{\rightarrow}(j,k_0,k_1). \qquad (6.6)$$

Результаты вычислений количества операций, выполненные при различных значностях операций k_f и переменных k, приведены в табл. 6.1

Таблица 6.1. Количество бинарных операций

k	k_f	$N_o(k_f,k,k)$	$N_o^{\rightarrow}(k_f,k,k)$
2	1	1	0
	2	14	12
3	1	1	0
	2	510	336
	3	18150	16542
4	1	1	0
	2	65534	43680
	3	42850116	39798720
	4	4123173624	4035566400

Теперь для подсчета количества функций N_{θ}^{*}, порождаемых конструкцией (6.4), имеем систему рекуррентных уравнений

$$\begin{cases} N_{\theta}(k_f,k_0,k_1)=N_o(k_f,k_0,k_1), \\ N_{\theta}(k_f,k_0,k_1,\ldots,k_t)=\sum\limits_{i=1}^{k_f}\dfrac{C_{k_f}^{i}}{i!}N_{\theta}(i,k_0,k_1,\ldots,k_{t-1})N_o^{\rightarrow}(i,i,k_t), \\ N_{\theta}^{*}(k,k_0,k_1,\ldots,k_{n-1})=\sum\limits_{j=1}^{k}C_k^j N_{\theta}(j,k_0,k_1,\ldots,k_{n-1}), \end{cases} \tag{6.7}$$

которая получена следующим образом.

Количество различных функций значности k_f двух переменных равно $N_o(k_f,k_0,k_1)$. Количество функций значности k_f от t переменных получаем как сумму количеств функций от $t-1$ переменной значности i, умноженное на количество операций $N_o^{\rightarrow}(i,i,k_t)$, где i пробегает значения от 1 до k_f. Все функции от $t-1$ переменной порождают различные функции t переменных. Однако каждая функция от t переменных повторяется $i!$ раз, т.к. имеется $i!$ различных операций (по числу перестановок множества из i элементов), порождающих от различных функций $t-1$ переменной одинаковые функции t переменных. Множитель $C_{k_f}^{i}$, как и ранее, учитывает функции значности i, принимающие значения не на всем множестве из k_f элементов, а на различных его подмножествах из i элементов.

В табл. 6.2 приведены результаты вычисления в соответствии с (6.7) количества функций значности k_f от n переменных, порождаемых аналитической конструкцией бесповторных бесскобочных формул в алгебре с числом элементов k, где значности переменных k_j равны значности алгебры.

Таблица 6.2. Количество бесповторных бесскобочных формул

k	k_f	$n=2$	$n=3$	$n=4$	$n=5$
2	2	14	84	504	3024
3	2	510	14280	399840	11195520
	3	18150	50082390	138078348750	380682041090310
4	2	65534	7864080	943689600	113242752000
	3	42850116	3584007294480	$2{,}99766372941\times10^{17}$	$2{,}50724594331\times10^{22}$
	4	4123173624	$6{,}93320208414\times10^{17}$	$1{,}16580823595\times10^{26}$	$1{,}96029022745\times10^{34}$

Теперь произведем оценку количества бесповторных бесскобочных формул, порождающих различные функции. Из (6.5) и (6.6) следует

$$k^{k_0(k_1-1)} < N_o^{\rightarrow}(k,k_0,k_1) < N_o(k,k_0,k_1) < k^{k_0k_1},$$

откуда из (6.7) получаем оценку для N_θ^* при $n>2$,

$$k^{k_0k_1+(k-1)\sum_{i=2}^{n-1}k_i} < N_\theta^*(k,k_0,k_1,\ldots,k_{n-1}) < k^{k_0k_1+k\sum_{i=2}^{n-1}k_i}. \qquad (6.8)$$

Оценка (6.8) допускает достаточно простую содержательную интерпретацию. Верхняя граница степеней свободы аналитической конструкции, приведенной на рис. 6.4, меньше суммарной площади операций. Нижняя граница определяется при более чем существенной зависимости операций от правого операнда, что эквивалентно вычеркиванию одной строки у каждой операции, кроме нулевой.

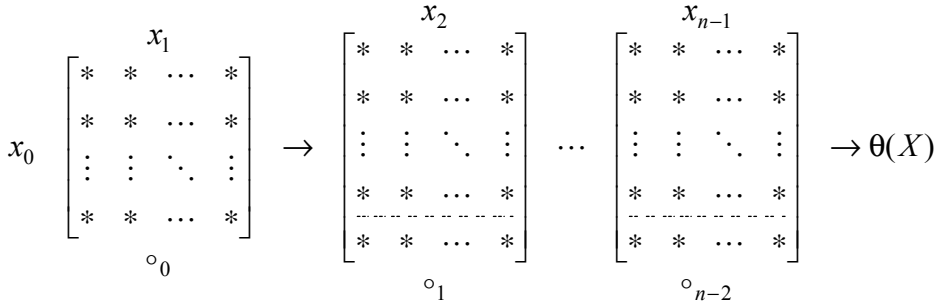

Рис. 6.4. Бесскобочная аналитическая конструкция

Заметим одно немаловажное обстоятельство. Количество порождаемых векторов (m-функций) не зависит от порядка переменных, а определяется только их значностями. В итоге констатируем, что порождающая способность аналитической конструкции определяется значностями переменных и не зависит от их порядка, кроме первых двух переменных, которые задают размерность матрицы первой операции.

Пример 6.2. Пусть формула для функции $\theta(x_0,x_1,x_2)$ задана ее бесскобочной аналитической конструкцией,

$$x_0 \quad \begin{matrix} & x_1 & \\ \begin{bmatrix} 1 & 0 & 2 \\ 2 & 1 & 1 \end{bmatrix} \\ & \circ_0 & \end{matrix} \quad \begin{matrix} & x_2 & \\ \begin{bmatrix} 2 & 1 \\ 0 & 2 \\ 1 & 0 \end{bmatrix} \\ & \circ_1 & \end{matrix} \to \theta(X).$$

Из анализа конструкции получаем формулу функции $\theta(X) = x_0 \circ_0 x_1 \circ_1 x_2$, значность функции $k_f = 3$ и значности переменных K, $K = \{2,3,2\}$. Вычислим вектор значений функции $\mathbf{G}$ и восстановим ее таблицу истинности (табл. 6.3).

Таким образом, в результате вычисления получен вектор значений функции $\mathbf{G} = [012010201202]$ при векторе значностей переменных $\mathbf{K} = [232]$. ♦

Таблица 6.3. Таблица истинности функции

x_0	x_1	x_2	θ
0	0	0	0
1	0	0	1
0	1	0	2
1	1	0	0
0	2	0	1
1	2	0	0
0	0	1	2
1	0	1	0
0	1	1	1
1	1	1	2
0	2	1	0
1	2	1	2

6.2.5. Порождающая способность

Для упрощения формул вместо неравенств (6.8) будем использовать равенство

$$N^*_\theta(k,k_0,k_1,\ldots,k_{n-1}) = k^{k_0k_1+(k-\alpha)\sum_{i=2}^{n-1}k_i}, \tag{6.9}$$

α – некоторый параметр, который назовем ***порождающей способностью*** аналитической конструкции. Вычисление α осуществим по формуле

$$\alpha = k - \frac{\log_k N^*_\theta - k_0k_1}{\sum_{i=2}^{n-1}k_i}, \tag{6.10}$$

где k_i – значности переменных, заданные в порядке их вхождения в формулу, N^*_θ – количество функций, порождаемых аналитической конструкцией бесскобочных формул, найденное путем решения системы рекуррентных уравнений (6.7).

На рис. 6.5 показана зависимость порождающей способности α от количества переменных n. Вычисления выполнены при одинаковых значностях образующей алгебры, функций и их переменных.

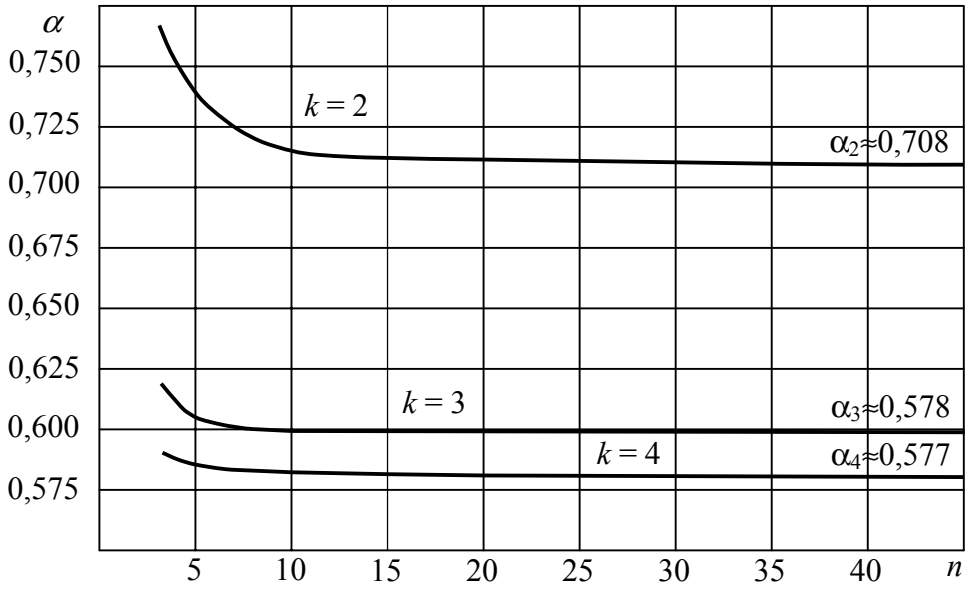

Рис. 6.5. Порождающая способность бесповторных формул

Заметим, что при смешанной значности переменных, когда их средняя значность меньше значности образующей алгебры, возможны значения порождающей способности, большие единицы. В табл. 6.4 приведены расчетные значения α, выполненные для пятизначной алгебры при различном числе переменных n и различных средних значностях переменных $\bar{k}$. Из анализа таблицы следует, что при равенстве значностей переменных значности образующей алгебры, $k = \bar{k}$, порождающая способность аналитической конструкции бесповторных формул слабо зависит от числа переменных n.

Таблица 6.4. Порождающая способность α при различных средних значностях переменных $\bar{k}$ в образующей алгебре значности $k = 5$

$\bar{k} \setminus n$	5	6	7	8	9
2	2,253	2,133	2,601	2,013	1,979
3	1,103	1,088	1,079	1,073	1,069
4	0,763	0,761	0,760	0,759	0,758
5	0,598	0,598	0,598	0,598	0,598

С ростом n порождающая способность аналитической конструкции α асимптотически стремится к некоторой величине α_k, которую назовем ***предельной порождающей способностью***. Из (6.7) находим

$$\alpha_k = k - \frac{1}{k}\log_k \prod_{i=0}^{k-1} \frac{k^k - i}{i+1}. \qquad (6.11)$$

Результаты вычисления α_k при различных k с точностью до трех знаков после запятой приведены в табл. 6.5, а зависимость α_k от k показана на рис. 6.6.

Таблица 6.5. Предельная порождающая способность α_k

k	2	3	4	5	6	7	8	9
α_k	0,708	0,578	0,577	0,595	0,612	0,625	0,637	0,647

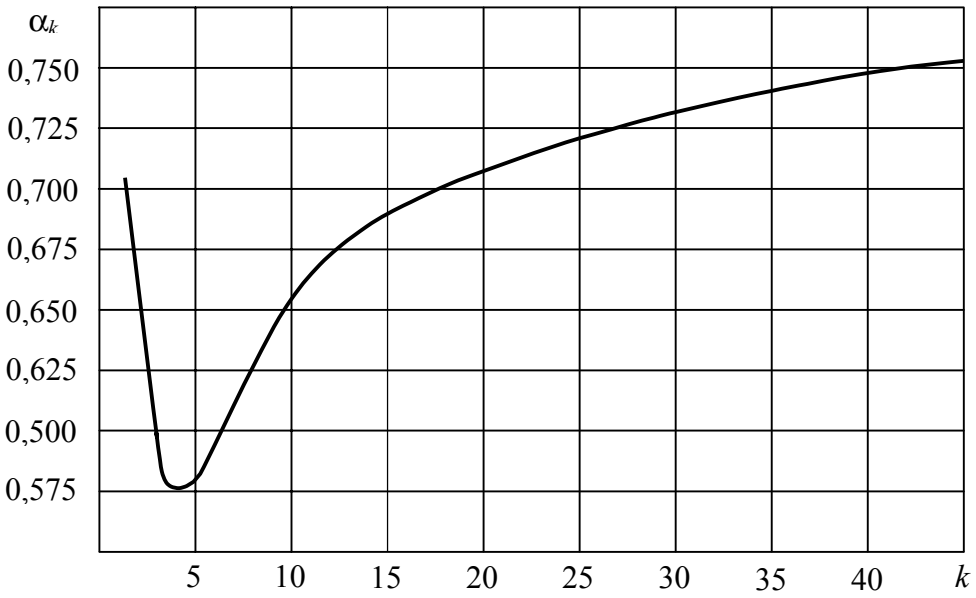

Рис. 6.6. Предельная порождающая способность

Из (6.11) находим, что с ростом k предельная порождающая способность α_k бесповторной аналитической конструкции стремится к единице, а на рис. 6.6 показано, что наименьшее ее значение достигается при $k = 4$.

6.2.6. Скобочная конструкция

Если конструкция (6.4) является скобочной, то для подсчета количества функций на каждом уровне вложенности скобок применим следующее представление,

$$\theta(X) = \theta_0(X_0) \circ_0 \theta_1(X_1) \circ_1 \ldots \circ_{t-2} \theta_{s-1}(X_{s-1}),$$

где θ_i – некоторые подформулы, заключенные в скобки и зависящие от переменных из подмножества X_i со значностями K_i (рис. 6.7).

В этом случае имеем

$$\begin{cases} N_\theta(k_f, K_0, \ldots, K_t) = \sum\limits_{i=1}^{k} \dfrac{C_{k_f}^i}{i!i!} N_\theta(i, K_0, \ldots, K_{t-1}) N_o^{\leftrightarrow}(i,i,i) N_\theta(i, K_t), \\ N_\theta^*(k, K_0, \ldots, K_{s-1}) = \sum\limits_{j=1}^{k} C_k^j N_\theta(j, K_0, \ldots, K_{s-1}), \end{cases}$$

где $N_\theta(k, K_i)$ – количество функций, порождаемых подформулой θ_i, а $N_o^{\leftrightarrow}(k, k_0, k_1)$ – количество операций значности k, порождающих различные функции при их более чем существенной зависимости от двух переменных, т.е. когда матрицы операций не содержат как одинаковых строк, так и одинаковых столбцов.

$$\theta_0(X_0) \underset{\circ_0}{\begin{bmatrix} * & * & \cdots & * \\ * & * & \cdots & * \\ \vdots & \vdots & \ddots & \vdots \\ * & * & \cdots & * \end{bmatrix}} \rightarrow \underset{\theta_1(X_1)}{\underset{\circ_1}{\begin{bmatrix} * & * & \cdots & * \\ * & * & \cdots & * \\ \vdots & \vdots & \ddots & \vdots \\ * & * & \cdots & * \end{bmatrix}}} \cdots \underset{\theta_{s-1}(X_{s-1})}{\underset{\circ_{s-2}}{\begin{bmatrix} * & * & \cdots & * \\ * & * & \cdots & * \\ \vdots & \vdots & \ddots & \vdots \\ * & * & \cdots & * \end{bmatrix}}} \rightarrow \theta(X)$$

Рис. 6.7. Скобочная аналитическая конструкция

Очевидно, $N_o^{\leftrightarrow}(k, k_0, k_1) < N_o^{\rightarrow}(k, k_0, k_1)$. Тогда

$$N_\theta^*(k, K_0, K_1, \ldots, K_{s-1}) < N_\theta^*(k, k_0, k_1, \ldots, k_{n-1}).$$

Отсюда делаем вывод, что скобочная аналитическая конструкция является менее выразительной, т.к. по сравнению с бесскобочной порождает меньшее число функций. Но произвольная расстановка скобок может породить функции, не существующие при естественном порядке вычисления. Однако представить в виде формул большее число m-функций, чем это определено степенями свободы аналитической конструкции, не представляется возможным,

$$N_\theta^*(k, k_0, k_1, \ldots, k_{n-1}) < k^{k_0 k_1 + k \sum\limits_{i=2}^{n-1} k_i}.$$

6.2.7. Порядок вхождения переменных

Для задания дискретной функции помимо вектора значений **F** также необходим и вектор значностей переменных **K**, причем один и тот же вектор **F** может быть использован для задания различных функций, отличающихся как значностями, так и порядком переменных. Исследуем, как изменится порождающая способность аналитической конструкции формул при изменении порядка вхождения переменных.

Изменение порядка переменных приводит к перестановке оснований позиционной системы счисления при пересчете значений переменных в индекс элемента вектора функ-

ции. Это эквивалентно некоторому переупорядочиванию его элементов. Но далеко не каждое переупорядочивание порождает новую функцию.

Рассмотрим аналитическую конструкцию при одинаковых значностях переменных. Перестановка переменных, кроме первых двух, преобразует одну конструкцию в другую, порождающую функции из того же множества. Интерпретируя этот результат при произвольных значностях переменных, имеем

$$N_{\theta}^{*}(k,k_0,k_1,\ldots,k_{n-1}) < k^{k\sum_{i=0}^{n-1}k_i},$$

где для учета перестановки с участием первых двух переменных оценка несколько увеличена и сделана симметричной. Отсюда делаем вывод, что использование аналитической конструкции формул с изменяемым порядком вхождения переменных не имеет существенного значения для выразительных качеств алгебраической декомпозиции.

Таким образом, при синтезе формул, осуществляемом при алгебраической декомпозиции, будем использовать бесповторную бесскобочную аналитическую конструкцию формул с фиксированным порядком вхождения переменных.

6.2.8. Определение порождающей способности

Точная оценка порождающей способности бесповторных скобочных аналитических конструкций может быть получена по методике, описанной в 6.2.4. Для этого достаточно подсчитать количество матриц размерности $k_0 \times k_1$, у которых отсутствует повторение не только строк, но и столбцов. Далее следует задать расстановку скобок и для заданной расстановки скобок определить количество функций, порождаемых рассматриваемой аналитической конструкцией. Ввиду того, что при другой расстановке скобок могут быть порождены функции из первого набора, то точный подсчет количества функций, порождаемых всеми возможными скобочными аналитическими конструкциями, представляет собой достаточно трудную задачу. Однако в решении такой задачи нет необходимости, так как в 6.2.6 показано, что количество таких функций заведомо меньше, чем количество функций, порождаемых бесскобочной аналитической конструкцией.

В результате исследования поведения порождающей способности бесповторной бесскобочной аналитической конструкции формул, заданных при различных значностях переменных, установлен замечательный факт, заключающийся в том, что наибольшую порождающую способность имеет аналитическая конструкция, заданная при значностях переменных, равных трем и четырем. Последнее может служить теоретическим обоснованием того, что при создании многозначных логических элементов следует ограничиться только алфавитами из трех или четырех знаков. Реализация логических элементов с большим числом состояний будет менее эффективна при прочих равных условиях.

6.3. Спектральный синтез формул

Синтез формул осуществим путем разложения функции в спектральном базисе с заранее неизвестным формульным представлением,

$$f(X) = \sum_{i=0}^{M-1} \theta_i(X) \times a_i \,, \tag{6.12}$$

где a_i – константы образующей алгебры.

Поставим задачу поиска при спектральной декомпозиции не только коэффициентов, но и формул для системы спектральных функций, причем таких, которые обеспечат наименьшую сложность представления.

6.3.1. Реализуемость спектральных функций

Предварительно исследуем множество спектральных функций в различных образующих алгебрах и определим возможность их порождения формулами с бесповторной аналитической конструкцией.

В алгебре логики и в мультипликативной алгебре количество спектральных функций равно m и $m(k-1)^m$, причем все они легко представляются бесповторными бесскобочными формулами.

В униполярной и биполярной аддитивной алгебре общее количество спектральных функций уже равно $2^m - 1$ и $3^m - 1$, а это превосходит комбинаторные возможности аналитической конструкции. В этом случае требуется порождать только функции значности 2 или 3. В итоге из (6.7) при $k = 3$ получаем

$$N_\theta(k, k_0, k_1, \ldots, k_{n-1}) = N_3(k_0, k_1, \ldots, k_{n-1}) \,,$$

а из (6.8) следует

$$N_3(k_0, k_1, \ldots, k_{n-1}) < 3^{k_0 k_1 + k \sum_{i=2}^{n-1} k_i} \le 3^{\prod_{i=0}^{n-1} k_i} = 3^m \,.$$

В фундаментальной алгебре имеем два случая. При конечной значности образующей алгебры декомпозиция производится в конечном поле, а когда алгебра определена на счетном множестве элементов – в коммутативном кольце без делителей нуля. Соответственно в конечном поле число спектральных функций конечно, в то время как в кольце имеется счетное их число. Т.к. все реальные вычисления производятся при ограниченной разрядности чисел, то практически значимым является использование некоторого конечного набора из $k^m - 1$ спектральных функций, где k ограничивается разрядностью вычислительного средства. Тогда из (6.8) получаем

$$N^*_\theta(k, k_0, k_1, \ldots, k_{n-1}) < k^{k_0 k_1 + k \sum_{i=2}^{n-1} k_i} \le k^{\prod_{i=0}^{n-1} k_i} = k^m \,.$$

Таким образом, полное использование комбинаторных возможностей бесповторной аналитической конструкции возможно только в аддитивной и фундаментальной алгебре. В других алгебрах аналитическая конструкция является избыточной и не влияет на эффективность синтеза.

Особенности декомпозиции функций в алгебре логики и в мультипликативной алгебре подробно рассмотрены в [74]. Далее будем предполагать, что декомпозиция производится в аддитивной или фундаментальной алгебре. В аддитивной алгебре число частичных функций будем оценивать неравенствами

$$k^{(k_0k_1+(k-1)\sum_{i=2}^{n-1}k_i)\log_k 2} < N_\theta^*(2,k_0,k_1,\ldots,k_{n-1}) < k^{(k_0k_1+k\sum_{i=2}^{n-1}k_i)\log_k 2}, \qquad (6.13)$$

а в фундаментальной алгебре будем использовать только те частичные функции, которые представимы при заданной разрядности вычислительного средства, т.е. конечное их число.

6.3.2. Классы эквивалентности формул

Как показано ранее, существует небольшое число бесповторных формул N_θ^*, представляющих различные функции. В свою очередь функции, порождаемые бесповторной аналитической конструкцией, образуют множество линейно-независимых классов. Внутри класса функции преобразуются друг в друга путем умножения на ненулевые константы. Следовательно, в один класс включаются функции, которые могут быть преобразованы друг в друга путем умножения элементов матрицы последней операции на ненулевую константу. В фундаментальной алгебре таких констант $k-1$, а в каждом классе – ровно по $k-1$ функций. В аддитивной алгебре имеется всего одна константа.

Известно [5], что количество невырожденных матриц размерности $m\times m$ и значности k равно N_χ,

$$N_\chi = k^{\frac{m(m-1)}{2}}\prod_{i=1}^{m}(k^i-1).$$

Используем этот результат для оценки количества линейно независимых классов бесповторных функций. Для этого найдем количество матриц N_δ размерности $m\times m$ с различающимися столбцами,

$$N_\delta = \prod_{i=1}^{m}k^m - i, \qquad (6.14)$$

Формула (6.14) получена следующим образом. В качестве первого столбца может быть выбран любой из k^m-1 возможных векторов длины m, исключая нулевой. На месте

второго столбца может быть размещен любой из $k^m - 2$ оставшихся векторов, и т.д., на место столбца с номером m может быть размещен один из $k^m - m$ векторов.

Выразим теперь долю невырожденных матриц χ от найденного общего числа матриц с различающимися столбцами,

$$\chi(m) = \frac{N_\chi}{N_\delta} = k^{\frac{m(m-1)}{2}} \prod_{i=1}^{m} \frac{k^i - 1}{k^m - i}.$$

Результаты расчета значения χ при различных k показаны на рис. 6.8. Заметим, что χ слабо зависит от размерности матрицы m.

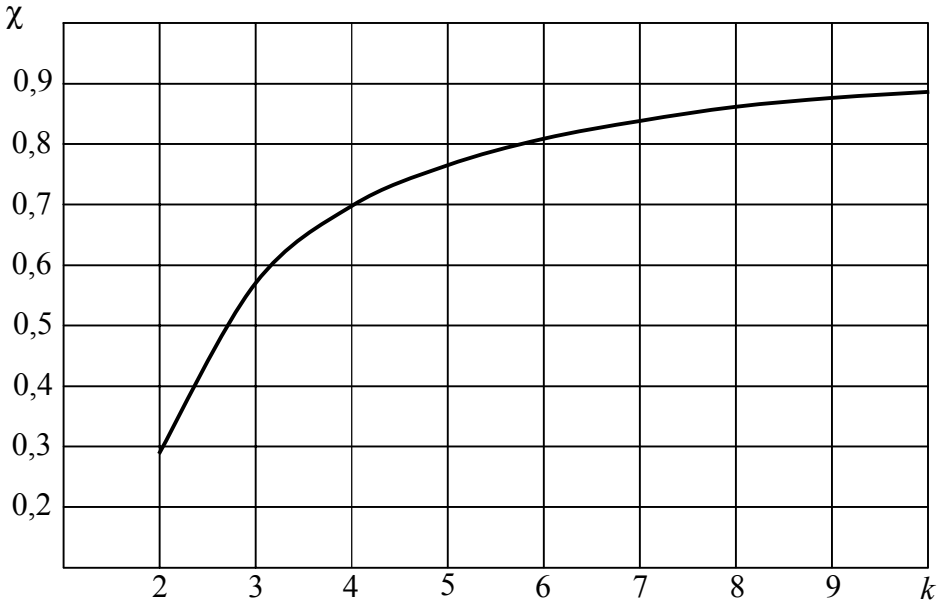

Рис. 6.8. Доля невырожденных матриц

Таким образом, нами оценено число линейно независимых бесповторных бесскобочных формул N_c,

$$N_c = \frac{\chi N_\theta}{k-1}.$$

Очевидно, в разложении (6.12) различные комбинации функций из линейно-независимых классов порождают различные функции. Используем это свойство для определения сложности спектрального разложения.

6.3.3. Сложность спектрального разложения

Множество бесповторных бесскобочных формул в образующей алгебре разбивается на классы эквивалентности, включающие функции с точностью до ненулевого множителя. Следовательно, в спектральном разложении (6.12) операция умножения может быть опущена

$$f(X) = \sum_{i=0}^{M-1} \theta_i(X). \qquad (6.15)$$

Пусть в разложении (6.15) при $M < m$ различные комбинации спектральных функций θ_i, взятые из различных классов, порождают различные функции (рис. 6.9). Для спектральных разложений (рис. 6.9) количество ненулевых слагаемых M определим из возможности разложения произвольной функции единственным образом.

$$\left[\begin{array}{ccccc:c} \theta_0 & \theta_1 & \theta_2 & \theta_3 & \theta_4 & \phi \\ \theta_0 & \theta_1 & \theta_2 & \theta_3 & \theta_4 & \phi \\ \theta_0 & \theta_1 & \theta_2 & \theta_3 & \theta_4 & \phi \\ \theta_0 & \theta_1 & \theta_2 & \theta_3 & \theta_4 & \phi \\ \theta_0 & \theta_1 & \theta_2 & \theta_3 & \theta_4 & \phi \\ \theta_0 & \theta_1 & \theta_2 & \theta_3 & \theta_4 & \phi \end{array}\right] \times \left[\begin{array}{c} + \\ + \\ + \\ + \\ + \\ \underline{\phi} \end{array}\right] = \left[\begin{array}{c} * \\ * \\ * \\ * \\ * \\ * \end{array}\right]$$

Рис. 6.9. Спектральное разложение: θ_i – элементы линейно-независимых столбцов; ϕ – нулеобразующий элемент; $*$ – произвольные элементы; $+$ – связанные ненулевые элементы

В этом случае для порождения всех функций необходимо, чтобы

$$k^m = \sum_{i=1}^{M} C_{N_c}^i (k-1)^i, \qquad (6.16)$$

где $N_c = \chi N_\theta/(k-1)$ – число классов функций, задаваемых с точностью до множителя и порождаемых канонической аналитической конструкцией бесповторных бесскобочных формул, $C_{N_c}^i$ – количество неупорядоченных выборок i функций из N_c, χ – доля невырожденных матриц размерности $m \times m$ от общего числа матриц с различающимися столбцами.

Так как $m < N_c$ и $2M < N_c$, то из (6.16) получаем

$$2^{-M}(k-1)^M N_c^M < k^m < (k-1)^M N_c^M,$$

а после логарифмирования с учетом (6.8) и (6.13) имеем оценку максимального числа слагаемых M_a и M_f при спектральной декомпозиции в аддитивной и фундаментальной алгебре соответственно:

$$M_a = \frac{1}{\log_k 2} \frac{1}{k-\alpha} \frac{\prod_{i=0}^{n-1} k_i}{\sum_{i=0}^{n-1} k_i - \beta}, \qquad M_f = \frac{1}{k-\alpha} \frac{\prod_{i=0}^{n-1} k_i}{\sum_{i=0}^{n-1} k_i - \beta}, \qquad (6.17)$$

где β – поправочный коэффициент, необходимый для учета начальной порождающей способности аналитической конструкции,

$$\beta = k_0 + k_1 - \frac{k_0 k_1 + \log_k \chi}{k - \alpha}, \tag{6.18}$$

где χ – доля невырожденных матриц размерности $m \times m$ от общего числа матриц с различающимися столбцами,

$$\log_k \chi = \frac{m(m-1)}{2} + \sum_{i=1}^{m} \log_k \frac{k^i - 1}{k^m - i}. \tag{6.19}$$

Результаты вычисления $\log_k \chi$ по формуле (6.19) при различных k показаны на рис. 6.10. Величина $\log_k \chi$ слабо зависит от m и не влияет на значение начальной порождающей способности β, выраженной формулой (6.18), так как изменение $\log_k \chi$ при различных m несоизмеримо с произведением значностей переменных k_0 и k_1.

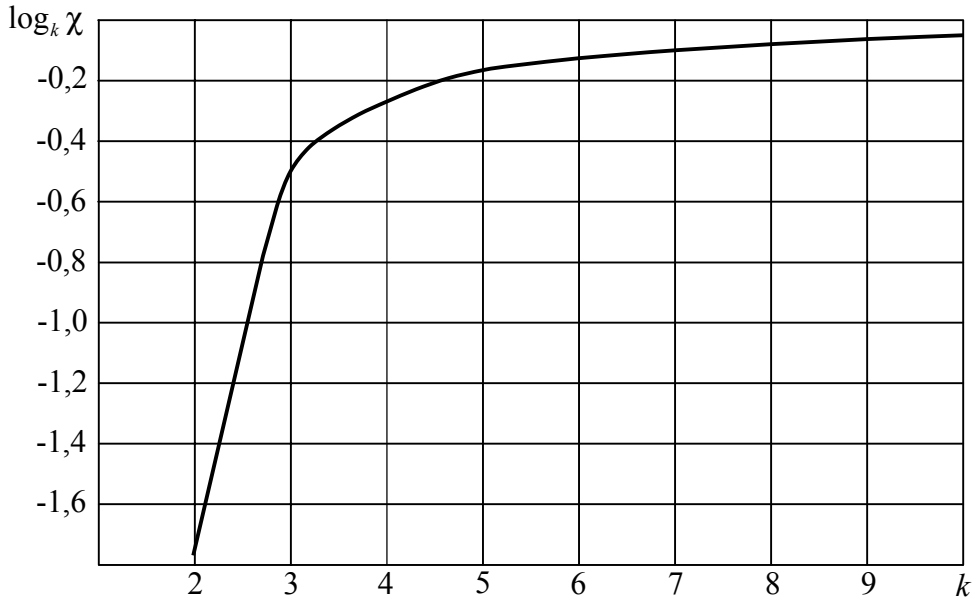

Рис. 6.10. Логарифм доли невырожденных матриц

Для оценки M будем использовать граничные значения α, а для точных вычислений – некоторое его промежуточное значение. Будем предполагать, что $2k > k_i$, тогда $\beta > 0$. При одинаковых значностях переменных, равных k, формула (6.17) принимает более простой вид

$$M = \frac{k^{n-1}}{(k-\alpha)(n-1)+\alpha}.$$

В частности, когда $n = 2$, имеем $M = 1$, что согласуется с возможностью представления произвольной функции двух переменных формулой с одной операцией.

Из (6.17) следует, что сложность спектрального разложения определяется с точностью до значностей первых двух переменных, а наиболее компактные разложения полу-

чаются в фундаментальных алгебрах. Поэтому разложения функций далее будем осуществлять в конечных полях или коммутативных кольцах без делителей нуля.

6.3.4. Сложность спектрального представления

Для максимального количества операций L, необходимых для реализации произвольной функции от n переменных со значностями k_0, k_1, …, k_{n-1}, из (6.17) следует

$$L(k,k_0,k_1,\ldots,k_{n-1}) = \frac{n}{k-\alpha}\frac{\prod_{i=0}^{n-1} k_i}{\sum_{i=0}^{n-1} k_i - \beta},$$

а после очевидных преобразований имеем

$$L(k,k_0,k_1,\ldots,k_{n-1}) = \frac{C}{2}m, \tag{6.20}$$

где C – константа, характеризующая аналитическую конструкцию формул,

$$C = \frac{1}{k-\alpha}\frac{2}{\frac{1}{n}(\sum_{i=0}^{n-1} k_i - \beta)}. \tag{6.21}$$

Оценка (6.20) минимальна, когда значности переменных стремятся к их среднему геометрическому

$$k_j \to \sqrt[n]{\prod_{i=0}^{n-1} k_i} = \sqrt[n]{m} \quad (j = \overline{0,n-1}).$$

В этом случае имеем

$$L(n) = \frac{1}{k-\alpha}\frac{k^{n-1}}{1-\frac{1-\alpha}{n}}.$$

6.3.5. Точность спектральных оценок

Ввиду того, что декомпозируемая функция была заранее неизвестна, нами получены оценки для максимальной сложности разложения и сложности представления функции. Точные значения перечисленных характеристик имеют смысл только для зафиксированной функции. Нами также найдены точные максимальные значения для M и L. Для получения последних в формулу (6.16) подставляется точное число линейно-независимых классов N_c, откуда находится M, а через него и точное значение L.

Нахождение точных характеристик связано с определенными вычислительными трудностями. Однако имеется возможность определить область применимости полученных оценочных значений. Для этого найдем долю функций δ от общего их числа, имеющих меньшую сложность, чем максимально возможная. Как и ранее, получим как точные, так и оценочные значения δ.

Очевидно функции, имеющие максимальную сложность, представимы разложением с M слагаемыми, где M – максимальное их число. Отсюда из (6.16) получаем точное значение δ,

$$\delta(m) = \frac{\sum_{i=1}^{M(1-\varepsilon)} C_{N_c}^{i} (k-1)^{i}}{k^{m}} < k^{-\varepsilon m}, \qquad (6.22)$$

где ε – доля слагаемых от максимального их числа, на которую уменьшается сложность разложения функции.

Из неравенства (6.22) следует, что с ростом m последовательность $\delta(m)$ является ограниченной и монотонно возрастающей, т.е. имеет предел. Следовательно, с ростом m доля функций δ, имеющих меньшую сложность, чем максимально возможная, стремится к нулю. Делаем вывод, что при спектральной декомпозиции количество ненулевых коэффициентов M в разложении функции и число операций L в синтезируемой при этом формуле удовлетворяют следующим асимптотическим оценкам:

$$M \sim \frac{C_\infty}{2}\frac{m}{n}, \quad L \sim \frac{C_\infty}{2} m, \quad C_\infty = \frac{1}{k-\alpha_k}\frac{2}{\bar{k}}, \qquad (6.23)$$

где $\sim$ – знак асимптотического равенства, α_k – предельная порождающая способность аналитической конструкции, $\bar{k}$ – средняя значность переменных. Более того, для любого $\varepsilon > 0$ доля функций δ от общего их числа, для которых

$$M < (1-\varepsilon)\frac{C_\infty}{2}\frac{m}{n}, \quad L < (1-\varepsilon)\frac{C_\infty}{2} m$$

меньше, чем $k^{-\varepsilon m}$ и стремится к нулю с ростом m.

6.3.6. Методика спектрального синтеза

Приведем методику спектрального синтеза формул.[73] Так как ранее обосновано, что без потери общности частичные (спектральные) функции могут быть представлены бесповторными бесскобочными формулами,

$$\theta(X) = x_0 \circ_0 x_1 \circ_1 x_2 \cdots \circ_{n-2} x_{n-1},$$

то для произвольной функции f значности k_f справедливо выражение,

$$f(X) = \sum_{i=1}^{m} \theta_i(X) \times a_i,$$

[73] Описываемая методика спектрального синтеза формул близка декомпозиции в базисе произвольной сложности, осуществляемой путем замены выходных функций, предложенной Пархоменко и Горовым [180, 87]. Однако в отличие от этой методики, благодаря матричному представлению функций и операций, спектральный синтез формул обеспечивает сходимость процесса синтеза.

где разложение осуществляется в одной из образующих алгебр $R =< N_k, +, \times >$, $k_f \le k$, m – длина вектора значений функции, a_i – коэффициенты разложения.

Путем умножения матриц последних операций θ_i на соответствующий им коэффициент a_i получаем разложение,

$$f(X) = \sum_{i=1}^{M} \theta_i(X),$$

записанное в алгебре R с одной операцией, $R =< N_k, + >$, где учтено, что часть коэффициентов a_i равны нулю, $M \le m$.

Спектральная алгебра. Потребуем, чтобы алгебра R была таковой, что для любых ее элементов a и b уравнение $a + x = b$ имеет единственное решение относительно x, $x \in N_k$. Последнее означает, что матрица операции сложения не имеет повторяющихся элементов в любой из ее строк. Заметим, что требование ассоциативности на операцию сложения не накладывается, если порядок вычисления функций θ_i будет соответствовать естественному. Если предполагается синтезированную формулу вычислять в любом порядке, то сложение алгебры R должно быть ассоциативной или ассоциативной и коммутативным операцией.

Матричное представление. Основной операцией спектрального синтеза является представление вектора функции в виде композиции операций, задаваемых в матричном виде.

Пусть задан вектор значений функции $\mathbf{F}$ и вектора значностей переменных $\mathbf{K}$. Найдем $n-1$ матрицу операций $\circ_t$, $t = \overline{0, n-2}$, таких, что $f(X) = \theta(X)$, где $\theta(X)$ – бесповторная бесскобочная функция.

Матрицы операций $\circ_t$ вычислим по следующему рекуррентному правилу:

$$\left.\begin{array}{l} K_t = k_t K_{t-1}, \\ \circ_t = [p_{ij}],\ p_{ij} = i + jK_t \end{array}\right\}\ t = \overline{0, n-3}, \tag{6.24}$$

где $K_0 = 1$, $i = \overline{0, K_{t-1} - 1}$, $j = \overline{0, k_{t+1} - 1}$, и

$$\left.\begin{array}{l} K_{n-1} = k_n K_{n-2}, \\ \circ_{n-1} = [p_{ij}],\ p_{ij} = f_q,\ q = i + jK_{n-1} \end{array}\right\}, \tag{6.25}$$

где f_q – элемент с номер q вектора $\mathbf{F}$, $q = \overline{0, m-1}$, m – длина вектора $\mathbf{F}$. В результате для произвольной функции f получаем матрицы операций ее бесповторной бесскобочной формы, где в общем случае значности операций могут быть больше чем значность образующей алгебры.

Пример 6.3. Пусть требуется найти формулу для вычисления первых 24 десятичных цифр числа $\pi = 3.14159265358979323846265$ по модулю 3. В этом случае вектор значений функции $\mathbf{F} = [011120202022010020210202]$.

Зададим вектор значностей переменных $\mathbf{K} = [2223]$ и спектральную алгебру $R =< N_3, + >$ операцией сложения,

$$+ = \begin{bmatrix} 0 & 1 & 2 \\ 1 & 2 & 0 \\ 2 & 0 & 1 \end{bmatrix},$$

у которой имеется нейтральный элемент $\sigma = 0$, а каждая строка матрицы не содержит двух одинаковых элементов из множества N_3, т.е. уравнение вида $a + x = b$ имеет единственное решения относительно x при любых a и b.

Используя рекуррентное правило (6.24)-(6.25) находим бесповторное бесскобочное представление искомой функции,

$$\begin{array}{c} x_0 \\ \begin{array}{c|cc} 0 & 0 & 2 \\ 1 & 1 & 3 \end{array} \\ \circ_0 \end{array} \quad \begin{array}{c} x_1 \\ \begin{array}{c|cc} 0 & 0 & 4 \\ 1 & 1 & 5 \\ 2 & 2 & 6 \\ 3 & 3 & 7 \end{array} \\ \circ_1 \end{array} \quad \begin{array}{c} x_2 \\ \begin{array}{c|ccc} 0 & 0 & 2 & 2 \\ 1 & 1 & 0 & 0 \\ 2 & 1 & 2 & 2 \\ 3 & 1 & 2 & 1 \\ 4 & 2 & 0 & 0 \\ 5 & 0 & 1 & 2 \\ 6 & 2 & 0 & 0 \\ 7 & 0 & 0 & 2 \end{array} \\ \circ_2 \end{array} \quad \begin{array}{c} x_3 \\ \\ \end{array} \rightarrow f(X).$$

Проверкой убеждаемся, что вычисление вектора функции f по найденному представлению (см. пример 6.2) дает исходный вектор $\mathbf{F}$. ♦

Редукция операций. Следующим шагом спектрального синтеза является редукция операций бесповторной формы, которая заключается в следующем.

Находятся классы эквивалентности строк каждой из матриц $\circ_t$, начиная с последней. Затем матрица $\circ_t$ преобразуется к виду, содержащему по одной строке из каждого класса, а предыдущая операция модифицируется таким образом, чтобы вычисляемая функция осталась неизменной.

Пусть $\{C_0, C_1, \ldots, C_{u-1}\}$ – классы эквивалентности строк матрицы $\circ_t$, состоящие из номеров одинаковых строк, или совместимых строк, если строки содержат неопределенные значения. Редуцированная матрица $\circ_t$ получается так: i-ая строка новой матрицы яв-

ляется строкой из класса эквивалентности C_i. Для сохранения равенства $f(X) = \theta(X)$ все элементы $c \in C_i$ матрицы $\circ_{t-1}$ заменяются на номер класса эквивалентности i. В итоге получаем модифицированную матрицу $\circ_{t-1}$ и матрицу $\circ_t$, не содержащую повторяющихся строк.

Процесс редукции завершается, если все матрицы, кроме первой, редуцированы. Первая матрица формы не нуждается в редукции, так как количество ее строк всегда равно значности первой переменной.

Пример 6.4. Выполним редукцию бесповторной формы функции из примера 6.3. Находим, что количество классов эквивалентности строк последней операции $\circ_2$ равно 7:

$C_0 = \{0\}$, $C_1 = \{1\}$, $C_2 = \{2\}$, $C_3 = \{3\}$, $C_4 = \{4,6\}$, $C_5 = \{5\}$, $C_6 = \{7\}$.

После замены элементов 6 и 7 матрицы $\circ_1$ на элементы 4 и 6 соответственно, получим:

$$x_0 \quad \begin{matrix} & x_1 \\ 0 & 0 & 2 \\ 1 & 1 & 3 \end{matrix} \quad \begin{matrix} & x_2 \\ 0 & 0 & 4 \\ 1 & 1 & 5 \\ 2 & 2 & 4 \\ 3 & 3 & 6 \end{matrix} \quad \begin{matrix} & & x_3 \\ 0 & 0 & 2 & 2 \\ 1 & 1 & 0 & 0 \\ 2 & 1 & 2 & 2 \\ 3 & 1 & 2 & 1 \\ 4 & 2 & 0 & 0 \\ 5 & 0 & 1 & 2 \\ 6 & 0 & 0 & 2 \end{matrix} \quad \rightarrow f(X).$$

$\circ_0$ $\circ_1$ $\circ_2$

Так как матрица операции $\circ_1$ не содержит одинаковых строк, то процесс редукции завершается. ♦

Если в процессе редукции число строк в матрице $\circ_t$ меньше или равно значности k алгебры R, то редукция является полной, в противном случае (см. пример 6.4) – частичной. Так как для синтеза формулы функции необходимо иметь функции с полностью редуцированными операциями, то в случае частичной редукции выберем только k классов эквивалентности и соответствующих им k строк, причем таких, что обеспечивается максимальное покрытие матрицы операции $\circ_t$, а вместо номеров классов эквивалентности в матрице $\circ_{t-1}$ проставим неопределенное значение $*$.

Пример 6.5. Выполним полную редукцию функции из примера 6.4. После редукции операции $\circ_2$ имеем

$$x_0 \; \begin{array}{c} 0 \\ 1 \end{array}\underset{\circ_0}{\overset{x_1}{\begin{bmatrix} 0 & 2 \\ 1 & 3 \end{bmatrix}}} \;\; \begin{array}{c} 0 \\ 1 \\ 2 \\ 3 \end{array}\underset{\circ_1}{\overset{x_2}{\begin{bmatrix} 0 & 2 \\ 1 & * \\ * & 2 \\ * & * \end{bmatrix}}} \;\; \begin{array}{c} 0 \\ 1 \\ 2 \end{array}\underset{\circ_2}{\overset{x_3}{\begin{bmatrix} 0 & 2 & 2 \\ 1 & 0 & 0 \\ 2 & 0 & 0 \end{bmatrix}}},$$

а, выполнив редукцию $\circ_1$, получаем функцию θ, которая уже не равна исходной,

$$x_0 \; \begin{array}{c} 0 \\ 1 \end{array}\underset{\circ_0}{\overset{x_1}{\begin{bmatrix} 0 & 2 \\ 1 & * \end{bmatrix}}} \;\; \begin{array}{c} 0 \\ 1 \\ 2 \end{array}\underset{\circ_1}{\overset{x_2}{\begin{bmatrix} 0 & 2 \\ 1 & * \\ * & 2 \end{bmatrix}}} \;\; \begin{array}{c} 0 \\ 1 \\ 2 \end{array}\underset{\circ_2}{\overset{x_3}{\begin{bmatrix} 0 & 2 & 2 \\ 1 & 0 & 0 \\ 2 & 0 & 0 \end{bmatrix}}} \to \theta(X).$$

Заметим, что полученная форма содержит неопределенные значения, которые на следующих шагах декомпозиции могут быть использованы для минимизации итоговой формулы. ♦

Вычисление остатка. Если в процессе редукции окажется, что редуцированная форма функции не равна исходной, т.е. $f(X) \neq \theta(X)$, то вычисляется остаточный вектор $\mathbf{F}'$, $\mathbf{F}' = \mathbf{F} - \mathbf{G}$, где $\mathbf{G}$ – вектор функции, полученной в результате полной редукции $\mathbf{F}$. Для этого в алгебре R решаются уравнения вида $\theta_i + f_i' = f_i$ относительно неизвестных f_i' при $i = \overline{0, m-1}$.

На следующем шаге спектрального синтеза полученный вектор $\mathbf{F}'$ используется как исходный вектор. Синтез завершается, когда полная редукция функции равна самой редуцируемой функции, т.е. когда $\mathbf{F} = \mathbf{G}$.

Таким образом, спектральный синтез в целом может быть описан следующими рекуррентными формулами,

$$\begin{cases} \mathbf{F}_0 = \mathbf{F}; \\ \mathbf{F}_i = \mathbf{F}_{i-1} - \mathbf{G}_{i-1} \;\; (i = \overline{1,t}); \\ \mathbf{F}_t = \mathbf{G}_i, \end{cases}$$

где $\mathbf{G}_i$ – полная редукция вектора $\mathbf{F}_i$.

Пример 6.6. Выполним вычисление остаточного вектора для функции из примера 6.5. После доопределения $\theta(X)$, вычисление формы

$$x_0 \quad \begin{matrix} & x_1 \\ 0 & \begin{bmatrix} 0 & 2 \end{bmatrix} \\ 1 & \begin{bmatrix} 1 & 0 \end{bmatrix} \end{matrix} \quad \begin{matrix} & x_2 \\ 0 & \begin{bmatrix} 0 & 2 \end{bmatrix} \\ 1 & \begin{bmatrix} 1 & 0 \end{bmatrix} \\ 2 & \begin{bmatrix} 0 & 2 \end{bmatrix} \end{matrix} \quad \begin{matrix} & x_3 \\ 0 & \begin{bmatrix} 0 & 2 & 2 \end{bmatrix} \\ 1 & \begin{bmatrix} 1 & 0 & 0 \end{bmatrix} \\ 2 & \begin{bmatrix} 2 & 0 & 0 \end{bmatrix} \end{matrix} \to \theta(X)$$
$$\circ_0 \qquad\qquad \circ_1 \qquad\qquad \circ_2$$

дает следующий вектор значений, $\mathbf{G} = [012020222002020020020200]$. Отсюда получаем 24 уравнения в алгебре R (определена в примере 6.3),

$$0 + f_0' = 0, \quad 1 + f_1' = 1, \quad 2 + f_2' = 1, \quad 0 + f_3' = 1, \quad 2 + f_4' = 2, \quad 0 + f_5' = 0,$$
$$2 + f_6' = 2, \quad 2 + f_7' = 0, \quad 2 + f_8' = 2, \quad 0 + f_9' = 0, \quad 0 + f_{10}' = 2, \quad 2 + f_{11}' = 2,$$
$$0 + f_{12}' = 0, \quad 1 + f_{13}' = 1, \quad 0 + f_{14}' = 0, \quad 0 + f_{15}' = 0, \quad 2 + f_{16}' = 2, \quad 0 + f_{17}' = 0,$$
$$0 + f_{18}' = 2, \quad 2 + f_{19}' = 1, \quad 0 + f_{20}' = 0, \quad 2 + f_{21}' = 2, \quad 0 + f_{22}' = 0, \quad 0 + f_{22}' = 2,$$

где использован исходный вектор функции $\mathbf{F} = [011120202022010020210202]$.

В результате решения найденных уравнений получаем остаточный вектор $\mathbf{F}'$,

$$\mathbf{F}' = [002100010020020000220002]. \blacklozenge$$

Перестановка переменных. В теории спектральной декомпозиции показано, что порядок переменных не влияет на сложность разложения функции. Однако описанный выше итерационный алгоритм последовательного приближения чувствителен к порядку переменных. По этой причине, для минимизации числа классов эквивалентности, будем изменять порядок переменных при синтезе формул спектральных функций.

Для вычисления вектора функции, полученной после изменении порядка переменных, будем использовать взаимно-однозначное соответствие между значением номера элемента вектора и значениями переменных, определенное в 5.2.2 на с. 236. Заметим, что порядок предыдущих переменных не влияет на число классов эквивалентности строк матрицы текущей операции.

Пример 6.7. Найдем наилучший порядок переменных для редукции функции из примера 6.4. Для этого вычислим вектора значений функции при различных последних переменных и найдем наилучшее покрытие S матрицы последней операции k строками, где k – значность спектральной алгебры R (табл. 6.6).

Таблица 6.6. Перестановки последней переменной

X	**F**	S
$[x_0\ x_1\ x_2\ x_3]$	[01112020202201002021020 2]	12
$[x_3\ x_0\ x_1\ x_2]$	[02210012212120001220000 2]	18
$[x_2\ x_3\ x_0\ x_1]$	[02202010010212202010201 2]	18
$[x_1\ x_2\ x_3\ x_0]$	[01222200220011000210012 2]	16

Выбрав в качестве рабочего второй вариант, получаем функцию $\theta_1(X)$ и остаточный вектор $\mathbf{F}_1$,

$$x_3 \begin{matrix} x_0 \\ \begin{bmatrix} 2 & 1 \\ 0 & 2 \\ 0 & 2 \end{bmatrix} \\ \circ_{10} \end{matrix} \begin{matrix} x_1 \\ \begin{bmatrix} 0 & 0 \\ 2 & 2 \\ 1 & 0 \end{bmatrix} \\ \circ_{11} \end{matrix} \begin{matrix} x_2 \\ \begin{bmatrix} 2 & 0 \\ 0 & 2 \\ 1 & 0 \end{bmatrix} \\ \circ_{12} \end{matrix} \rightarrow \theta_1(X),$$

$$\mathbf{F}_1 = [02210012212120001220000 2].$$

После редукции $\mathbf{F}_1$ с учетом порядка переменных получаем вторую функцию $\theta_2(X)$, вектор которой совпадает с исходным вектором $\mathbf{F}_1$,

$$x_3 \begin{matrix} x_0 \\ \begin{bmatrix} 1 & 0 \\ 0 & 2 \\ 0 & 1 \end{bmatrix} \\ \circ_{20} \end{matrix} \begin{matrix} x_1 \\ \begin{bmatrix} 0 & 0 \\ 0 & 1 \\ 2 & 0 \end{bmatrix} \\ \circ_{21} \end{matrix} \begin{matrix} x_2 \\ \begin{bmatrix} 0 & 0 \\ 2 & 2 \\ 0 & 2 \end{bmatrix} \\ \circ_{22} \end{matrix} \rightarrow \theta_2(X).$$

Окончательно имеем:

$$\mathbf{F} = \mathbf{G}_1 + \mathbf{G}_2,$$

$$f(X) = \theta_1(X) + \theta_2(X),$$

$$f(X) = x_3 \circ_{10} x_0 \circ_{11} x_1 \circ_{12} x_2 + x_3 \circ_{20} x_0 \circ_{21} x_1 \circ_{22} x_2,$$

где в результате проведенного синтеза вычислены следующие операции:

$$\circ_{10} = \begin{bmatrix} 2 & 1 \\ 0 & 2 \\ 0 & 2 \end{bmatrix}, \circ_{11} = \begin{bmatrix} 0 & 0 \\ 2 & 2 \\ 1 & 0 \end{bmatrix}, \circ_{12} = \begin{bmatrix} 2 & 0 \\ 0 & 2 \\ 1 & 0 \end{bmatrix},$$

$$\circ_{20} = \begin{bmatrix} 1 & 0 \\ 0 & 2 \\ 0 & 1 \end{bmatrix}, \circ_{21} = \begin{bmatrix} 0 & 0 \\ 0 & 1 \\ 2 & 0 \end{bmatrix}, \circ_{22} = \begin{bmatrix} 0 & 0 \\ 2 & 2 \\ 0 & 2 \end{bmatrix}.$$

Подсчитаем теперь теоретическую сложность функции f, для чего воспользуемся формулой (6.17) при $\alpha = 0{,}91$,

$$M \le \frac{1}{3-0{,}91}\ \frac{24}{9-(2+2-1{,}67)} \approx 1{,}72\,.$$

Полученный результат означает, что для представления произвольной функции с длиной вектора 24 и значностями переменных $K = \{2,2,2,3\}$ требуется не более чем две частичные функции. Таким образом, в классе бинарных операций синтезирована минимальная спектральная формула функции, предназначенная для вычисления первых 24 цифр числа π, взятых по модулю 3. ♦

Приведем еще один более сложный пример спектрального синтеза.

Пример 6.8. Пусть требуется найти формулу для вычисления первых 32 десятичных цифр числа $\pi = 3{,}14159265358979323846264338327950$ по модулю 3. Вычислим вектор значений функции и зададим вектор значностей переменных

$$\mathbf{F} = [01112020202001002021222100202102],\ \mathbf{K} = [22222]$$

Для заданной функции $\alpha = 1{,}16$ и $M = 2{,}2$. В результате спектрального синтеза в алгебре $R =< N_3, + >$, образованной на N_3 операцией сложения по модулю 3, имеем:

$$f(X) = x_4 \overset{x_1}{\begin{bmatrix} 0 & 0 \\ 1 & 2 \end{bmatrix}} \overset{x_3}{\begin{bmatrix} 0 & 1 \\ 1 & 2 \\ 2 & 1 \end{bmatrix}} \overset{x_2}{\begin{bmatrix} 2 & 0 \\ 0 & 2 \\ 1 & 1 \end{bmatrix}} \overset{x_0}{\begin{bmatrix} 2 & 0 \\ 2 & 1 \\ 0 & 1 \end{bmatrix}} +$$

$$+ x_1 \overset{x_2}{\begin{bmatrix} 1 & 2 \\ 0 & 0 \end{bmatrix}} \overset{x_3}{\begin{bmatrix} 0 & 0 \\ 0 & 2 \\ 1 & 0 \end{bmatrix}} \overset{x_0}{\begin{bmatrix} 0 & 0 \\ 2 & 1 \\ 1 & 2 \end{bmatrix}} \overset{x_4}{\begin{bmatrix} 0 & 0 \\ 0 & 1 \\ 0 & 2 \end{bmatrix}} +$$

$$+ x_0 \overset{x_1}{\begin{bmatrix} 0 & 1 \\ 0 & 2 \end{bmatrix}} \overset{x_2}{\begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 2 \end{bmatrix}} \overset{x_3}{\begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 2 \end{bmatrix}} \overset{x_4}{\begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 2 & 1 \end{bmatrix}}. \ ♦$$

Методика синтеза формул, аналогичная описанной выше, может быть использована для решения широкого круга прикладных задач. В частности, в Приложении 5 рассмотрена полиномиальная факторизация спектральных базисов и показано ее применение для сжатия сигналов и изображений.

6.3.7. Эффективность реализации

Эффективность спектрального представления для дискретной обработки данных определим относительно метода табличных вычислений, который инвариантен выбору функции, описывающей эту обработку. При этом будем учитывать не только время вы-

числения, но и объем памяти, необходимы для его организации. Для этого введем критерий эффективности E:

$$E = \frac{V_T}{V_S}\frac{T_T}{T_S},$$

где V_S – объем памяти, необходимый для спектрального представления функции, V_T – объем памяти, требуемый для хранения вектора значений функции, T_S – время, затраченное на вычисление функции по спектральному представлению, T_T – время табличного вычисления,

$$V_T = \prod_{i=1}^{n} k_i \ , \ T_T = 1;$$

$$V_S = 1 \ , \ T_S \le \frac{n}{k-\alpha}\frac{\prod_{i=1}^{n} k_i}{\sum_{i=1}^{n} k_i - \beta}.$$

Отсюда эффективность спектральной реализации произвольной функции может быть выражена следующим образом:

$$E_1 \ge \frac{1}{n}(k-\alpha)\left(\sum_{i=0}^{n-1} k_i - \beta\right) = (k-\alpha)\left(\bar{k} - \frac{\beta}{n}\right) > 1,$$

где $\bar{k}$ – средняя значность переменных. В асимптотической области имеем

$$E_1 \sim (k-\alpha_k)\bar{k}\ .$$

Пример 6.9. Вычислим эффективность реализации спектрального представления функции из примера 6.7,

$$E_1 = \frac{1}{4}(3-0{,}91)(5+\frac{4-0{,}5}{3-0{,}91}) \approx 3{,}46\ . \ \blacklozenge$$

Таким образом, спектральное представление позволяет реализовать вычисление дискретной функции примерно в $(k-\alpha)\bar{k}$ раз эффективнее, чем это возможно при табличных вычислениях. Заметим, что для табличной реализации необходимо вычислять индекс x входа в вектор значений функции,

$$x = x_0 + k_0(x_1 + k_1(x_2 + \ldots + k_{n-2}x_{n-1})\ldots),$$

где x_i – значения переменных, k_i – их значности, $i = \overline{0,n-1}$. На это уходит $2(n-1)$ единиц времени. Но и спектральное представление требует дополнительных затрат на реализацию операций спектральной формы. Однако в противоположность табличному методу, спектральная обработка данных допускает распараллеливание (рис. 6.11).

В этом случае $M(n-1)$ операций могут выполняться одновременно, что потребует всего $n-1$ единиц времени. В итоге получаем $T_S \le M+n-2$ и

$$E_2 \ge \frac{(k-\alpha)\left(\sum_{i=1}^{n} k_i - \beta\right)\prod_{i=1}^{n} k_i}{(n-2)(k-\alpha)\left(\sum_{i=1}^{n} k_i - \beta\right)+\prod_{i=1}^{n} k_i} > E_1 .$$

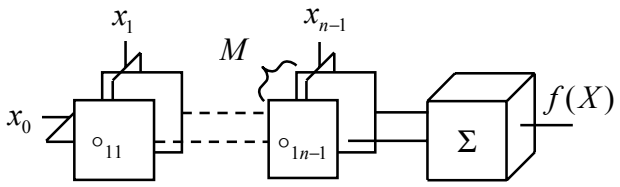

Рис. 6.11. Параллельные спектральные вычисления

Пример 6.10. Параллельные вычисления функции из примера 6.7 могут быть выполнены с эффективностью $E_2 \approx 6{,}44$, где вычисления выполнены при $\alpha = 0{,}91$ и $\beta = 2{,}33$. ♦

6.4. Алгебраический синтез формул

Ранее показано, что сложность представления функции при ее спектральном разложении примерно в $(k-\alpha)\bar{k}$ раз меньше длины вектора функции. Найдем, как изменится сложность представления при алгебраической декомпозиции в общем случае.

6.4.1. Связанные коэффициенты

Усложним аналитическую конструкцию синтезируемых формул, для чего рассмотрим решающую декомпозицию вида

$$f(X) = \sum_{i=0}^{M-1} \theta_i'(X') \times \theta_i''(X'') = \sum_{i=0}^{M-1} \theta_i(X', X''), \qquad (6.26)$$

где θ_i' и θ_i'' – бесповторные бесскобочные подформулы, $\theta_i(X', X'')$ – бесповторные скобочные формулы с двумя парами скобок единичной глубины вложенности.

На рис. 6.12 показано, что спектральные функции и коэффициенты в рассматриваемом случае являются связанными, т.е. возможность их изменения определяется комбинаторными возможностями аналитической конструкции бесповторных бесскобочных формул.

Подсчитаем количество функций, порождаемых разложением (6.26). Для порождения всех функции необходимо, чтобы

$$k^m = \sum_{i=1}^{M} C_{N_c'}^{i} C_{N_c''}^{i},$$

где N'_c и N''_c – число линейно-независимых классов функций с длиной вектора k' и k'' соответственно.

$$\begin{bmatrix} \theta'_0 & \theta'_1 & \theta'_2 & \theta'_3 & \theta'_4 & \theta'_5 & \theta'_6 & \theta'_7 \\ \theta'_0 & \theta'_1 & \theta'_2 & \theta'_3 & \theta'_4 & \theta'_5 & \theta'_6 & \theta'_7 \\ \theta'_0 & \theta'_1 & \theta'_2 & \theta'_3 & \theta'_4 & \theta'_5 & \theta'_6 & \theta'_7 \\ \theta'_0 & \theta'_1 & \theta'_2 & \theta'_3 & \theta'_4 & \theta'_5 & \theta'_6 & \theta'_7 \\ \theta'_0 & \theta'_1 & \theta'_2 & \theta'_3 & \theta'_4 & \theta'_5 & \theta'_6 & \theta'_7 \\ \theta'_0 & \theta'_1 & \theta'_2 & \theta'_3 & \theta'_4 & \theta'_5 & \theta'_6 & \theta'_7 \end{bmatrix} \times \begin{bmatrix} \theta''_0 & \theta''_0 & \theta''_0 & \theta''_0 \\ \theta''_1 & \theta''_1 & \theta''_1 & \theta''_1 \\ \theta''_2 & \theta''_2 & \theta''_2 & \theta''_2 \\ \theta''_3 & \theta''_3 & \theta''_3 & \theta''_3 \\ \theta''_4 & \theta''_4 & \theta''_4 & \theta''_4 \\ \theta''_5 & \theta''_5 & \theta''_5 & \theta''_5 \\ \theta''_6 & \theta''_6 & \theta''_6 & \theta''_6 \\ \theta''_7 & \theta''_7 & \theta''_7 & \theta''_7 \end{bmatrix} = \begin{bmatrix} * & * & * & * \\ * & * & * & * \\ * & * & * & * \\ * & * & * & * \\ * & * & * & * \\ * & * & * & * \end{bmatrix}$$

Рис. 6.12. Решающее разложение при связанных коэффициентах: θ'_i – элементы линейно-независимых столбцов; θ''_i – элементы линейно-независимых строк; $*$ – произвольные элементы

По аналогии со спектральным разложением имеем

$$M = \frac{1}{k-\alpha} \frac{\prod_{i=0}^{n-1} k_i}{\sum_{i=0}^{n-1} k_i - \beta' - \beta''}, \tag{6.27}$$

где

$$\beta' = k'_0 + k'_1 - \frac{k'_0 k'_1 + \log_k \chi - 1}{k-\alpha}, \quad \beta'' = k''_0 + k''_1 - \frac{k''_0 k''_1 + \log_k \chi - 1}{k-\alpha},$$

а доля невырожденных матриц χ подсчитывается по формуле (6.19).

Как и следовало ожидать, использование скобочных конструкций незначительно увеличило сложность разложения, однако при большой длине вектора функции это не оказывает заметного влияния. Из оценки (6.27) также видно, что сложность разложения не зависит от разделения переменных, а при небольшом числе переменных определяется с точностью до выбора первых двух переменных из каждого подмножества.

6.4.2. Свободные коэффициенты

Рассмотрим решающее разложение при свободных коэффициентах,

$$f(X) = \sum_{i=0}^{M-1} \theta'_i(X') \times a_i(X''), \tag{6.28}$$

где a_i – произвольные функции, формульные представления которых неизвестны.

На рис. 6.13 показано, что частичные функции являются связанными и порождаются бесповторной аналитической конструкцией, а функции-коэффициенты – свободными, т.к. имеется возможность их изменения произвольным образом.

Воспользуемся степенями свободы задания частичных функций в разложении (6.28) при минимальном M и представим эти функции бесповторными формулами.

$$\begin{bmatrix} \theta'_0 & \theta'_1 & \theta'_2 & \theta'_3 & \phi & \phi \\ \theta'_0 & \theta'_1 & \theta'_2 & \theta'_3 & \phi & \phi \\ \theta'_0 & \theta'_1 & \theta'_2 & \theta'_3 & \phi & \phi \\ \theta'_0 & \theta'_1 & \theta'_2 & \theta'_3 & \phi & \phi \\ \theta'_0 & \theta'_1 & \theta'_2 & \theta'_3 & \phi & \phi \\ \theta'_0 & \theta'_1 & \theta'_2 & \theta'_3 & \phi & \phi \end{bmatrix} \times \begin{bmatrix} + & + & + & + \\ + & + & + & + \\ + & + & + & + \\ + & + & + & + \\ \phi & \phi & \phi & \phi \\ \phi & \phi & \phi & \phi \end{bmatrix} = \begin{bmatrix} * & * & * & * \\ * & * & * & * \\ * & * & * & * \\ * & * & * & * \\ * & * & * & * \\ * & * & * & * \end{bmatrix}$$

Рис. 6.13. Решающее разложение при свободных коэффициентах: θ'_i – элементы линейно-независимых столбцов; $+$ – свободные элементы; $*$ – произвольные элементы, ϕ – нулеобразующий элемент

Выберем в качестве первого приближения системы частичных функций линейно-независимые столбцы матрицы функции (рис. 6.14).

$$\begin{bmatrix} \theta'_0 & \theta'_1 & \theta'_2 & \theta'_3 & \phi & \phi \\ \theta'_0 & \theta'_1 & \theta'_2 & \theta'_3 & \phi & \phi \\ \theta'_0 & \theta'_1 & \theta'_2 & \theta'_3 & \phi & \phi \\ \theta'_0 & \theta'_1 & \theta'_2 & \theta'_3 & \phi & \phi \\ \theta'_0 & \theta'_1 & \theta'_2 & \theta'_3 & \phi & \phi \\ \theta'_0 & \theta'_1 & \theta'_2 & \theta'_3 & \phi & \phi \end{bmatrix} \times \begin{bmatrix} \tau & \phi & \phi & \phi \\ \phi & \tau & \phi & \phi \\ \phi & \phi & \tau & \phi \\ \phi & \phi & \phi & \tau \\ \phi & \phi & \phi & \phi \\ \phi & \phi & \phi & \phi \end{bmatrix} = \begin{bmatrix} \theta'_0 & \theta'_1 & \theta'_2 & \theta'_3 \\ \theta'_0 & \theta'_1 & \theta'_2 & \theta'_3 \\ \theta'_0 & \theta'_1 & \theta'_2 & \theta'_3 \\ \theta'_0 & \theta'_1 & \theta'_2 & \theta'_3 \\ \theta'_0 & \theta'_1 & \theta'_2 & \theta'_3 \\ \theta'_0 & \theta'_1 & \theta'_2 & \theta'_3 \end{bmatrix}$$

Рис. 6.14. Первое приближение частичных функций: θ'_i – элементы линейно-независимых столбцов функции; ϕ – нулеобразующий элемент; τ – единичный элемент

Прибавим к правой и левой части формулы (6.28) элемент $c \times \theta_u(X') \times a_v(X'')$, где u и v – допустимые индексы, c – произвольная константа. После приведения подобных слагаемых находим, как изменились в разложении функции и коэффициенты:

$$\theta'_v(X') = \theta'_v(X') + c \times \theta'_u(X'), \quad a_u(X'') = a_u(X'') - c \times a_v(X'').$$

Отсюда делаем вывод, что для формирования системы частичных функций возможно использование различных линейных комбинаций столбцов исходной матрицы. Количество таких комбинаций равно $k^{k''} - 1$, а это позволяет сформировать фактически произвольную систему функций, в том числе и представимую формулами с наименьшим числом операций.

При уменьшении k'' количество слагаемых в разложении уменьшается. В пределе имеем тривиальное разложение, состоящее из одной функции и одного коэффициента. Однако такая функция не имеет бесповторного представления, как изначально не имеет такого представления декомпозируемая функция. Следовательно, существует некоторое

минимальное k'', при котором система частичных функций все еще представима бесповторными формулами.

Так как при достаточно большом $k'' \le k'$ частичные функции фактически произвольные, представим их как спектральное разложение некоторой неизвестной функции с длиной вектора k'. Тогда из (6.17) следует условие

$$\prod_{j=0}^{n''-1} k_j'' = \frac{1}{k-\alpha} \frac{\prod_{i=0}^{n'-1} k_i'}{\sum_{i=0}^{n'-1} k_i' - \beta'}, \tag{6.29}$$

при котором гарантируется представимость системы из k'' частичных функций длины k' бесповторными формулами, а сложность решающего разложения M при свободных коэффициентах равна k''.

6.4.3. Двухступенчатая декомпозиция

Рассмотрим декомпозицию со свободными коэффициентами, у которой коэффициенты, в свою очередь, подвергнуты спектральному разложению,

$$f(X) = \sum_{i=0}^{M'-1} \theta_i'(X') \times \sum_{j=0}^{M''-1} \theta_{ij}''(X'') = \sum_{i=0}^{M-1} \theta_i(X', X''),$$

где $M = M'M''$, $M' = k''$, θ_i' и θ_{ij}'' – бесповторные бесскобочные конструкции, θ_i – бесповторные скобочные формулы, состоящие из двух бесскобочных подформул.

Из (6.17) и (6.29) получаем,

$$M = \frac{1}{(k-\alpha)^2} \frac{\prod_{i=0}^{n-1} k_i}{(\sum_{i=0}^{n'-1} k_i' - \beta')(\sum_{j=0}^{n''-1} k_j'' - \beta'')}. \tag{6.30}$$

Найдем разделение переменных, которое обеспечивает наименьшую сложность разложения. Сформулируем задачу поиска условного экстремума (6.30):

$$\begin{cases} s = \sum_{i=0}^{n'-1} k_i' + \sum_{j=0}^{n''-1} k_j''; \\ m = (k-\alpha)\left(\sum_{i=0}^{n'-1} k_i' - \beta'\right) \prod_{j=0}^{n''-1} k_j''^2; \\ (\sum_{i=0}^{n'-1} k_i' - \beta')(\sum_{j=0}^{n''-1} k_j'' - \beta'') \to \max, \end{cases} \tag{6.31}$$

где s и m – константы, первое и второе тождество задает зависимость переменных, а последнее выражение определяет целевую функцию. Решение (6.31) методом неопределен-

ных множителей Лагранжа [29, с. 386] выявляет следующую связь между значностями переменных[74], при которой достигается глобальный максимум целевой функции:

$$\sum_{i=0}^{n'-1} k_i' - \sum_{j=0}^{n''-1} k_j'' = \mu(2n''-1), \tag{6.32}$$

где μ – некоторая константа, не зависящая от разделения переменных. Полагая $\mu(2n''-1) = \rho n$, получаем

$$M = \frac{4}{(k-\alpha)^2} \frac{\prod_{i=0}^{n-1} k_i}{(\sum_{i=0}^{n-1} k_i - \beta' - \beta'')^2 - \rho^2 n^2}, \quad \rho = \frac{1}{n}(\sum_{i=0}^{n'-1} k_i' - \sum_{j=0}^{n''-1} k_j''), \tag{6.33}$$

где ρ – параметр, характеризующий разделение переменных.

Для оценки μ используем прологарифмированную формулу (6.29), заданную при одинаковых значностях переменных. Полагая $n'' \approx n' + 1$ получаем

$$\mu \approx \frac{\log_{\tilde{k}}\left(\frac{1}{2}(k-\alpha)(\bar{k}n - 2\beta + \bar{k})\right)}{n-1}, \tag{6.34}$$

где $\bar{k}$ ($\tilde{k}$) – соответственно средняя арифметическая и средняя геометрическая значность переменных. После оценки μ осталось оценить ρ_{opt}. Для этого запишем систему уравнений,

$$\begin{cases} \rho n = \bar{k}(n' - n''); \\ \rho n = \mu(2n'' - 1); \\ n = n' + n'', \end{cases} \tag{6.35}$$

где первое уравнение получено из второй формулы (6.34), записанной для средних значностей переменных. Решая (6.35) относительно ρ получаем

$$\rho_{opt} = \frac{\bar{k}\mu}{\bar{k} + \mu} \frac{n-1}{n}. \tag{6.36}$$

Для более точных вычислений ρ_{opt} следует использовать трансцендентное уравнение

$$\rho n = \bar{k} \log_{\tilde{k}}\left(\frac{1}{2}(k-\alpha)(\bar{k}n - 2\beta + \rho n)\right), \tag{6.37}$$

полученное из (6.29) и (6.32).

На рис. 6.15 приведена зависимость оптимального значения ρ_{opt} от количества переменных n при их одинаковых значностях, равных k, вычисленная путем численного

[74] Решение (6.31) получено при условии $\sum_{i=0}^{n'-1} k_i' >> \beta'$.

решения уравнения (6.37). Заметим, что оптимальное значение ρ стремиться к нулю с ростом n.

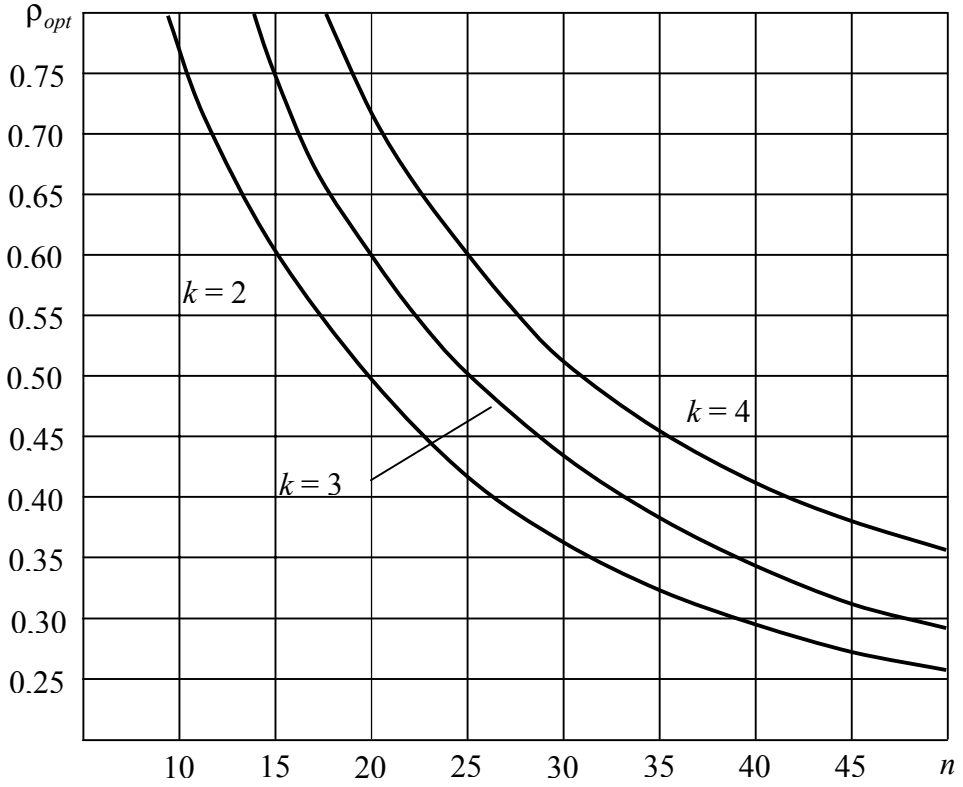

Рис. 6.15. Оптимальное разделение переменных

6.4.4. Многоступенчатая декомпозиция

Максимально усложним аналитическую конструкцию формул и рассмотрим декомпозицию при свободных коэффициентах, у которой функции-коэффициенты, в свою очередь, подвергнуты декомпозиции со связанными коэффициентами:

$$f(X) = \sum_{i=0}^{M'-1} \theta'_i(X') \times \sum_{j=0}^{M''-1} \theta''_{ij}(X''_0, X''_1) = \sum_{i=0}^{M-1} \theta_i(X', X''_0, X''_1),$$

где $M = k''M''$, θ'_i – бесповторные бесскобочные формулы, θ''_{ij} – бесповторные скобочные формулы, состоящие из двух бесскобочных подформул, $X'' = X''_0 \cup X''_1$, а θ_i – бесповторные формулы, состоящие из трех бесскобочных подформул.

Аналогично двухступенчатой декомпозиции находим оценку сложности разложения,

$$M = \frac{4}{(k-\alpha)^2} \frac{\prod_{i=0}^{n-1} k_i}{(\sum_{i=0}^{n-1} k_i - \beta' - \beta''_0 - \beta''_1)^2 - \rho^2 n^2},$$

а обобщая полученный результат на случай относительно произвольной расстановки скобок, имеем

$$M = \frac{4}{(k-\alpha)^2} \frac{\prod_{i=0}^{n-1} k_i}{(\sum_{i=0}^{n-1} k_i - \sum_{j=0}^{\gamma-1} \beta_j)^2 - \rho^2 n^2}, \tag{6.38}$$

где γ – количество скобок в аналитической конструкции, $2 \le \gamma < \log_2 n - 1$, β_j – поправочные коэффициенты, зависящие от значностей операндов значностей k_0^j и k_1^j первых операций в каждой j-й скобке,

$$\beta_j = k_0^j + k_1^j - \frac{k_0^j k_1^j + \log_k \chi - 1}{k - \alpha}. \tag{6.39}$$

Очевидно, увеличивая глубину вложенности скобок, оценку (6.38) можно только ухудшить. Из анализа (6.38) также следует, что наименьшая сложность разложения достигается при $\gamma = 2$.

6.4.5. Сложность представления

Из формулы (6.38) получаем максимальное количество операций, необходимых для представления произвольной функции,

$$L = C^2 \frac{m}{n}, \qquad C = \frac{1}{k-\alpha} \frac{2}{\sqrt{\left(\frac{1}{n}(\sum_{i=0}^{n-1} k_i - \sum_{j=0}^{\gamma-1} \beta_j)\right)^2 - \rho^2}}, \tag{6.40}$$

где C – константа, характеризующая аналитическую конструкцию, α – порождающая способность формул, γ – количество скобок, β_j – поправки для начальной порождающей способности скобочных подформул, ρ – оптимальность разделения переменных.

Из (6.40) находим, что количество ненулевых коэффициентов M в решающем разложении функции и число операций L в синтезируемой при этом формуле удовлетворяют следующим асимптотическим оценкам:

$$M \sim C_\infty^2 \frac{m}{n^2}, \quad L \sim C_\infty^2 \frac{m}{n}, \quad C_\infty = \frac{1}{k-\alpha_k} \frac{2}{\bar{k}}, \tag{6.41}$$

причем для любого $\varepsilon > 0$ доля функций δ от общего их числа[75], для которых

$$M < (1-\varepsilon) C_\infty^2 \frac{m}{n^2}, \quad L < (1-\varepsilon) C_\infty^2 \frac{m}{n}$$

стремится к нулю с ростом m как $k^{-\varepsilon\sqrt{m}}$.

[75] Сложность решающего разложения может быть уменьшена только за счет спектрального разложения коэффициентов. Полагая $k'' < \sqrt{m}$, из (6.22) получаем искомую долю функций.

Зависимость квадрата C_∞ от значности образующей алгебры k показана на рис. 6.16.

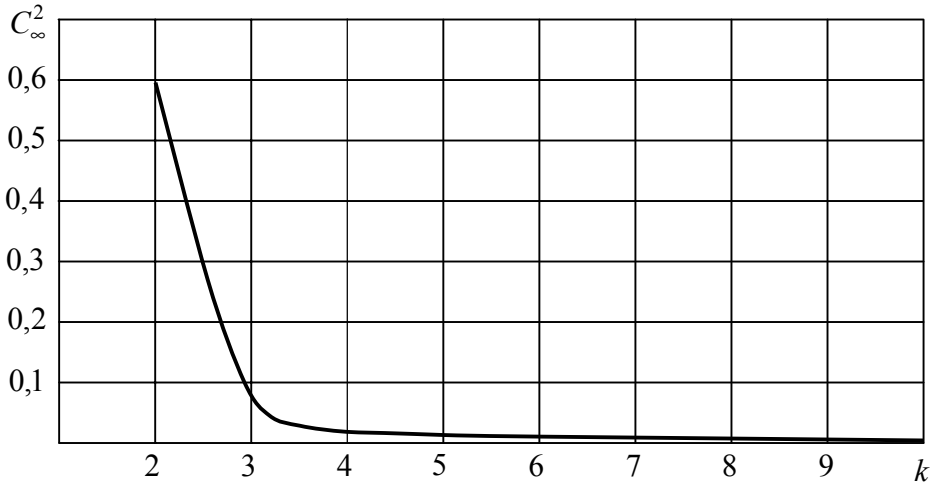

Рис. 6.16. Зависимость C_∞ от значности образующей алгебры k

В частности, из (6.41) следует асимптотическая оценка количества операций, необходимых для представления произвольной булевой функции,

$$L(n) \sim \frac{4}{(1+\log_2 3)^2}\frac{2^n}{n} \approx 0{,}5986\frac{2^n}{n},$$

которая согласуется с оценкой количества команд (операторов и условных переходов), полученной для программной реализации последней [141]. Заметим, что программная реализация булевых функций имеет ту же природу, что и аппаратурная [138].

6.4.6. Методика алгебраического синтеза

При алгебраическом синтезе необходимо обеспечить разделение переменных, такое, что выполняется условие (6.32). Однако при небольшой длине вектора функции, эта задача невыполнима, так как дискретность значностей переменных не позволяют выполнить их оптимальное разделение на два непересекающихся подмножества. Для подтверждения этого вывода рассмотрим результаты вычисления сложности разложения при спектральной и алгебраической декомпозиции, приведенные в табл. 6.7.

Из анализа таблицы следует, что минимальная длина вектора функции, при которой возможна ее алгебраическая декомпозиция в двузначной алгебре, равна 512. В этом случае сложность алгебраической декомпозиции функции будет равна 16, в противоположность спектральной декомпозиции, которая обеспечивает сложность разложения, равную 25.

В трехзначной алгебре требуется рассмотреть функцию с длиной вектора 729, что обеспечит получение разложения с 9 слагаемыми против 20 слагаемых, даваемых спектральным разложением.

Таблица 6.7. Сложность спектральной и алгебраической декомпозиции[76]

k	n	$M_s\ (M)$	$M_a\ (M)$	$\rho_{opt}\ (\rho)$	α
2	5	2,96 (3)	–	–	0,740
	6	4,78 (5)	–	–	0,732
	7	8,00 (8)	7,70 (8)	1,009 (0,857)	0,727
	8	13,79 (14)	8,08 (16)	0,927 (0,500)	0,724
	9	24,21 (25)	10,19 (16)	0,859 (0,666)	0,721
	10	43,14 (44)	14,15 (16)	0,801 (0,800)	0,720
3	5	8,08 (9)	–	–	0,602
	6	19,56 (20)	7,22 (9)	1,490 (1,000)	0,602
	7	49,17 (50)	10,44 (9)	1,333 (1,286)	0,601
	8	126,93 (127)	19,02 (27)	1,209 (0,750)	0,600
	9	324,16 (325)	38,80 (81)	1,107 (0,333)	0,600
	10	893,15 (894)	84,80 (243)	1,023 (0,000)	0,600
4	5	18,07 (19)	7,88 (16)	2,140 (0,800)	0,587
	6	58,25 (59)	9,34 (16)	1,866 (1,333)	0,586
	7	195,09 (196)	19,39 (64)	1,659 (0,571)	0,586
	8	671,16 (672)	48,64 (64)	1,497 (1,000)	0,586
	9	2355,11 (2356)	134,74 (256)	1,365 (0,444)	0,585
	10	8390,54 (8391)	397,36 (1024)	1,258 (0,000)	0,585
5	5	34,39 (35)	7,73 (25)	2,587 (1,000)	0,598
	6	138,42 (139)	12,77 (25)	2,246 (1,666)	0,598
	7	579,15 (580)	34,06 (125)	1,990 (0,714)	0,598
	8	2489,52 (2490)	108,11 (125)	1,790 (1,250)	0,598
	9	10916,22 (10917)	377,07 (625)	1,629 (0,555)	0,598
	10	48601,85 (48602)	1396,54 (3125)	1,497 (0,000)	0,598

Однако показать методику синтеза на примерах столь большой размерности не представляется возможным. Поэтому рассмотрим демонстрационный пример меньшей размерности, в котором будет синтезировано алгебраическое представление функции, совпадающее по эффективности со спектральным.

Пример 6.11. Синтезируем формулу для вычисления 48 десятичных цифр числа π по модулю 3,

$$\pi = 3{,}141592653589793238462643383279502884197169399 37\,,$$

т.е. вектор значений функции с переменными значности [32222] равен

$$[011120202022010020212221002021020222110110000001]\,.$$

Разделим переменные на два непересекающихся множества,

$$X' = \{x_0, x_1, x_2\}\,,\ X'' = \{x_3, x_4\}\,,$$

и представим функцию ее алгебраическим разложением (рис. 6.14),

[76] В круглых скобках указаны реальные сложности спектральной и алгебраической декомпозиции, а также параметр разделения переменных, полученные с учетом дискретности значностей переменных.

$$\left[\begin{array}{cccc:cccccccc}
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
2 & 2 & 2 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
2 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
2 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 2 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
2 & 2 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
2 & 1 & 2 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{array}\right] \times \left[\begin{array}{cccc}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0
\end{array}\right] = \left[\begin{array}{cccc}
0 & 0 & 0 & 1 \\
1 & 1 & 0 & 1 \\
1 & 0 & 2 & 0 \\
1 & 0 & 0 & 1 \\
2 & 2 & 2 & 1 \\
0 & 0 & 1 & 0 \\
2 & 2 & 0 & 0 \\
0 & 1 & 2 & 0 \\
2 & 2 & 0 & 0 \\
0 & 2 & 2 & 0 \\
2 & 2 & 2 & 0 \\
2 & 1 & 2 & 1
\end{array}\right],$$

выполненном в конечном поле, образованном на множестве N_3 операциями сложения и умножения по модулю 3. В матричной форме разложение имеет вид,

$$\mathbf{D} \times \mathbf{A} = \mathbf{T},$$

где $\mathbf{D}$ ($\mathbf{A}$) – матрица частичных функций θ_i (a_i), $\mathbf{T}$ – матрица функции, найденная для заданного разделения переменных.

В общем случае, при единичной матрице $\mathbf{A}$, вектора частичных функций могут не иметь бесповторного бесскобочного представления. Найдем такие линейные комбинации столбцов матрицы $\mathbf{T}$, которые представимы бесповторными бесскобочными формулами,

$$\begin{cases}
\mathbf{D}_0 = a_{00} \times \mathbf{T}_0 + a_{01} \times \mathbf{T}_1 + a_{02} \times \mathbf{T}_2 + a_{03} \times \mathbf{T}_3; \\
\mathbf{D}_1 = a_{10} \times \mathbf{T}_0 + a_{11} \times \mathbf{T}_1 + a_{12} \times \mathbf{T}_2 + a_{13} \times \mathbf{T}_3; \\
\mathbf{D}_2 = a_{20} \times \mathbf{T}_0 + a_{21} \times \mathbf{T}_1 + a_{22} \times \mathbf{T}_2 + a_{23} \times \mathbf{T}_3; \\
\mathbf{D}_3 = a_{30} \times \mathbf{T}_0 + a_{31} \times \mathbf{T}_1 + a_{32} \times \mathbf{T}_2 + a_{33} \times \mathbf{T}_3,
\end{cases}$$

или

$$\mathbf{D}_i = \sum_{j=0}^{3} a_{ij} \times \mathbf{T}_j \ \ (i = \overline{0,3}),$$

где a_{ij} – искомые коэффициенты, $\mathbf{T}_i$ – столбцы матрицы функции $\mathbf{T}$, $\mathbf{D}_i$ – столбцы матрицы частичных функций $\mathbf{D}$.

Заметим, что коэффициенты a_{ij} образуют матрицу $\tilde{\mathbf{A}}$, состоящую из строк функций-коэффициентов $a_i(X'')$, которые на следующем шаге подлежат спектральной декомпозиции. Для заданной в примере функции методом перебора различных линейных комбинаций столбцов $\mathbf{T}_i$ находим коэффициенты a_{ij},

$$\widetilde{\mathbf{A}} = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 2 & 0 & 1 & 1 \end{bmatrix},$$

при которых первые четыре столбца матрицы $\mathbf{D}$ имеют бесповторную бесскобочную формулу. Отсюда получаем новое матричное разложение функции,

$$\begin{bmatrix} 1 & 1 & 0 & 1 \\ 1 & 2 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 2 & 2 & 1 \\ 0 & 1 & 2 & 2 \\ 0 & 2 & 2 & 1 \\ 0 & 2 & 2 & 2 \\ 0 & 2 & 1 & 0 \\ 1 & 2 & 1 & 1 \end{bmatrix} \times \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 2 & 0 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 0 & 2 & 0 \\ 1 & 0 & 0 & 1 \\ 2 & 2 & 2 & 1 \\ 0 & 0 & 1 & 0 \\ 2 & 2 & 0 & 0 \\ 0 & 1 & 2 & 0 \\ 2 & 2 & 0 & 0 \\ 0 & 2 & 2 & 0 \\ 2 & 2 & 2 & 0 \\ 2 & 1 & 2 & 1 \end{bmatrix},$$

у которого столбцы $\mathbf{D}_i$ являются векторами значений $\theta_i(X')$ и представляются формулами:

$$\theta_0(X') = x_0 \circ_{00} x_1 \circ_{01} x_2, \quad \circ_{00} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 1 & 2 \end{bmatrix}, \quad \circ_{01} = \begin{bmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 1 \end{bmatrix};$$

$$\theta_1(X') = x_0 \circ_{10} x_1 \circ_{11} x_2, \quad \circ_{10} = \begin{bmatrix} 0 & 0 \\ 1 & 2 \\ 2 & 2 \end{bmatrix}, \quad \circ_{11} = \begin{bmatrix} 1 & 2 \\ 2 & 1 \\ 0 & 2 \end{bmatrix};$$

$$\theta_2(X') = x_0 \circ_{20} x_1 \circ_{21} x_2, \quad \circ_{20} = \begin{bmatrix} 0 & 1 \\ 1 & 2 \\ 0 & 2 \end{bmatrix}, \quad \circ_{21} = \begin{bmatrix} 0 & 2 \\ 1 & 2 \\ 1 & 1 \end{bmatrix};$$

$$\theta_3(X') = x_0 \circ_{30} x_1 \circ_{31} x_2, \quad \circ_{30} = \begin{bmatrix} 0 & 1 \\ 1 & 2 \\ 0 & 0 \end{bmatrix}, \quad \circ_{31} = \begin{bmatrix} 1 & 1 \\ 0 & 2 \\ 1 & 0 \end{bmatrix}.$$

В итоге получаем следующее разложение искомой функции:

$$f(X) = \sum_{i=0}^{3} (x_0 \circ_{i0} x_1 \circ_{i1} x_2) \times (x_3 \circ_{i2} x_4),$$

где в результате спектрального синтеза функций-коэффициентов $a_i(X'') = x_3 \circ_{i2} x_4$ найдены операции $\circ_{i2}$:

$$\circ_{02} = \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix}, \quad \circ_{12} = \begin{bmatrix} 0 & 0 \\ 1 & 1 \end{bmatrix}, \quad \circ_{22} = \begin{bmatrix} 1 & 1 \\ 0 & 0 \end{bmatrix}, \quad \circ_{32} = \begin{bmatrix} 2 & 1 \\ 0 & 1 \end{bmatrix}.$$

Теперь выполним минимизацию операций. Так как операции $\circ_{12}$ и $\circ_{12}$ имеют одинаковые столбцы, то эти операции несущественным образом зависят от второго операнда и могут быть представлены как унарные операции $\neg_{12}$ и $\neg_{12}$,

$$\circ_{12} = \begin{bmatrix} 0 & 0 \\ 1 & 1 \end{bmatrix} \to \neg_{12} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \quad \circ_{22} = \begin{bmatrix} 1 & 1 \\ 0 & 0 \end{bmatrix} \to \neg_{12} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}.$$

В свою очередь унарная операция $\neg_{12}$ имеет столбец, совпадающий по значениям с ее операндом. Следовательно, такая операция может быть опущена. Окончательно, после минимизации операций, получаем:

$$f(X) = (x_0 \circ_{00} x_1 \circ_{01} x_2) \times (x_3 \circ_{02} x_4) + (x_0 \circ_{10} x_1 \circ_{11} x_2) \times x_3 +$$
$$+ (x_0 \circ_{20} x_1 \circ_{21} x_2) \times (\neg_{22} x_3) + (x_0 \circ_{30} x_1 \circ_{31} x_2) \times (x_3 \circ_{32} x_4).$$

Таким образом, для вычисления 48 десятичных цифр числа π по модулю 3 требуется 18 операций. Заметим, что спектральная декомпозиция рассматриваемой функции приводит к синтезу формулы, содержащей 3 слагаемых и 15 операций ($M \approx 2{,}83$ при $\alpha = 1{,}094$). ♦

6.5. Заключительные замечания

Объединение алгебраических методов, разделительной декомпозиции и ортогональных разложений позволяет синтезировать эффективные формульные представления дискретных функций, реализуемые в виде последовательности операций (команд) некоторого вычислительного средства.

Однако при достаточно большой длине вектора почти все функции реализуются со сложностью, близкой к максимальной. Эффективным называется такое представление, которое содержит количество операций L, удовлетворяющее оценке [65]

$$L \le \frac{4}{n(k-\alpha)^2} \frac{\prod_{i=0}^{n-1} k_i}{\frac{1}{n^2}(\sum_{i=0}^{n-1} k_i - 2\beta)^2 - \rho^2}, \qquad (6.42)$$

где k – значность образующей алгебры, α – порождающая способность аналитической конструкции, β – начальная порождающая способность, ρ – параметр разделения переменных. Оценка (6.42) определяет количество операций, которых достаточно для реализа-

ции произвольной дискретной функции. Подстановочные значения для α, β и ρ вычисляются по методике, описанной ранее.

Если функция принадлежит множеству функций, требующих для своей реализации меньшее число операций, чем получаемое при алгебраическом синтезе, то возможна ее минимизация. При этом целесообразно использовать, в том числе и скобочные аналитические конструкций с произвольным порядком вхождения переменных. Если же функция принадлежит множеству функций с максимальной сложностью, то следует говорить о ее эффективном синтезе, для реализации которого достаточно бесповторных бесскобочных аналитических конструкций формул с фиксированным порядком вхождения переменных.

Последнее достижимо при алгебраической декомпозиции булевых функций (возможно, с небулевыми переменными) в аддитивной алгебре, осуществляемой по системам мультимодальных двузначных функций, а многозначных функций – только при декомпозиции в фундаментальной алгебре (конечных полях и коммутативных кольцах без делителя нуля), осуществляемой по системам мультимодальных многозначных функций. В свою очередь, в алгебре логики и в мультипликативной алгебре, где разложение осуществляется по унимодальным двузначным и многозначным функциям соответственно, получение эффективных представлений наперед неизвестной дискретной функции не представляется возможным.

Формула (6.42) позволяет по объему обрабатываемых данных и выполненному количеству операций L' оценить эффективность произвольной дискретной обработки данных, а в том случае, если количество операций L' будет меньше, чем L, то определить и степень минимизации математической модели, равную отношению L к L'.

Так, для вычислительного средства с разрядностью r эффективность обработки данных при достаточно большом их объеме может быть подсчитана по следующей формуле, непосредственно следующей из неравенства (6.42),

$$L \approx 4\frac{D_y}{D_x}2^{D_x-4r}, \quad E=\frac{L}{L'},$$

где L (L') – эффективное (реальное) количество операций в программе или количество элементов в устройстве обработки данных, D_x (D_y) – объем входных (выходных) данных в битах, r – разрядность процессора или разрядность логических элементов в битах, E – эффективность модели обработки данных.

Эффективный синтез осуществляется при алгебраической декомпозиции функции путем оптимального разделения переменных X на два непересекающихся множества X' и X'', таких что

$$\frac{1}{n}\left(\sum_{i=0}^{n'-1} k'_i - \sum_{j=0}^{n''-1} k''_j\right) \approx \rho_{opt},$$

где k'_i и k''_j – значности переменных из множеств X' и X'' соответственно, а искомая формула ищется в виде

$$f(X) = \sum_{i=0}^{M-1} g_i(X') \times h_i(X''),$$

где g_i и h_i – частичные функции, задаваемые бесповторными бесскобочными формулами, а M таково, что достигается выполнение условия (6.42).

При алгебраической декомпозиции, в отличие от разделительной, не требуется проверять все варианты разделения переменных. Минимизация числа ненулевых коэффициентов обеспечивается оптимальным их разделением, при котором гарантируется бесповторное представление системы частичных функций.

В отличие от алгебраического подхода, основанного на получении формулы функции и тождественных ее преобразованиях, при алгебраической декомпозиции возможен прямой синтез эффективного представления. Ортогональность алгебраического разложения и фундаментальность синтезируемых частичных функций позволяют применить хорошо разработанный аппарат линейной алгебры.

В итоге, при алгебраической декомпозиции, возможно получение эффективных формульных представлений дискретных функции. Для таких представлений найдены как точные оценки сложности синтезируемых формул, так и их асимптотические значения. Однако трудоемкость алгоритмов алгебраического синтеза, как это имеет место и при разложении функции по Карунену-Лоэву, достаточно высока.

Для решения прикладных задач большой размерности видится путь, заключающийся в использовании не общих, а частных декомпозиционных схем. Частная декомпозиционная схема, в отличие от общей, применима для оптимального или минимального синтеза не любых функций, а только принадлежащих некоторому ограниченному классу. В этом случае трудоемкость синтеза может быть сокращена до приемлемых на практике значений, но для каждого класса функций требуется строить свои декомпозиционные схемы, определяемые из содержательных представлений о предметной области и решаемых в ней задачах.

Выводы к Главе 6

1. Рассмотрены общие типы аналитических конструкций формул: повторные скобочные, повторные бесскобочные, бесповторные скобочные и бесповторные бесскобочные; и установлено, что при синтезе формул при алгебраической декомпозиции достаточно использовать бесповторную бесскобочную аналитическую конструкцию с фиксированным порядком вхождения переменных.

2. Найдено точное количество функций, которые порождаются бесповторной бесскобочной аналитической конструкцией, и показано, что по сравнению с бесповторной скобочной она имеет наибольшую выразительную способность, то есть порождает большее число функций.

3. Получено выражение оценки для порождающей способности бесповторной бесскобочной аналитической конструкции, где под порождающей способностью понимается количество различных дискретных функций, которые могут быть выражены в виде формулы.

4. Установлено, что полное использование порождающей способности бесповторной бесскобочной аналитической конструкции формул возможно только в аддитивных и фундаментальных алгебрах, в других алгебрах (алгебре логики и мультипликативной алгебре) эта конструкция является избыточной и не влияет на эффективность алгебраического синтеза.

5. Найдены точные, приближенные и асимптотические оценки сложности разложения и сложности представления функций при спектральном синтезе формул. Для приближенных значений получены оценки их точности.

6. Показано, что сложность представления функции при алгебраической декомпозиции минимальна, когда значности переменных стремятся к их среднему геометрическому.

7. Доказано, что наименьшая сложность представления функций алгебраической декомпозиции достигается при глубине вложенности скобок равной 2, то есть поиск эффективных формульных представлений необходимо осуществлять при двухступенчатой решающей декомпозиции в классе частичных подформул, описываемых бесповторной бесскобочной аналитической конструкцией.

8. Доказано, что при алгебраической декомпозиции не требуется проверять все варианты разделения переменных: минимизация сложности представления обеспечивается оптимальным их разделением, при котором гарантируется и бесповторное представление частичных функций. Найдены условия оптимального разделения переменных.

9. Определены точные, приближенные и асимптотические оценки сложности представления функций при алгебраическом синтезе формул. Для приближенных значений сложности найдены оценки их точности.

10. Показано, что при алгебраической декомпозиции в аддитивных и фундаментальных алгебрах асимптотическая оценка количества операций, необходимых для представления произвольной булевой функции, согласуется с наилучшими результатами, полученными другими исследователями.

11. Обоснована методика определения абсолютной эффективности и степени минимизации математических моделей дискретной обработки данных при конечной размерности задачи, которая основана на определении объемов обрабатываемых данных и количества выполняемых при этом операций.

12. Показано, что получение эффективных представлений дискретных функций сопряжено с высокой трудоемкостью процедуры поиска, необходимого для этого набора частичных функций. Для сокращения объема вычислений предложено на основе содержательного анализа предметной области строить частные декомпозиционные схемы применимые не для всех, а для некоторого ограниченного класса дискретных функций, описывающих решение заданного класса прикладных задач.

Заключение

В настоящей диссертации исследована проблема синтеза эффективного описания дискретной обработки данных для современных и перспективных вычислительных средств и, на основе полученных результатов, решена важная прикладная задача преобразования первичного высокоуровневого описания предметной области в терминах содержательной постановки задачи в ее низкоуровневое описание в виде последовательности команд (операций) вычислительного средства.

Основные результаты, представленные в диссертации, разделяются на следующие группы.

1) Предложена методология понятийного анализа, позволяющая получать эффективные декомпозиционные схемы предметной области в виде семантически замкнутых формальных спецификаций. Для этого:

– формализованы основные абстракции понятий и разработана формальная теория понятий, предназначенная высокоуровневого описания предметных областей и обладающая полнотой и непротиворечивостью;

– разработан протоязык для спецификации результатов понятийного анализа, предусматривающий выразительные средства для определения понятийной структуры, синтаксиса выражения понятий и многоаспектного описания семантики;

– найден универсальный метод описания семантики формальных языков на основе метода семантической индукции, осуществляемый посредством синтаксического и семантического замыкания понятийной модели и заключающийся в определении семантических категорий в процессе описания семантики и описанными ранее средствами.

2) Создана контекстная технология обработки данных на основе отражения понятийной структуры предметной области в понятиях создаваемого проблемного языка, а получаемых при понятийном анализе декомпозиционных схем – в его языковых конструкциях. При реализации системы контекстного программирования:

– доказано, что выразительные возможности языковых средств контекстной технологии обработки данных эквивалентны формализму контекстных грамматик;

– разработан метод разнесенного грамматического разбора, позволяющий выполнять эффективный анализ текста, порожденного неоднозначными и контекстными грамматиками;

– предложен метод определения прагматики понятийной модели в виде многоаспектного описания семантики, позволяющий, в том числе, для такого описания использо-

вать различные средства: от команд процессора целевой вычислительной платформы до текстов на целевом языке программирования.

3) Обобщена теория алгебраической декомпозиции дискретных функций на основе объединения алгебраических методов, разделительной декомпозиции и ортогональных разложений в широком спектре алгебраических систем. В частности:

– показано существование четырех типов образующих алгебр (алгебры логики, мультипликативной, аддитивной и фундаментальной алгебр), изучены их свойства и доказаны условия существования алгебраических разложений;

– найдены четыре класса частичных функций (унимодальные и мультимодальные, двузначные и многозначные), имеющие эффективную реализацию на вычислительных средствах с различными операционными возможностями;

– предложен метод многоступенчатого алгебраического синтеза формул, позволяющий учесть нетривиальные свойства декомпозируемой функции и получить более компактные формульные представления.

4) Разработан метод синтеза эффективных моделей дискретной обработки данных при двухступенчатой алгебраической декомпозиции дискретных функций, базирующийся на следующих результатах:

– выполнен количественный и качественный анализ порождающей способности аналитических конструкций формул и установлено, что для эффективного синтеза достаточно использовать наиболее простую аналитическую конструкцию бесповторных бесскобочных формул с фиксированным порядком вхождения переменных;

– доказано, что при алгебраическом синтезе не требуется проверять все варианты разделения переменных, минимизация сложности представления обеспечивается оптимальным их разделением, гарантирующим бесповторное бесскобочное представление функционального базиса;

– получены точные, приближенные и асимптотические оценки сложности синтезируемых формул и доказано, что количество операций, необходимое для представления дискретных функций, не хуже известных наилучших оценок;

– найдена методика оценки абсолютной эффективности и степени минимизации математических моделей дискретной обработки данных при конечной размерности решаемых задач, что позволяет сравнивать по эффективности различные модели.

Литература

1. *Агафонов В. Н.* Надъязыковая методология спецификации программ // Программирование. 1993. № 5. С. 28-48.

2. *Айдукевич К.* Картина мира и понятийный аппарат // Философия науки. Вып. 2: Гносеологические и методологические проблемы. М.: Институт философии РАН, 1996. С. 231-240.

3. *Александреску А.* Современное проектирование на C++: Обобщенное программирование и прикладные шаблоны проектирования. М.: Вильямс, 2004.

4. *Андерсон Р.* Доказательство правильности программ. М.: Мир, 1982.

5. *Артин Э.* Геометрическая алгебра. М.: Наука, 1969.

6. Архипова М. В. Генерация тестов для семантических анализаторов // Вычислительные методы и программирование. 2006. Т. 7. С.55-70.

7. *Ахманова О. С.* Словарь лингвистических терминов / Издание 2-е, стереотипное. М.: Едиториал УРСС, 2004.

8. *Ахмед Н., Рао К. Р.* Ортогональные преобразования при обработке цифровых сигналов. М.: Связь, 1980.

9. *Ахо А., Сети Р., Ульман Д.* Компиляторы. Принципы, технологии, инструменты. М.: Вильямс, 2001.

10. *Ахо А., Ульман Д.* Теория синтаксического анализа, перевода и компиляции. М.: Мир, 1978.

11. *Баранов С. Н., Ноздрунов Н. Р.* Язык Форт и его реализации. Л.: Машиностроение, 1988.

12. *Барвайс Дж.* Введение в логику первого порядка. Справочная книга по математической логике. Ч.1. М.: Наука, 1982.

13. *Барвайс Дж.* Теория множеств. Справочная книга по математической логике. Ч.2. М.: Наука, 1982.

14. *Барендрегт Х.* Ламбда-исчисление, его синтаксис и семантика. М.: Мир, 1985.

15. *Барулин А. Н.* Основания семиотики. Знаки, знаковые системы, коммуникация. Ч. 1. Базовые понятия. Эволюционная теория происхождения языка. М.: Спорт и культура, 2002.

16. *Башмаков А. И., Башмаков И. А.* Интеллектуальные информационные системы: Учеб. пособие. М.: Изд-во МГТУ им Н.Э. Баумана, 2005.

17. *Бездушний А. Н., Лютый В. Г., Серебряков В. А.* Разработка компиляторов в системе СУПЕР. М.: ВЦ АН СССР, 1991.

18. Бениаминов Е. М. Алгебраические методы в теории баз данных и представлении знаний. М.: Научный мир, 2003.

19. *Бердяев Н.А*. Творчество и объективация. М.: Экономпресс, 2000.

20. *Бибило П. Н.* Декомпозиция булевых функций (обзор) // В кн. Проектирование устройств логического управления. М.: Наука, 1985. С. 106-126.

21. *Блох А. Ш.* Канонический метод синтеза контактных схем // АиТ. 1961. № 6.

22. Большая советская энциклопедия. 3-е изд. 1969-1978 гг.

23. Большой психологический словарь / Сост. Мещеряков Б., Зинченко В. М.: Олма-пресс. 2004.

24. Большой энциклопедический словарь. М., 1998.

25. *Борщев В.Б., Хомяков М.В*. Аксиоматический подход к описанию формальных языков // В сб. Математическая лингвистика. Под ред. С.К. Шаумяна. М.: Наука, 1973. С. 5-47.

26. *Борщев В. Б.* Естественный язык – наивная математика для описания наивной картины мира // Московский лингвистический альманах. 1996. № 1. С. 203-225.

27. *Братко И*. Алгоритмы искусственного интеллекта на языке PROLOG. М.: Вильямс, 2004.

28. *Братчиков И. Л.* Синтаксис языков программирования. М.: Наука, 1975.

29. *Бронштейн И. Н., Семендяев К. А.* Справочник по математике для инженеров и учащихся втузов. М.: Наука, 1980.

30. *Булос Дж., Джеффри Р*. Вычислимость и логика. М.: Мир, 1994.

31. *Бурбаки Н*. Теория множеств. Структуры. М.: Мир, 1965.

32. *Буч Г.* Объектно-ориентированное проектирование с примерами применения. М.: Конкорд, 1992.

33. *Вагин В. Н., Еремеев А. П.* Некоторые базовые принципы построения интеллектуальных систем поддержки принятия решений реального времени // Изв. РАН. ТиСУ. 2001. № 6. С. 114-123.

34. *Вагин В. Н., Головина Е. Ю., Загорянская А. А. Фомина М. В*. Достоверный и правдоподобный вывод в интеллектуальных системах. М.: Физматлит, 2004.

35. *Ван Хао, Р.Мак-Нотон*. Аксиоматические системы теории множеств. М.: Изд. Иностранной литературы, 1963.

36. *Вайнгартен Ф*. Трансляция языков программирования. М.: Мир, 1977.

37. *Варламов О. О*. Эволюционные базы данных и знаний для адаптивного синтеза интеллектуальных систем. Миварное информационное пространство. М.: Радио и связь, 2002.

38. *Васильев Н. А.* Воображаемая логика. Избранные труды. М.: Наука, 1989. С. 126-131.

39. *Васильев С. Н., Жерлов А. К., Федосов Е. А., Федунов Б. Е.* Интеллектное управление динамическими системами. М.: Физико-математическая литература, 2000.

40. *Васюков В. Л.* Формальная феноменология. М.: Наука, 1999.

41. *Величковский Б. М.* Когнитивная наука: Основы психологии познания. В 2-х томах. М.: Академия, 2006.

42. *Виленкин Н.Я.* Класс полностью ортогональных систем // Известия Академии наук СССР. 1947. Сер. Математика. № 11. С. 363-400.

43. *Вирт Н.* Систематическое программирование. Введение. М.: Мир, 1977.

44. *Вирт Н.* Алгоритмы и структуры данных. М.: Мир, 1989.

45. *Виттих В. А.* Интеграция знаний при исследованиях сложных систем на основе инженерных теорий // Известия РАН. Теория и системы управления. 1998. № 5.

46. *Вольфенгаген В .Э.* Конструкции языков программирования. Приемы описания. – М.: Центр ЮрИнфоР, 2001.

47. *Вольфенгаген В .Э.* Комбинаторная логика в программировании. Вычисления с объектами в примерах и задачах. М.: Центр ЮрИнфоР, 2003.

48. *Вольфенгаген В. Э.* Методы и средства вычислений с объектами. Аппликативные вычислительные системы. М.: Центр ЮрИнфоР, 2004.

49. *Вудс В. А.* Сетевые грамматики для анализа естественных языков // Кибернетический сборник. Вып. 13. М.: Мир, 1978. С. 120-158.

50. *Выхованец В. С.* Дискретное преобразование Фурье и его применение при логических вычислениях / Приднестровский гос.-корпорат. унив. им. Т.Г. Шевченко. Тирасполь, 1997. 18 с. Деп. в ВИНИТИ 05.06.97, № 1852-В97.

51. *Выхованец В. С.* Кратные логические вычисления и их применение при моделировании дискретных объектов / Автореферат дис. на соиск. уч. ст. канд. техн. наук. М., 1998. 23 с.

52. *Выхованец В. С.* Обобщенные полиномиальные формы // Радиоэлектроника. Информатика. Управление. 1999. № 2. С. 55-59.

53. *Выхованец В. С.* Булевы мультипликативные формы // Тезисы докладов международной научно-практической конференции «Математические методы в образовании, науке и промышленности». Тирасполь, 1999. С. 52-53.

54. *Выхованец В. С.* Контекстная технология программирования // Труды IV Международной научно-технической конференции по телекоммуникациям (Телеком-99). Одесса, 1999. С. 116-119.

55. *Выхованец В. С.* Асимптотические оценки при идентификации дискретных объектов // Материалы международной конференции «Идентификация систем и задачи управления» М., 2000. Электрон. опт. диск. ISBN 5-201-09605-0.

56. *Выхованец В. С.* Асимптотические оценки в многозначной логике // Материалы юбилейной конференции профессорско-преподавательского состава, посвященной 70-летию Приднестровского государственного университета им. Т.Г. Шевченко. Тирасполь, 2000. С. 264-270.

57. *Выхованец В. С.* Язык контекстного программирования // Тезисы докладов 8-й Международной конференции «Проблемы управления безопасностью сложных систем». М., 2000. Т. 2. С. 89-91.

58. *Выхованец В. С.* О вычислимости конечных полей // Материалы международной научно-практической конференции «Математическое моделирование в образовании, науке и производстве». Тирасполь, РИО ПГУ, 2001. С. 475-477.

59. *Выхованец В.С.* Спектральные методы в логической обработке данных // АиТ. 2001. № 10. С. 28-53.

60. *Выхованец В. С.* Аддитивная алгебра в цифровой обработке сигналов // Доклады 4-й Международной конференции и выставки «Цифровая обработка сигналов и ее применение». М., 2002. Т. 2. С. 255-258.

61. *Выхованец В. С.* Алгебраическая декомпозиция дискретных функций в аддитивной алгебре // Теория и практика логического управления. Тезисы докладов Международной конференции, посвященной 100-летию со дня рождения члена-корреспондента АН СССР М. А. Гаврилова. М, 2003. С. 81-84.

62. *Выхованец В. С.* Хааро-подобные системы сигналов // Доклады 5-й Международной конференции и выставки «Цифровая обработка сигналов и ее применение». М., 2003. Т. 2. С. 279-283.

63. *Выхованец В. С.* Разнесенный грамматический разбор // Проблемы управления. 2006. № 1. С. 32-43.

64. *Выхованец В. С.* Алгебраическая декомпозиция дискретных функций // Автоматика и телемеханика. 2006. № 3. С. 20-56.

65. *Выхованец В. С.* Оптимальный синтез логического управления // Тезисы докладов 3-ой Международной конференции по проблемам управления. М., 2006. Т. 2. С. 105.

66. *Выхованец В. С., Гордиенко К. П.* Процессор с сокращенным набором команд // Вестник Приднестровского университета. Тирасполь, 1995. № 1. С. 124-128.

67. *Выхованец В. С., Иосенкин В. Я.* Высокоуровневая форма синтеза систем управления // Тезисы докладов 2-ой Международной конференции по проблемам управления. М., 2003. Т. 2. С. 109.

68. *Выхованец В. С., Иосенкин В. Я.* Компиляция знаний, представленных на языке ESSE // Тезисы докладов 2-ой Международной конференции по проблемам управления. М., 2003. Т. 2. С. 165.

69. *Выхованец В. С., Иосенкин В. Я.* Понятийный анализ и контекстная технология программирования // Проблемы управления. 2004. №.4. С. 3-25.

70. *Выхованец В. С., Малюгин В. Д.* Спектральные методы в логическом управлении // Труды 2-й Международной научно-технической конференции «Современные методы цифровой обработки сигналов в системах измерения, контроля, диагностики и управления». Минск, 1998. С. 56-59.

71. *Выхованец В. С., Малюгин В. Д.* Кратные логические вычисления // Автоматика и телемеханика. 1998. № 6. С. 163-171.

72. *Выхованец В. С., Малюгин В. Д.* Мультипликативные формы и их применение при логических вычислениях // Тезисы докладов 2-ой Международной конференции по проблемам управления. М., 2003. Т. 2. С. 110-111.

73. *Выхованец В. С., Малюгин В. Д.* Аппаратная и программная реализация мультипликативных форм // Теория и практика логического управления. Тезисы докладов Международной конференции, посвященной 100-летию со дня рождения члена-корреспондента АН СССР М. А. Гаврилова. М., 2003. С. 79-81.

74. *Выхованец В. С., Малюгин В. Д.* Мультипликативная алгебра и ее применение в логической обработке данных // Проблемы управления. 2004. № 3. С. 67-77.

75. *Жегалкин И.И.* О технике вычисления предложений в символической логике // Математический сборник. 1927. Т. 43. С. 9-28.

76. *Замулин А. В.* Алгебраическая семантика императивного языка программирования // Программирование. 2003. № 6. С. 51-64.

77. *Замулин А. В.* Абстрактная модель компилятора как результат алгебраической семантики языка программирования. //Программирование. 2004. № 5. С. 69-80.

78. *Гаврилов М. А.* Релейно-контактные схемы с вентильными элементами // Изв. АН СССР. ОТН. 1945. № 3. С. 153-164.

79. *Гаврилов М. А.* Методы синтеза релейно-контактных схем. // Электричество. 1946. № 2. С. 54-59.

80. *Гаврилова Т. А., Хорошевский В. Ф.* Базы знаний интеллектуальных систем. СПб.: Питер, 2000.

81. *Гаскаров Д. В.* Интеллектуальные информационные системы. М.: Высш. шк., 2003.

82. *Гинзбург С.* Математическая теория контекстно-свободных языков. М.: Мир, 1970.

83. *Гладкий А.В.* Формальные грамматики и языки. М.: Наука, 1973.

84. *Гладкий А.В.* Математическая логика. М.: Российск. гос. гуманит. ун-т, 1998.

85. *Голдблатт Р.* Топосы. Категорный анализ логики. М.: Мир, 1983.

86. *Горбатов В.А.* Синтез логических схем в произвольном базисе // В кн. Теория дискретных автоматов. Рига: Зинатне, 1967.

87. *Горовой В.Р.* Синтез релейных структур методом замены выходных функций // АиТ. 1967. № 1. С. 112-121.

88. *Горский Д. П.* Вопросы абстракции и образования понятий. М.: Изд-во АН СССР, 1961.

89. ГОСТ 34.320-96. Информационные технологии. Система стандартов по базам данных. Концепции и терминология для концептуальной схемы и информационной базы.

90. *Грис Д.* Конструирование компиляторов для цифровых вычислительных машин. М.: Мир, 1975.

91. *Дантеман Д., Мишел Д., Тейлор Д.* Программирование в среде Delphi. Киев: Диасофт, 1995.

92. *Данильян О. Г., Панова Н. И.* Современный словарь по общественным наукам. М.: Эксмо-Пресс, 2005.

93. *Девятков В. В.* Онтологии и проектирование систем // Приборы и системы. Управление, контроль, диагностика. 2000. № 1.

94. *Девятков В. В.* Системы искусственного интеллекта. – М.: Изд-во МГТУ им. Н.Э. Баумана, 2001.

95. *Демьянков В. З.* Доминирующие лингвистические теории в конце XX века // Язык и наука конца 20 века. М.: Институт языкознания РАН, 1995. С.239-320.

96. *Джексон П.* Введение в экспертные системы / Пер. с англ. М.: Издательский дом «Вильямс», 2001.

97. *Дубровский Д. И.* Проблема идеального. Субъективная реальность. М.: Канон, 2002.

98. *Евреинов Э. В., Косарев Ю. Г.* Однородные универсальные вычислительные системы высокой производительности. Новосибирск: Наука. 1966.

99. *Еремеев А. П.* Проектирование интеллектуальный систем принятия/поддержки решений в инструментальной среде G2 // Перспективные информационные технологии и интеллектуальные системы. 2000. № 2 (http://pitis.tsure.ru).

100. *Закревский А. Д.* Логические уравнения. Минск: Наука и техника, 1975.

101. *Закревский А. Д.* Алгоритм декомпозиции булевых функций // Труды Сибирского физико-технического института. 1964. Вып. 44. С. 5-16.

102. *Захаров В.Н.* Автоматы с распределенной памятью. М.: Энергия, 1975.

103. *Зинченко В. П.* Живое знание: психологическая педагогика. Самара, 1998.

104. *Зубков С. В*. Ассемблер для DOS, Windows и Unix. М.: ДМК, 2004.

105. *Зыков С. В.* Концепция и методология интегрированного проектирования корпоративных информационных систем для глобальной среды вычислений // Сборник докладов Первой Международной научно-практической конференции «Современные информационные технологии и ИТ-образование». М., Изд-во МГУ им. М.В.Ломоносова, 2005. С. 411-427.

106. *Иосенкин В. Я., Выхованец В. С.* Технология контекстного программирования в экономике и бизнесе // Материалы межвузовской научно-технической конференции «Управляющие и вычислительные системы. Новые технологии». Вологда: ВоГТУ, 2001. С. 180-181.

107. *Иосенкин В. Я., Выхованец В. С.* Технология контекстного программирования в телекоммуникационных системах // Труды второй международной научно-практической конференции «Современные информационные и электронные технологии». Одесса: Друк, 2001. С. 118-119.

108. *Иосенкин В. Я., Выхованец В. С.* Применение технологии контекстного программирования для решения больших прикладных задач // Труды международной конференции «Параллельные вычисления и задачи управления». Москва: Институт проблем управления, 2001. С. (4)-121-139. Электрон. опт. диск. ISBN 5-201-09559-3.

109. *Иосенкин В. Я., Выхованец В. С.* Контекстное программирование в моделировании // Материалы межд. науч.-практ. конф. «Моделирование. Теория, методы и средства». Новочеркасск: УПЦ “Набла” ЮРГТУ(НПИ), 2001. Ч. 7. С. 49-51.

110. *Иосенкин В. Я., Выхованец В. С.* Формализация семантики искусственных языков // Материалы международной научно-практической конференции «Математическое моделирование в образовании, науке и производстве». Тирасполь, РИО ПГУ, 2001. С. 475-477.

111. *Иосенкин В. Я., Выхованец В. С.* Контекстная модель технологического процесса предприятия // Труды II международной конференции «Идентификация систем и задачи управления» (SICPRO'03). М., 2003. С. 859-871. ISBN 5-201-14948-0.

112. Искусственный интеллект: В 3-х кн. Кн. 2. Модели и методы: Справочник / Под ред. Д.А. Поспелова. М.: Радио и связь, 1990.

113. Искусственный интеллект: В 3-х кн. Кн. 3. Программные и аппаратные средства: Справочник / Под ред. В.Н. Захарова, В.Ф. Хорошевского. М.: Радио и связь, 1990.

114. *Калашян А.Н., Калянов Г.Н.* Структурные модели бизнеса: DFD-технологии. М.: Финансы и статистика, 2003.

115. *Кант И.* Критика чистого разума. М.: Мысль, 1994.

116. *Кантор Г.* Труды по теории множеств. М.: Наука, 1985.

117. *Карповский М.Г., Москалев Э.С.* Реализация частично-определенных функций алгебры логики с помощью разложения в ортогональные ряды // АиТ. 1970. № 8.

118. *Карри Х. Б* Основания математической логики. М.: Мир, 1969.

119. *Карнап Р.* Значение и необходимость. М.: Мир, 1959.

120. *Карпенко А. С.* Логика на рубеже тысячелетий // Логические исследования. Вып. 7. М.: Наука, 2000. С. 7-60.

121. *Карпенко А. С.* Предмет логики в свете основных тенденций ее развития // Логические исследования. Вып. 11. М.: Наука, 2004. С. 149-172.

122. *Касьянов В. И., Поттосин И. В*. Методы построения трансляторов. Новосибирск: Наука, 1986.

123. *Катречко С. Л.* Формальная и трансцендентальная логика // Тезисы докл. межд. конф. «Современная логика». СПб, 2004.

124. *Киндлер Е.* Языки моделирования. М.: Энергия, 1985.

125. *Клини С.К.* Математическая логика. М., Мир, 1973.

126. *Кнут Д.* Семантика контекстно-свободных языков // В сб.: Семантика языков программирования. М.: Мир, 1980.

127. *Кондаков Н. И.* Логический словарь. М., 1971.

128. *Кондрашина Е. Ю., Литвинцева Л. А., Поспелов Д. А*. Представление знаний о времени и пространстве в интеллектуальных системах / Под ред. Д. А. Поспелова. М.: Наука, 1989.

129. *Конноли Т., Бегг К.* Базы данных: проектирование, реализация и сопровождение. Теория и практика. М.: Вильямс, 2003.

130. *Костогрызов А. И., Липаев В. В*. Сертификация качества функционирования автоматизированных информационных систем. М.: Изд-во «Вооружение. Политика. Конверсия», 1996.

131. *Кравцов Л. Г.* Методологические проблемы психологического анализа мышления в понятиях // Материалы Первой российской конференции по когнитивной науке. Казань, Казанский гос. ун-т, 2004 (http://www.ksu.ru/ss/cogsci04/).

132. Краткий психологический словарь / Под общ. ред. А.В Петровского, М.Г. Ярошевского. Ростов-на-Дону: Феникс, 1999.

133. *Крицкий С. П.* Аксиоматическое описание контекстных связей и условий // Программирование. 1980. № 6.

134. *Кузин С. Г.* Ролевый граф в качестве модели понятия // Вестник Нижегородского университета: Математическое моделирование и оптимальное управление. Н. Новгород: Изд-во Нижегородского университета, 1998. № 2 (19). С. 224-235.

135. *Кузнецов А. В.* О бесповторных контактных схемах и бесповторных суперпозициях функций алгебры логики // Труды Матем. ин-та им. В. А. Стеклова АН СССР. 1958. Т. 51. С. 186-225.

136. *Кузнецов В. И.* Двухместные и трехместные отношения между научными понятиями // Logical Studies. 2004. № 12. С. 1-24. (http://www.logic.ru/Russian/LogStud/).

137. *Кузнецов В. И.* Изоляционистский и экологический подходы к моделированию понятий // Материалы IX научной конференции «Современная логика: проблемы теории, истории и применения в науке». СПб.: Санкт-Петербург. гос. ун-та, 2006. С. 50-52.

138. *Кузнецов О. П.* О программной реализации логических функций и автоматов // Автоматика и телемеханика. 1977. № 7. С. 163-174. № 9. С. 137-139.

139. *Кузнецов О. П., Адельсон-Вельский Г. М.* Дискретная математика для инженера. М.: Энергоатомиздат, 1988.

140. *Кузнецов О. П.* Дискретная математика для инженера. М.: Лань, 2005.

141. *Кузьмин В. А.* Оценка сложности реализации функций алгебры логики простейшими видами бинарных программ // Методы дискретного анализа в теории кодов и схем. Новосибирск: 1976. Вып. 29. С. 24-36.

142. *Кун Т.* Структуры научных революций. М.: Прогресс, 1977.

143. *Кухарев Г. А., Шмерко В. П., Янушкевич С. П.* Техника параллельной обработки бинарных данных на СБИС. Минск: Выш. шк., 1991.

144. *Лагута О. Н.* Логика и лингвистика. Новосибирск, 2000.

145. *Лапкин Л.Я.* О векторной программной реализация логических функций // АиТ. 1983. № 3. С. 120-128.

146. *Лейбниц Г. В.* Сочинения: В 4-х т. Т. 3. М.: Мысль, 1984.

147. *Леоненков А. В.* Самоучитель UML. СПб.: BHV, 2006.

148. *Лидл Л., Нидеррайтер Г.* Конечные поля: В 2-х т. Т. 1. М.: Мир, 1988.

149. *Лисичкина Н. В.* Проблемы современной логической теории понятий // Матер. Третьего Российского философского конгресса «Рационализм и культура на пороге III тысячелетия». Ростов-на-Дону, 2002.

150. *Липаев В. В.* Качество программных средств. М.: Эдиториал УРСС, 2002.

151. Логический подход к искусственному интеллекту / Под ред. Гаврилова Г. П. М.: Мир,1990.

152. *Лозовский В. С.* Сетевые модели // Искусственный интеллект. В 3х кн. Кн.2. Модели и методы: Справочник / Под ред. Д. А. Поспелова. М.: Радио и связь, 1990.

153. *Лорьер Ж.-Л.* Системы искусственного интеллекта. М.: Мир, 1991.

154. *Лупанов О.Б.* Об одном методе синтеза схем // Известия высших учебных заведений. Радиофизика. 1958. № 1. С. 120-140.

155. *Лупанов О.Б.* О синтезе некоторых классов управляющих систем. М.: Физматгиз, 1963. Вып. 10. С. 63-97.

156. *Люгер Д. Ф.* Искусственный интеллект: стратегии и методы решения сложных проблем. М.: Вильямс, 2005.

157. *Майерс Г.* Архитектура современных ЭВМ: В 2-х книгах. Кн. 1. М.: Мир, 1985.

158. Макетирование, проектирование и реализация диалоговых информационных систем / Под ред. Е. И. Ломако. М.: Финансы и статистика, 1993.

159. *Макклеллан Дж.Х., Рейдер Ч.М.* Применение теории чисел в цифровой обработке сигналов. Москва: Радио и связь, 1983.

160. *Мальковский М. Г.* Языковой процессор системы TULIPS-2 // Докл. Второй Всесоюзная конференция по созданию машинного фонда русского языка. М.: ИРЯз АН СССР, 1987. С. 176-204.

161. *Малюгин В.Д.* О полиномиальной реализации кортежа булевых функций // ДАН СССР. 1982. Т. 265. № 6. С. 1338-1341.

162. *Малюгин В. Д.* Реализация кортежей булевых функций посредством линейных арифметических полиномов // Автоматики и телемеханика. 1984. № 2. С. 114-122.

163. *Марка Д.А., Мак-Гоуэн К.* Методология структурного анализа и проектирования. М.: Мета Технология, 1993.

164. *Марков А. А.* Теория алгорифмов. М.: Изд-во АН СССР, 1954.

165. Математическая энциклопедия: В 4-х т. М.: Советская энциклопедия, 1977-1985.

166. *Мендельсон Э.* Введение в математическую логику. М.: Наука, 1976.

167. *Мерекин Ю. В.* Арифметические формы записи булевых выражений и их применение для расчета надежности схем // Вычислительные системы. Вып. 7. Новосибирск: Ин-т математики СО АН СССР, 1963. С. 13-23.

168. *Морага К., Сасао Т., Станкович Р.* Обобщенный подход к диаграммам решений с нагруженными ребрами и диаграммам решений для арифметического преобразования. // Автоматика и телемеханика. 2002. № 6. С. 140-153.

169. *Моррис Ч.* Основания теории знаков // В кн.: Семиотика / Под ред. Ю. С. Степанова. М.: Наука, 1983.

170. *Маторин С. И.* О новом методе системологического анализа, согласованном с процедурой объектно-ориентированного проектирования // Кибернетика и системный анализ. 2002. №1. С. 118-130.

171. *Назаренко Г. И., Осипов Г. С.* Основы теории медицинских технологических процессов. Часть1.-М.: ФИЗМАТЛИТ, 2005.

172. *Нариньяни А. С.* Недоопределенность в системе представления и обработки знаний // Техничеся кибернетика. 1986. № 5.

173. *Непейвода Н. Н.* Прикладная логика. Новосибирск, 1999.

174. *Непейвода Н. Н., Скопин И. Н.* Основания программирования. М.: РХД, 2003.

175. *Никаноров С. П.* Характеристика и область применения метода концептуального проектирования систем организационного управления. // Концептуальное проектирование систем организационного управления и его применение в капитальном строительстве: Сб. науч. тр. М.: ЦНИИЭУС Госстроя СССР, 1989. С. 8-29.

176. *Никаноров С. П., Персиц Д. Б.* Метод формального проектирования целостных систем организационного управления // В сб.: Рефераты докладов Международного симпозиума по проблемам организационного управления и иерархическим системам. Ч. 1. М., ИПУ АН СССР, 1972.

177. *Ничепорук Э. И.* О синтезе схем с помощью линейных преобразований переменных // Докл. АН СССР. 1958. Т. 123. Вып. 4. С. 610-612.

178. Новейший философский словарь: 3-е изд., исправл. Мн.: Книжный Дом. 2003.

179. *Осипов Г. С.* От ситуационного управления к прикладной семиотике // Новости искусственного интеллекта. 2002. №6. С.56-59.

180. *Пархоменко П. П.* Синтез релейных структур на различных функционально полных системах логических элементов // АиТ. 1964. № 6.

181. *Перекрестенко А.* Разработка системы автоматического синтаксического анализа на основе мягко контекстно-зависимой унификационной грамматики // Тр. конф. «Диалог-2004». 2004.

182. Пересмотренное сообщение об Алголе 68 / Ред. А. Ван Вейнгаарден, Б.Майу, Дж.Пек, К.Костер, М.Синцов, Ч.Линдси, Л.Меертенс, Р.Фискер. М.: Мир, 1979.

183. *Перетятькин М. Г.* Конечно аксиоматизируемые теории. Новосибирск: Научная книга, 1997.

184. *Першиков В. И., Савинков В. М.* Толковый словарь по информатике. М.: Финансы и статистика, 1991.

185. *Петренко А*. Венский метод разработки программ (неформальное введение) // Программирование. 1991. № 6. С. 23-27.

186. *Пильщиков В. Н.* Язык ПЛЭНЕР. М.: Наука, 1983.

187. *Пирс Ч. С.* Избранные философские произведения. М.: Логос, 2000.

188. *Пирс Ч. С.* Логические основания теории знаков. СПб.: Лаборатория метафизических исследований философского факультета СпбГУ; Алетейя, 2000.

189. *Плесневич Г. С.* Силлогистики для семантических сетей // Новости искусственного интеллекта. 2004. № 3.

190. *Поваров Г. Н.* О функциональной разделимости булевых функций // Докл. АН СССР. 1954. Т. 94. С. 801-803.

191. *Поваров Г. Н.* Математическая теория синтеза контактных (1,k)-полюсников // Докл. АН СССР. 1955. Т. 5.

192. *Поспелов Д. А.* Логико-лингвистические модели в системах управления. М.: Энергоиздат, 1981.

193. *Поспелов Д. А.* Ситуационное управление: теория и практика. М.: Наука, 1986.

194. *Поспелов Д. А.* Моделирование рассуждений. Опыт анализа мыслительных актов. М.: Радио и связь, 1989.

195. *Поспелов Д. А.* Где исчезают виртуальные миры? // Новости искусственного интеллекта. 2003. № 3. С. 9-25.

196. *Поспелов Д. А., Осипов Г. С.* Прикладная семиотика // Новости искусственного интеллекта. 1999. № 1. С. 9-35.

197. *Поспелов Д. А*. Семиотические модели: успехи и перспективы // Кибернетика. 1976. № 6. С.114-123.

198. *Поспелов Д. А., Пушкин В. Н.* Мышление и автоматы. М.: Сов. радио, 1972.

199. Поиск подходов к решению проблем / Прангишвили И. В., Абрамова Н. А., Спиридонов В. Ф. и др. М.: Синтег, 1999.

200. *Пратт Т., Зелковиц М.* Языки программирования: разработка и реализация. СПб.: Питер, 2002.

201. Программирование в ограничениях и недоопределенные модели / Нариньяни А. С., Телерман В. В., Ушаков Д. М., Швецов И. Е. // Информационные технологии, 1998. №7.

202. *Пушкин В. Н.* Эвристика – наука о творческом мышлении. М., 1967.

203. *Рамбо Дж., Буч Г., Якобсон А.* UML. Специальный справочник. СПб.: Питер, 2002.

204. *Расёва Е., Сикорский Р.* Математика метаматематики. М.: Наука, 1972.

205. *Рейуорд-Смит В.* Теория формальных языков. Вводный курс. М.: Радио и связь, 1988.

206. *Рыков В. В.* Корпусная лингвистика. Научно-аналитический обзор // Социальные и гуманитарные науки: Зарубежная литература. М.: ИНИОН, 1996. № 4. С. 43-51.

207. *Себеста Р. У.* Основные концепции языков программирования. М.: Вильямс, 2001.

208. *Серебряков В. А., Галочкин М. П.* Основы конструирования компиляторов. М.: Едиториал УРСС, 2003.

209. Система ALEX как средство для многоцелевой автоматизированной обработки текстов / И.С. Кононенко и др. // Тр. межд. сем. «Диалог-2002». Протвино, 2002. Т.2. С. 192-208.

210. *Смирнов С. В.* Онтологический анализ предметных областей // Известия Самарского научного центра РАН. 2001. Т. 3. № 1. С. 62-98.

211. *Смирнова Е. Д.* Формализованные языки и проблемы логической семантики. М., 1982.

212. *Смит Дж., Смит Д.* Принципы концептуального проектирования баз данных / В сб.: Требования и спецификации в разработке программ / Пер. с англ. под ред. В.Н. Агафонова. М.: Мир, 1984. С. 165-198.

213. Социология: Энциклопедия / Сост. А.А. Грицанов, В.Л. Абушенко, Г.М. Евелькин, Г.Н. Соколова, О.В. Терещенко. Мн.: Книжный Дом, 2003. 1312 с.

214. Стандарт онтологического исследования IDEF5 (www.idef.com/idef5.html).

215. *Столл Р.* Множества. Логика. Аксиоматические теории. М.: Просвещение, 1968.

216. *Столлингс В.* Операционные системы. М.: Издательский дом «Вильямс», 2002.

217. *Тамм Б. Г., Пуусепп М. Э., Таваст Р. Р.* Анализ и моделирование производственных систем. М.: Финансы и статистика, 1987.

218. *Тарский А.* Введение в логику и методологию дедуктивных наук. М.: Изд-во иностр. лит., 1948.

219. Тестирование на основе моделей / Петренко А., Бритвина Е., Грошев С. И и др. // Открытые системы. 2003. № 9.

220. *Тузов В. А.* Семантический анализ текста на русском языке: функциональная модель предложения // Экономико-математические исследования: математические модели и информационные технологии. СПб.: Наука, 2003. Вып. 3. С. 304–328.

221. *Турчин В. Ф.* Метаалгоритмический язык // Кибернетика. 1968. № 4. С. 45-54.

222. *Турчин В. Ф.* Базисный РЕФАЛ. Описание языка и основные приемы программирования. М.: ЦНИПИАСС, 1974.

223. *Тыугу Э. Х.* Концептуальное программирование. М.: Наука, 1984.

224. *Успенский В.А.* Машина Поста. М.: Наука, 1988.

225. *Фараджиев Р. Г., Ципкин Я. З.* Преобразования Лапласа-Галуа в теории последовательных машин // Доклады Академии наук СССР. 1966. Т. 166 № 36.

226. *Филд А., Харрисон П.* Функциональное программирование. М.: Мир, 1993.

227. *Финько О. А.* Модулярная арифметика параллельных логических вычислений. Краснодар, Краснодарский военный институт, 2003.

228. Формальные спецификации в технологиях обратной инженерии и верификации программ / Бурдонов И.Б., Демаков А.В., Косачев А.С. и др. // Труды Института системного программирования. 1999. № 1. С. 31-43.

229. *Фреге Г.* Избранные работы: Пер. с нем. А.Л. Никифорова. М.: ДИК, Русское феноменологическое общество, 1997.

230. *Френкель А., Бар-Хиллел И.* Основания теории множеств. М., 1966.

231. *Фролов С. С.* Социология: Учебник. М.: Наука, 1994.

232. *Хантер Р.* Основные концепции компиляторов. М.: Вильямс, 2002.

233. *Хомский Н.* Три модели описания языка // Кибернетический сборник. 1961. Вып. 2. С. 237-266.

234*Хомский Н.* Формальные свойства грамматик // Кибернетический сборник. 1965. Вып. 1. С. 121-227.

235. *Хомский Н., Миллер Дж.* Введение в формальный анализ естественных языков // Кибернетический сборник. 1965. Вып. 1. С. .231-290.

236. *Хорошевский В. Ф.* ATNL-машина – вопросы программной и аппаратной реализации // Сборник трудов I-го симпозиума ИФАК по искусственному интелекту. Ленинград, 1983.

237. *Цейтин Г. С.* О соотношении естественного языка и формальной модели // Вопросы кибернетики. М., 1982. С. 20-34.

238. *Цейтин Г. С.* Программирование на ассоциативных сетях // ЭВМ в проектировании и производстве. Л.: Машиностроение, 1985. Вып. 2. С. 16-48.

239. *Чебурахин И. Ф.* Синтез дискретных систем и математическое моделирование. М.: Физматлит, 2004.

240. *Чен П.* Модель «сущность-связь» – шаг к единому представлению данных // СУБД. 1995. № 3. С. 137-158.

241. *Черемных С. В., Семенов И. О., Ручкин В. С.* Структурный анализ систем: IDEF-технологии. М.: Финансы и статистика, 2001.

242. *Черч А.* Введение в математическую логику. М.: Из-во иностр. лит-ры, 1959.

243. *Шагурин И .И, Бердышев Е. М.* Процессоры семейства INTEL P6: Pentium II, Pentium III, Celeron и др. Архитектура, программирование, интерфейс. / Справочник. М.: Радио и связь, 2000.

244. *Шалыто А. А.* SWITCH-технология. Алгоритмизация и программирование задач логического управления. СПб.: Наука, 1998.

245. *Шалыто А. А.* Логическое управление. Методы аппаратной и программной реализации алгоритмов. СПб.: Наука, 2000.

246. *Шамис А. Л.* Поведение, восприятие, мышление: проблемы создания искусственного интеллекта. М.: Едиториал УРСС, 2005.

247. *Шаров С. А.* Средства компьютерного представления лингвистической информации. М.: Российский научно-исследовательский институт искусственного интеллекта, 1996.

248. *Шаров С. А.* О различии между онтологией языка и онтологией предметной области // Тр. 6-й Российской Национальной Конференции по искусственному интеллекту. Пущино, 1998. С. 41-49.

249. *Шеннон К. Э.* Работы по теории информации и кибернетике. М.: Изд-во иностр. лит., 1963.

250. *Эббинхауз Г.Д., Якобс К., Ман Ф.-К., Хермес Г.* Машины Тьюринга и рекурсивные функции. М., 1972.

251. *Эдельман С. Л.* Математическая логика. М., Наука, 1975.

252. *Эйнштейн А.* Физика и реальность. М., 1965.

253. Энцикловедия «Кругосвет». М.: Медиа-Хаус, 2006.

254. *Яблонский С. В.* Введение в дискретную математику. М., Наука, 1988.

255. *Яблонский С. В.* Функциональные построения в k-значной логике // Труды Матем. ин-та АН СССР им. В. А. Стеклова. 1958. Т. 51. С. 5-142.

256. *Яблонский С. В.* Об алгоритмических трудностях синтеза минимальных контактных схем // Проблемы кибернетики. Вып. 2. М.: Физматгиз, 1959. С. 75-121.

257. *Янов Ю. И.* О логических схемах алгоритмов // Проблемы кибернетики. Вып. 1. 1958.

258. *Янов Ю. И.* Математика, метаматематика и истина // М.: Институт прикладной математики им. М.В. Келдыша, 2006.

259. *Akers S. B.* On a Theory of Boolean Functions // Journal of Society for Industrial and Applied Mathematics. 1959. No. 7, 4.

260. *Akers S. B.* Binary decision diagrams // IEEE Trans. Computers. 1978. Vol. C-27, No. 6. P. 509-516.

261. *Allen J.* Natural Language Understanding. Melno Park, CA: Benjamin/Cummings, 1987.

262. *Ashenhurst R. L.* The Decomposition of Switching Functions // Bell Laboratory Report, 1952, No. BL-1(11). P. 541-642.

263. *Bach E., Jelinek E., Kratzer A., Partee B*. Quantification in Natural Languages. Dordrecht: Kluwer, 1995.

264. *Bernstein B.* Operations with Respect to witch the Elements of Boolean Algebra from a Group // Trans. Amer. Math. Soc. 1924. Vol. 26. P. 171-175.

265. *Boole G.* The Laws of Thought. London: Macmillan, 1854.

266. *Brodie L.* Thinking FORTH. A Language and Philosophy for Solving Problems. – N.J.: Englewood Cliffs, Prentice-Hall, Inc., 1984.

267. *Brodie M. L., Mylopoulos J., Schmidt J. W.* On Conceptual Modeling: Perspectives from Artificial Intelligence, Databases and Programming Languages. New York: Springer-Verlag, 1984.

268. *Bryant R. E.* Graph-based algorithms for Boolean function manipulation // IEEE Trans. on Comp. 1986. V. 35. P. 677-691.

269. *Chomsky N.* Aspects of the Theory of Syntax. Cambridge, MA: MIT Press, 1965.

270. *Chomsky N.* Knowledge of Language: Its Origin. Nature and Use. New York: Praeger Publishers, 1986.

271. *Chrestenson H. E.* A class of generalized Walsh functions // Pacific J. Math. 1955. Vol. 5. P. 17-31.

272. *Cohn M.* Switching Functions Canonical Form over Integer Fields (Ph.D. Thesis). Cambridge: Harvard Univ., 1960.

273. *Collins A., Quillian M. R.* Retrieval time from semantic memory // Journal of Verbal Learning and Verbal Behavior. 1969. No 8. P. 240-247.

274. *Colmerauer A.* Les Grammaires de Metamorphose. Universite Aix-Marseille, 1975.

275. Coloring Large Graph // Proc. of the 1981 S.E. Conf. on Graph Theory, Combinatorics.

276. CCortex: A scalable virtual brain (http://www.ad.com).

277. Conceptual modelling of database applications using an extended ER model / Engels G., Gogolla M., Hohenstein U., Hulsmann K., Lohr-Richter P., Saake G., Ehrich H.-D. // Data & Knowledge Engineering. North-Holland. 1992. No 9(2). P. 157-204.

278. *Cousineau G., Curien P.-L., Mauny M.* The categorical abstract machine // Science of Computer Programming. 1987. Vol. 8, No 2. PP. 173-202.

279. *Curtis H.A.* Non-Disjunctive Decomposition // Bell Laboratory Report, 1958, No. 19, P. 49.

280. *Davies J., Fensel D., Bussler C., Studer R.* The Semantic Web Research And Applications: First European Semantic Web Symposium. Heraklion, 2004.

281. *Dubrova E. V., Muzio J. C.* Generalized Reed-Muller Canonical Form for a Multiple-Valued Algebra // Multiple-Valued Logic. 1996. No. 1. P. 65-84.

282. *Farber D. J., Griswold R. E., Polonsky I. P.* A String Manipulation Language // JACM. 1964. No 11. PP. 21-30.

283. *Floyd R.* Assigning meaning to programs // Mathematical Aspects Computer Science. Amer. Math. Soc. 1967. Vol. XIX. PP. 19-32.

284. *Forgy C. L.* Rete: a fast algorithm for the many pattern/many object pattern match problem // Artificial Intelligence. 1982. No 19. P. 17-37.

285. Formal Methods for Distributed Processing / Ed. Bowman H., Derreck J. Cambridge, Cambridge University Press, 2002.

286. *Francez N.* Verification of programs. Addison-Wesley. 1992.

287. *Frege G.* Begriffsschrift, eine der arithmetischen nachgebildete Formelsprache des reinen Denkens. Hale (Hebert), 1879. (Англ. пер.: Begriffsschrift, a formula language, modelled upon that of arithmetic, for pure thought / Ed. by Van Heijenoort J. 1967. P. 1-82).

288. *Ganter B., Wille R.* Formal Concept Analysis: Mathematical Foundations, Springer, 1999.

289. *Gazdar G., Klein E., Pullum G., Sag I.* Generalized Phrase Structure Grammar, Oxford: Basil Blackwell, 1985.

290. *Gazdar G., Mellish C.* Natural Language Processing in Prolog: An Introduction to Computational Linguistics. Massachusetts: Addison-Wesley, 1989.

291. *Green D.H.* Reed-Muller Expansions of Incompletely Specified Functions // Proc. IEE. 1987. Part E-134. P. 228-236.

292. *Gruber T.* A. Translation Approach to Portable Ontology Specifications // Knowledge Acquisition Journal. 1993. Vol. 5. PP. 199-220.

293. *Haar A.* Zur Theorie der Orthogonalen Funktionensysteme // Math. Ann. 1910. Vol. 69. P. 331-371.

294. *Halliday M. A.* An Introduction to Functional Grammar. London: Edward Arnold, 1985.

295. *Heidorn G. E.* Augmented phrase structure grammar / In Schank and Nash-Webber. 1975.

296. *Herbrand J.* Une methode de demonstration. Thesis, 1931.

297. *Hoare C. A.* An axiomatic basis for computer programming // Communications ACM. 1969. Vol. 12, No 10. PP. 576-583.

298. *Hopfield J. J* Neural networks and physical systems with emergent collective computational abilities // Proc. National Academy of Science. No 79. 1982.

299. *Hendrix G. G.* LIFER: A natural language interface utility // SIGART Newsletter. 1977. Vol. 61. P. 25-26.

300. *Johnson C. D.* Formal Aspects of Phonological Descriptions. Mouton: Hague, 1972.

301. *Kasper R. T., Rounds W. C.* A logical semantics for feature structures. // Proc. 24th Annual Conference of ACL. New York. 1986. P. 235-242.

302. *Kay A.* The Early History of Smalltalk // ACM SIGPLAN Notices. 1993. No 28(3). PP. 69-75.

303. *Knuth D. E.* Semantics of context-free languages // Math. Systems Theory. 1968. Vol. 2, No 2, P. 127-145.

304. *Kohonen T.* Self-Organization and Associative Memory. Berlin, Springer-Verlag, 1984.

305. *Kowalski R. A.* Algorithm = Logic + Control // CACM. 1979. Vol. 22, No 7.

306. *Lambek J.* The mathematics of sentence structure // American Mathematical Monthly. 1958. Vol. 65.

307. *Landin P. J.* The mechanical evaluation of expressions // Computer Journal. 1964. Vol. 6. PP. 308-320.

308. *Lechner R.J.* Harmonic Analysis of Switching Functions. // Recent Developments in Switching Theory. Academic Press, 1971. P. 121-228.

309. *Lee C.Y.* Representation of switching circuits by binary decision programs // Bell Systym Technical Journal. 1959. Vol. 38, No. 4. P. 985.

310. *Lenat D., Miller G., Yokoi T.* CYC, WordNet and EDR: critiques and responses // Communications of the ACM. 1995. Vol. 38 (11). P. 45-48.

311. *Linger R., Mills H., Witt B.* Structured Programming: Theory and Practice. Reading. MA: Addison-Wesley, 1979.

312. *Wegner P.* Vienna definition language // Computer Surveys. 1972. Vol. 4. No 1. PP. 5-63.

313. *Maitra K. K.* Cascaded Switching Networks of Two-Input Flexible Cells // IRE Trans. Electr. Comput. 1962. Vol. TC-11. P. 136-143.

314. *McCluskey E. J.* Logic Design of Multivalued IIL Logic Circuits // IEEE Trans. Comput. 1979. Vol. C28. No. 8. P. 564-569.

315. *Meyer B.* Object Technology: The Conceptual Perspective // Computer. 1996. No 1. P. 86-88.

316. *Morris Ch. W.* Foundations of the theory of signs // International Encyclopedia of Unified Scence I. Chicago, 1938. PP. 17-31.

317. *Moortgat M.* Categorical Type Logics: Handbook of Logic and Language. Elsevier, 1997.

318. *Muller D.E.* Application of Boolean algebra to switching circuit design and to error detection // IRE Trans. Electron. Comput. 1954. V. EC-3. P. 6-12.

319. *Newell A.* The knowledge level // Artificial Intelligence. 1982. No 18. PP. 87-127.

320. *Parr T.* The Definitive ANTLR Reference: Building Domain-Specific Languages. Dallas: Pragmatic Bookshelf, 2007.

321. *Pereira F., Warren D.* Definite Clause Grammars for Language Analysis – a Survey of Formalism and Comparison with Augment Transition Networks // Artificial Intelligence. 1980. Vol. 13. P. 231-278.

322. *Perkowski M. A.* The Generalized Orthonormal Expansion of Function with Multiple-Valued Inputs and Some of its Application // Proc. Int. Symp. of Multi-Valued Logic. 1992. P. 442-450.

323. *Post T. L.* Introduction to a General Theory of Elementary Proposition // Amer. J. Math. 1921. Vol. 43. P. 163-185.

324. *Pradhan D. K.* A Multi-Valued Algebra Based on Finite Fields // Proc. Int. Symp. On MVL. 1974. P. 95-112.

325. *Pratt V. R.* LINGOL: A Progress Report // Proc. 4th IJCAI. 1975. P. 422-428.

326. *Rademacher H.* Einige Sätze von allgemeinen Orthogonalfunktionen // Math. Annalen. 1922. Vol. 87. P. 122-138.

327. *Rader C. M.* Discrete convolution via Mersenne transform // IEEE Trans. Comp. 1972. Vol. C-21.

328. *Reed L. S.* A class of multiple error correction codes and their decoding scheme // IRE Trans. on Inform. Theory. 1954. V. 4. P. 38-42.

329. *Roth J. P.* Minimization over Boolean Trees // IBM Journal. 1960. Vol. 4, 5. P. 543-555.

330. *Roth J.P., Karp R.M.* Minimization Over Boolean Graphs // IBM Journal Res. and Develop. 1962. No. P. 227-238.

331. *Rosenblatt F.* Principles of Neurodynamics. New York: Spartan, 1962.

332. *Rudeanu S.* Boolean Functions and Equations. Amsterdam; London: North-Holland Publ. Co., 1974.

333. *Saraswat V. A.* Concurrent Constraint Programming. Cambridge: MIT Press, 1993.

334. *Savinov A.* Logical Navigation in the Concept-Oriented Data Model // Journal of Conceptual Modeling. 2005. Issue 36.

335. *Schank R. C., Rieger C. J.* Inference and the computer understanding of natural languages. Artifical Intelligence. 1974. Vol. 5, No 4. P. 373-412.

336. *Scott D. S.* Lectures on a mathematical theory of computations. Oxford University Computing, 1981.

337. *Selz O.* Zur Psychologie des Productiven Denkens und des Irrtums // Bonn: Friedrich Cohem, 1922.

338. *Semon W. L.* Characteristic Numbers and Their Use in the Decomposition of Switching Functions // Proc. ACM. 1952. Vol. 17. P. 273-280.

339. *Shannon C.E.* The a Symbolic Analysis of Relay and Switching Circuits // Trans. of American Inst. of Electrical Eengineers. 1938. Vol. 57. P. 713.

340. *Shannon C. E.* The Synthesis of Two-Terminal Switching Circuits // Bell Syst. Techn. J. 1949. Vol. 28. No. 1. P. 59-98.

341. *Simmons R. F., Yu Y.-H.* The acquisition and use of context dependent grammars for English // Computational Linguistics. 1992. Vol. 18, No 4. P. 391-418.

342. *Smith B., Mulligan K.* Framework for Formal Ontology // Topoi. 1983. V.2. P. 73-85.

343. *Sowa J. F.* Conceptual Structures: information processing in mind and machine. Cambridge, MA: Addison Wesley, 1984.

344. *Sowa J. F.* Towards the expressive power of natural language // In Principles of Semantic Networks. Morgan Kaufman, 1991. P. 157–189.

345. *Sowa J. F.* Conceptual Graphs as a universal knowledge representation. // Computers and Mathematics with Applications. 1992. Vol. 23, No 2-5. P. 75-93.

346. Specification Case Studies in RAISE / Ed. Van H.D., George C., Janowski T., Moore R. // In Formal Approaches to Computing and Information Technology. Springer, 2002.

347. *Strazdins I.* The Polynomial Algebra of Multiple-Valued Logic // Algebra, Combinatorics and Logic in Computer Science. 1983. Vol. 42. P. 777-785.

348. *Stoy J. E.* Denotational semantics: the Scott-Strachey approach to programming language theory. MIT Press. 1977.

349. *Su Y.H., Cheung P.T.* Computer Minimization of Multiple-Valued Switching Function // IEEE Trans. Comput. 1972. Vol. C21. P. 995-1003.

350. *Tarski A.* Logic, Semantics, Metamathematics. Oxford, 1956.

351. *Tarski A.* Das Wahrheitsbegriff in den formalisierten Sprachen // Studia Philosophica. 1935. № 1. S. 261-405.

352. *Terry P. D.* Compilers and Compiler Generators – an introduction with C++. International Thomson Computer Press, 1997.

353. *Thompson S.* Type Theory and Functional Programming. Addison-Wesley, 1991.

354. *Tokmen V. H.* Disjoint Decomposability of Multi-Valued Functions by Spectral Means // Proc. IEEE 10th Int. Symp. on Multiple Valued Logic. 1980. P. 83-89.

355. *Tosic Z.* Analytical Representation of an m-Valued Logical Function over the Ring of Integers Modulo m (Ph.D. Thesis). Beograd, 1972.

356. *Vranesic Z. C., Lee E. S., Smith K. S.* A Many-Valued Algebra for Switching Systems // IEEE Trans. Comput. 1970. Vol. C-19. P. 964-971.

357. *Vykhovanets V. S.* The generalized multiplicative forms // Международная конф. по пробл. упр. М., 1999. Т. 3. С. 319-321.

358. *Vykhovanets V. S.* Fundamental Theorems for Polynomial Representation of Discrete Functions // Proceedings of 4th International Conference “Computer-Aided Design of Discrete Devices”. Mink, 2001. Vol. 1. PP. 69-76.

359. *Vykhovanets V. S.* Additive algebra for signal and image processing // Proceedings of SPIE – The International Society for Optical Engineering. 2005. Vol. 5822. PP. 94-97.

360. *Wadler P.* Why no one uses functional languages. ACM SIGPLAN Notices. 1998.

361. *Walliuzzaman K. M., Vranesic Z. G.* On Decomposition of Multiple-Valued Switching Functions // Computer Journal. 1970. Vol. 13. P. 359-362.

362. *Walsh J. L.* A closed set of orthogonal functions // Amer. J. Math. 1923. Vol. 55. P. 5-24.

363. *Webb D. L.* Generation of any N-valued Logic by One Binary Operator // Proc. Nat. Acad. Sci. 1935. Vol. 21. P. 252-254.

364. *Wijngaarden V.A., Mailloux B.J., Peck J.E., Koster C.H*. Report on the algorithmic language Algol-68. Amsterdam: Mathematisch Centrum, 1969.

365. *Wille R., Ganter D.* Formal Concept Analysis. Springer. Berlin: Verlag, 1999.

366. *Wirth N.* Compiler Construction. Addison-Wesley, 1996.

367. *Woods W. A.* Cascaded ATN grammars // American Journal of Comp. Linguistics. 1980. Vol. 6, No 1.

368. *Woods W. A.* What's in a Link: Foundations for Semantic Networks / In Representation and Understanding Studies in Cognitive Science. New York: Academic Press, 1975. P. 35-82.

369. XML Schema. Part 2: Datatypes. W3C Recommendation. http://www.w3.org/.

Приложение 1.
Задача управления лифтом

П1.1. Содержательная постановка задачи

Целью, преследуемой в настоящем приложении, является на примере реальной задачи показать преодоление семантического разрыва между описанием предметной области на естественном языке и ее описанием на создаваемом проблемном языке, а также определение эффективности полученного описания. В качестве примера рассматривается задача управления лифтом[77], ставшая уже традиционной для демонстрации различных подходов к программированию[78].

П1.2. Условие задачи

Дано 9-этажное здание с одним лифтом. Этаж с номером 0 – подвальный, 1 – первый, 2 – второй, и т.д.

На каждом этаже – две кнопки для вызова лифта на движение вверх и вниз. На нулевом этаже кнопка «Вниз» заблокирована, как и кнопка «Вверх» на 9-м этаже.

В кабине лифта имеется панель с кнопками для перемещения на конкретный этаж. В ней также размещены индикаторы движения лифта. Первоначально лифт находится на втором этаже в режиме ожидания.

Разработать программу управления лифтом.

П1.3. Описание работы лифта

Рассмотрим описание работы лифта, выполненное на естественном языке и следующее из содержательных представлений о предметной области.

П1.3.1. Ожидание вызова

Если вызовов нет, то ожидать вызов (П1.3.1). Если вызов со второго этажа, то перейти к открытию дверей (П1.3.2), иначе – на принятие решения о направлении движения (П1.3.4).

П1.3.2. Открытие дверей

Открыть двери. Если дверь открылась, то перейти на принятие решения о направлении движения (П1.3.4), иначе повторить открытие двери (П1.3.2).

---

[77] *Крут Д.* Искусство программирования. В 3-х томах. Т. 1: Основные алгоритмы. М.: Вильямс, 2001.

[78] *Наумов Л. А., Шалыто А. А*. Искусство программирования лифта. Объектно-ориентированное программирование с явным выделением состояний // Информационно-управляющие системы. 2003. №6. С. 38-49

П1.3.3. Закрытие дверей

Закрыть дверь. Если дверь не закрылась, то повторить закрытие двери (П1.3.3), иначе перейти к определению направления движения (П1.3.4).

П1.3.4. Принятие решения

Продолжение движения. Если лифт двигался вверх (вниз) и имеются вызовы на движение вверх (вниз), то начать движение вверх (П1.3.5) (вниз П1.3.6).

Изменение направления. Если вызовов на движение вверх (вниз) нет, но есть вызовы на движение вниз (вверх), то начать движение вниз (П1.3.6) (вверх П1.3.5).

Возврат в начало. Если вызовов нет и при этом лифт выше (ниже) второго этажа, то начать движение вниз (П1.3.6) (вверх П1.3.5), иначе перейти в состояние ожидания (П1.3.1).

П1.3.5. Движение вверх

Начать движение вверх. Если при проходе этажа имеется вызов на движение вверх, то остановить лифт и открыть дверь (П1.3.2), иначе продолжить движение вверх (П1.3.5).

П1.3.6. Движение вниз

Начать движение вниз. Если при проходе этажа имеется вызов для движения вниз, то остановить лифт и открыть дверь (П1.3.2), иначе продолжить движение вниз (П1.3.6).

П1.4. Понятийный анализ

Основными сущностями, используемыми при описании задачи, являются ***Здание***, ***Этаж***, ***Лифт***, ***Дверь***, ***Вызов***, ***Команда*** и ***Движение***.

П1.4.1. Этаж, Вызов, Движение

Здание имеет несколько ***Этажей***. ***Этаж*** характеризуется номером (от 0 до 9). На ***Этаже*** имеются кнопки ***Вызова***, задающие направление ***Движения***. Понятие ***Движение*** будем рассматривать как совокупность сущностей «вверх», «вниз», «нет». В итоге ***Вызов*** может быть определен как агрегат понятий ***Этаж*** и ***Движение***.

П1.4.2. Лифт, Дверь, Команда

Для описания ***Лифта*** используются такие понятия как ***Этаж***, на котором ***Лифт*** находится, состояние ***Движения*** и управление ***Движением***, состояние ***Двери*** (открыта, закрыта) и управление ***Дверью*** (открыть, закрыть). ***Команда*** на ***Движение*** выдается внутри кабины нажатием на кнопку соответствующего ***Этажа***. Вызов ***Лифта*** осуществляется нажатием на кнопки ***Вызова*** на ***Этажах***.

П1.4.3. Управление лифтом

Определим проблемный язык, максимально близкий описанию работы лифта, данному на естественном языке.

Анализ текста П1.3 показывает, что последний структурирован и разделен на пункты (секции). На секции имеются ссылки. Предусмотрим в языке определение таких секций и реализуем механизм перехода из одной секции к другой по ссылке. Для этого введем такие понятия как ***Метка*** и ***Переход***.

В описании работы лифта имеются общеупотребительные языковые конструкции, такие как условное предложение и исчисление высказываний. Для обеспечения наглядности не будем использовать подгружаемые подмодели, описывающие эти области, а определим соответствующие языковые конструкции в создаваемой понятийной модели.

В итоге получаем понятийную структуру, приведенную на рис. П1.1.

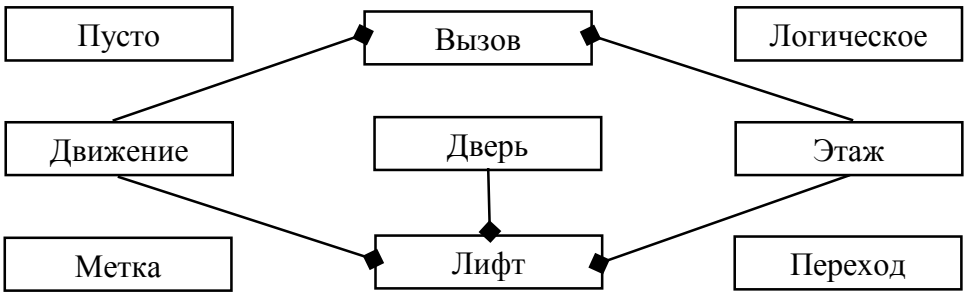

Рис. П1.1. Понятийная структура

П1.5. Понятийная модель для управления лифтом

```
() Движение ()                                  \# Понятие движения:
    ″[Дд]вижение″ {}                            \#     направление движения лифта.
() Дверь ()                                     \# Понятие двери лифта:
    ″[Дд]верь″ {}                               \#     состояние двери.
() Этаж ()                                      \# Понятие этажа:
    'этаж' ″[0-9]″ {}                           \#     выражение этажа по его номеру;
    'этаж' 'вызова' ″ | вверх | вниз″ {} \#
    'этаж' 'лифта' {}                           \#     этаж, где находится лифт
() Вызов (Этаж Движение)                        \# Понятие вызова:
    'вызов' {}                                  \#     состояние вызова лифта.
() Лифт (Этаж Движение Дверь)                   \# Понятие лифта:
    'лифт' {}                                   \#     запуск системы управления
() Метка ()                                     \# Понятие метки секции:
    ″[А-Яа-я][А-Яа-я0-9]*″ {}                   \#     имя (адрес) перехода.
() Переход ()                                   \# Понятие перехода на метку:
    Метка {}                                    \#     перехода по имени (адресу).
```

```
() Логическое ()                          \# Понятие логического высказывания
    Этаж ″выше|ниже|равен″ Этаж {}\#     сравнение этажей по высоте
    Вызов ″вверх|вниз|нет″ {}        \#     проверка направления вызова
    Дверь ″открыта|закрыта″ {}       \#     проверка состояния двери лифта
    'не ' Логическое {}              \#     логическое отрицание
() ()                                \# «Пустое» понятие
    '. ' {}                          \#     точка как знак пунктуации
    Метка ': ' {}                    \#     метка секции описания
    Движение ″вверх|вниз|останов″{}\#     команды управления лифтов
    Дверь ″открыть | закрыть″ {}     \#     команды управления дверью
    'Если ' Логическое ', ' 'то ' Переход {}   \# основные предложения
    'Если ' Логическое ', ' 'то ' Переход ', ' 'иначе ' Переход {}
```

где \# используется для комментирования текста программы.

В приведенной выше модели все предложения только объявлены, т.е. определен их синтаксис. Описание семантики этих предложений может следовать далее по тексту. Будем предполагать, что такое описание выполнено одним из возможных способов:

– на дополнительно определенном в модели проблемном подъязыке;

– последовательностью команд целевой вычислительной платформы;

– на целевом языке программирования.

П1.6. Ситуационное описание системы управления лифтом

Опишем работу лифта на определенном ранее языке. Описание оформим в виде ситуационной части модели, подлежащей исполнению сразу после компиляции.

```
<
    Ожидание:
        Если вызов нет, то Ожидание.
        Если этаж вызова равен этаж 2, то Открыть, иначе Решение.
    Открыть:
        Дверь открыть.
        Если дверь открыта, то Решение, иначе Открыть.
    Закрыть:
        Дверь закрыть.
        Если дверь закрыта, то Решение, иначе Закрыть.
    Решение:
        Если лифт вверх и вызов вверх, то Вверх.
        Если лифт вниз и Вызов вниз, то Вниз.
        Если не вызов вверх и вызов вниз, то Вниз.
        Если не вызов вниз и вызов вверх, то Вверх.
        Если вызов нет и этаж лифта выше этаж 2, то Вниз.
        Если вызов нет и этаж лифта ниже этаж 2, то Вверх, иначе Ожидание.
    Вверх:
```

```
        Движение вверх.
        Если этаж лифта равен этаж вызова вверх, то Открыть, иначе Вверх.
    Вниз:
        Движение вниз.
        Если этаж лифта равен этаж вызова вниз, то Открыть, иначе Вниз.
>
```

Из текста ситуационной части видно, что описание работы лифта имеет легко читаемую форму и не превосходит по числу знаков исходное описание на естественном языке. Последнее позволяет надеяться на высокое качество и надежность порожденного исполняемого кода. Если в рассматриваемой реализации и имеются ошибки, то они вызваны или неточным (противоречивым) исходным описанием, или неверным описанием семантики.

П1.7. Реализация системы управления лифтом

Для реализации системы управления лифтом в виде параллельного процесса мультизадачной вычислительной системы перенесем ситуационное описание из П1.6 в императив одного из предложений понятия Лифт.

```
() Лифт ()                          \# Понятие лифт (доопределение)
    ‘лифт’
        {
            \# Текст ситуационной части из П1.6
        }
```

После компиляции этого предложения получаем исполняемый код, представленный неименованным императивом. Для запуска системы управления лифтом определим еще одно предложение.

```
() ()
    ‘Запустить’ ‘лифт’
        {
            \# Создать вируальную машину в новом процессе и
            \# передать ей для исполнения указатель на императив
            \# предложения ‘лифт’
        }
```

При описании этого предложения используются подгружаемые понятийные модели параллельных процессов и используемой виртуальной машины (в настоящем примере не определены).

П1.8. Анализ эффективности системы управления

В основу определения эффективности положим измерение семантического разрыва между исходным описанием задачи управления лифтом, выполненном на естественном

языке, и формализованным описанием, реализованным средствами контекстной технологии.

Базовой характеристикой исходного описания примем ее длину, равную 1545 знакам. Заметим, что исходный текст не раскрывает семантику терминов и понятий, на основе которого строится описание работы лифта. Поэтому для сравнения выберем текст понятийной модели, также не включающий описании семантики. Суммарное количество знаков в понятийной модели и ситуационном описании равно 1606. Отсюда получаем предварительную эффективность реализации системы управления лифтом, которая равна 1606/1545, что в итоге дает 1,04.

Таким образом, семантический разрыв между текстом исходного описания и текстом программы фактически отсутствует. Небольшое превышение длины текста программы над длиной исходного описания связано с присутствием в программе понятийной структуры и синтаксиса выражения понятий в тексте, в то время как в исходном описании эти сведения не содержатся, а подразумеваются или выявляются в процессе понятийного анализа предметной области.

Учет описания семантики не должен изменить полученную эффективность в худшую сторону. Последнее основано на следующих соображениях. Семантика исходного текста, подразумевающаяся и необходимая для реализации реальной системы управления лифтом, может быть получена только после детального изучения технической документации на лифт и на те подсистемы ввода-вывода вычислительной системы, которые задействованы для управления лифтом и определения его состояния. Эти же сведения должны быть представлены в виде текстов описания семантики соответствующих предложения понятийной модели. Если предположить, что эти частные подзадачи, полученные в результате понятийной декомпозиции предметной области и не превышающие исходную по сложности, будут описаны средствами контекстной технологии с той же эффективностью, что общая задача, то и итоговая эффективность всего решения существенно не изменится.

П1.9. Синтаксическое замыкание понятийной модели

Заметим еще одно немаловажное обстоятельство, проявившиеся в рассматриваемом примере: для интерпретации исходного описания задачи необходимо знание грамматики используемого для этого естественного языка, в то время как в тексте программы содержатся все необходимые сведения для полного грамматического разбора этого текста. Иными словами понятийная модель и ситуационное описании являются синтаксически

замкнуты, в противоположность этому текст исходного описания синтаксически разомкнут.

П1.10. Определение семантики низкоуровневыми средствами

Как в исходном описании работы лифта, так и в понятийной модели не содержится описание семантики. Семантику исходного описания следует выявлять в процессе изучения технической документации на лифт и вычислительную системы, предназначенную для управления лифтом.

Может оказаться, что в результате такого изучения будет установлено, что для реализации той или иной функции системы управления лифтом требуется выполнение дискретной обработки данных, для которой нет адекватных средств в используемой вычислительной системе. А если такая реализация и возможна, то она неэффективна в силу специфичности требований к этой обработке. Более того, применение аппарата понятийного анализа и контекстной технологии для реализации этой обработки не представляется возможным ввиду отсутствия содержательной интерпретации выполняемого преобразования данных. Например, такая ситуация может возникнуть при чтении показания датчиков, когда данные, необходимые для определения состояния лифта, получаются из поступающих данных и задаются на основе табличных форм.

Табличная форма преобразования данных используется в случае, когда отсутствует понятийное описание и содержательная интерпретация выполняемой обработки данных. Для формализации такой обработки следует применять аппарат функциональной декомпозиции, например, как это описано в Главах 5 и 6; а реализацию осуществить или в виде низкоуровневой программы для процессора вычислительной системы, или аппаратурно, в виде отдельного устройства.

Как при программно-аппаратной, так и при аппаратурной реализации табличной обработки данных декомпозицию необходимо выполнять в базисе операций, реализуемых системой команд процессора, или в базисе операций, реализуемых системой логических элементов.

Приложение 2.
Исчисление предикатов

П2.1. Содержательная постановка задачи

В естественном языке синтаксис и семантика находятся в неразрывном единстве, что, собственно, и образует этот язык. В математике и информатике оказалось удобным под формальным языком понимать только его синтаксическую часть, в то время как семантика языка может варьироваться в зависимости от предметной области, круга задач и других условий.

Основным языком современной математики считается язык предикатов, алфавит которого обычно содержит имена предметных переменных и констант, имена предикатов и функций, логических операций, включая кванторы, а также вспомогательные знаки, например, скобки, запятые и т.п. Язык предикатов, в котором допускаются кванторы только по предметным переменным, называется языком первого порядка. Если же кроме этого допускаются кванторы по предикатным и (или) функциональным переменным, то соответствующий язык называется языком второго порядка. Что касается логики предикатов более высоких порядков, то они не нашли значимого применения, так как не являются рекурсивно аксиоматизируемыми, и кроме того, недостаточно изучены[79].

В описании формальной семантики в рамках исчисления предикатов первого порядка одной из известных проблем является проблема разрешимости области применения квантификаторов (quantificational domain, scope)[80], т.е. проблема соответствия между конкретным языковым значением, выраженным с помощью квантификатора и набором референтов реального мира.

Со времени возникновения Пролога, реализующего исчисление предикатов в виде языка программирования высокого уровня, было предложено множество идей по его усовершенствованию. Некоторые из них направлены на то, чтобы сделать его более пригодным для промышленной разработки больших проектов. В этой области Прологу не хватает средств структурирования программ и статической проверки типов. Другая область, в которой Пролог себя хорошо зарекомендовал – исследовательское программирование. Здесь важна максимальная гибкость, возможность быстро написать прототип системы и быстро его изменять, т.к. часто алгоритмы решения задач заранее неизвестны, да и сами задачи постоянно уточняются.

---

[79] *Янов Ю. И.* Математика, метаматематика и истина // М.: Институт прикладной математики им. М.В. Келдыша, 2006.

[80] *Bach E., Jelinek E., Kratzer A., Partee B*. Quantification in Natural Languages. Dordrecht: Kluwer, 1995.

Требования этих подходов к развитию системы логического программирования противоречивы, но поскольку вторая область – исследовательское программирование, является исторически первой, именно ее интересы соблюдаются в первую очередь. До недавнего времени подобным образом обстояло дело и в мире функционального программирования, где доминировал Лисп. Ситуация изменилась с выходом функциональных языков из стен лабораторий в область промышленного программирования. Такие же изменения ожидаются и в логическом программировании. В любом случае, потребность уменьшить разрыв между процедурной и декларативной семантикой Пролога ощущается уже давно.

Покажем, как на основе методологии понятийного анализа может быть решена проблема разрешимости областей применения квантификаторов. Друга цель настоящего приложения – на примере показать эффективность контекстной технологии обработки данных. Последнее выражается в том, что на основе примитивной виртуальной машины, описанной в Приложении 3, и в небольшом объеме исходного текста реализована система логического программирования, не содержащая, разве что, подсистемы объяснения получаемых результатов. И, наконец, третьей целью настоящего приложений является демонстрация решения законченного примера, показывающего выразительные возможности системы контекстного программирования. В этом случае показывается, что контекстная технология является универсальной технологией, т.к. позволяет реализовать исчисление предикатов первого порядка в том виде, как это реализовано в системах логического программирования.

П2.2. Описание исчисления предикатов

Исчисление предикатов – формальный язык классической логики, который использует функторы и предикаты для описания отношений между отдельными сущностями предметной области.

Исчисление предикатов обычно строится на основе исчисления высказываний и включает кванторы, предикатные буквы, функторы, переменные и константы. В отличие от исчисления высказываний, которое не позволяет оперировать обобщенными утверждениями, в исчисление предикатов входят знаки сущностей, их отношения, а также выражения с кванторами. Понятия предметной области моделируются отношениями, рассматриваемыми как наборы взаимосвязанных сущностей.

П2.2.1. Базовые понятия

Константы – элементарные сущности. Не ограничивая общности, в качестве констант будем использовать целые числа.

Область интерпретации – универсальное множество констант. В качестве области интерпретации будем использовать множество целых чисел, возможно, ограниченное некоторой фиксированной разрядностью вычислительного средства.

Переменные – именованные константы. Каждая переменная может принимать произвольные значения из области интерпретации, Однако в каждый момент времени, значением переменной может только одна константа.

Специальные знаки $\neg$, &, $\vee$, $\rightarrow$, $\leftrightarrow$, $\forall$, $\exists$,), (, $\leftarrow$, и , – соответственно знаки отрицания, конъюнкции, дизъюнкции, импликации, эквиваленции, квантора всеобщности, квантора существования, левой скобки, правой скобки, обратной импликации и запятой.

Предикаты – предметные отношения различных местностей, задаваемые именами. Предикат рассматривается как множество упорядоченных наборов констант. Имя предиката совместно с подстановочными значениями его аргументов служит для выражения функции, принимающей значение на специально выделенном множестве констант логической интерпретации.

Логическая интерпретация в самом простом случае осуществляется на множестве из двух констант: «истина», «ложь». Если упорядоченный набор подстановочных значений принадлежит отношению, то предикат считается истинным, в противном случае – ложным.

Функторы – предметные функции различных местностей. Область значений функтора и области определения его аргументов – подмножества области интерпретации, возможно, совпадающие с самой областью.

Имена – обозначения переменных, предикатов и функторов. Имя представляется последовательностью букв и цифр и начинается с буквы.

П2.2.2. Термы и формулы

Термы – это выражения, принимающие предметные (целочисленные) значения. Переменная и константа являются термами по определению. Если τ_1, τ_2, …, τ_n – термы и F – n-местный функтор, то $F(\tau_1, \tau_1, \ldots, \tau_n)$ – терм.

Формулы – это высказывания, которые могут быть или истинными или ложными. Если формула истинна, то ее значение есть *true*, в противном случае – *false*. Если φ_1 и φ_2 – формулы, то $\neg\varphi_1$, $(\varphi_1 \rightarrow \varphi_2)$, $(\varphi_1 \,\&\, \varphi_2)$, $(\varphi_1 \vee \varphi_2)$ также являются формулами. Если x – переменная и φ – формула, то $(\forall x\, \varphi)$ и $(\exists x\, \varphi)$ – формулы. Если τ_1, τ_2, …, τ_n – термы и P – n-местный предикат, то $P(\tau_1, \tau_1, \ldots, \tau_n)$ – формула.

П2.2.3. Язык исчисления предикатов

Тексты на языке классического исчисления предикатов имеют синтаксис, представленный грамматикой на рис. П2.1, где нетерминальные знаки грамматики интерпретируются следующим образом: Calculus – аксиома грамматики, Principle – правило, Proposition – высказывание (формула), Predicate – предикат, Terms – список термов, Term – терм, Functor – функтор, Variable – переменная, Constant – константа, Name – имя, Digit – цифра, Letter – буква.[81]

Из грамматики следует, что логической программой является множество правил Predicate '←' Proposition. Правила в исчислении предикатов могут быть трех видов:

Predicate '←' – ввод фактов (суждений);

Predicate '←' Proposition – определение правил вывода (умозаключений);

'←' Proposition – высказывание, которое надо доказать (цель).

Целевое высказывание может содержать переменные, значения которых не определены (не связаны кванторами). В этом случае требуется найти такие значения переменных, при которых выполняется поставленная цель.

```
Calculus    →  Principle [ Calculus ]
Principle   →  [ Predicate ] '←' [ Proposition ]
Proposition →  Predicate |
               '(' Proposition ')' | '¬' Proposition |
               Proposition '&' Proposition | Proposition '∨' Proposition |
               Proposition '→' Proposition | Proposition '↔' Proposition |
               '∀' Variable Proposition | '∃' Variable Proposition
Predicate   →  Name '(' Terms ')'
Terms       →  Term [ ',' Terms ]
Term        →  Functor | Variable | Constant
Functor     →  Name '(' Terms ')'
Variable    →  Name
Constant    →  Digit [ Constant ]
Name        →  Letter | Name Digit | Name Letter
Digit       →  '0' | '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9'
Letter      →  'a' | 'b' | 'c' | 'd' | 'e' | 'f' | 'g' | 'h' | 'i' | 'j' | 'k' | 'l' | 'm' |
               'n' | 'o' | 'p' | 'q' | 'r' | 's' | 't' | 'u' | 'v' | 'w' | 'x' | 'y' | 'z'
```

Рис. П2.1. Грамматика языка исчисления предикатов

[81] Язык исчисления предикатов, хотя и представлен в исторически сложившейся контекстно-свободной форме, однако контекстно-свободным не является. Об этом свдетельствует, в частности, контексные вхождения переменных в правила вывода грамматики, а также необходимость связывания переменных с их квантификаторами.

П2.2.4. Предметная интерпретация

Предметную семантику языка исчисления предикатов будем представлять совокупностью предметных значений и логических содержаний его предложений. Поскольку речь идет не об анализе уже существующего, а о построении нового формализованного языка, то под ***предметной семантикой*** будем понимать совокупность правил приписывания предметных значений выражениям этого языка. В языках логического программирования предметная семантика, как правило, описывается на естественном языке, т.е. в качестве протоязыка для выражения предметной семантика используется естественный язык.

Под ***предметной интерпретацией*** в исчислении предикатов понимается совокупность областей интерпретации переменных и соответствий, относящих каждый предикат в множество логической интерпретации, а каждый функтор – в некоторой подмножество области интерпретации. В этом случае предметные переменные мыслятся как пробегающие области своей интерпретации, а специальным знакам приписывается смысл логических связок:

– отрицание $\neg$ – логическая связка «***не***»,

– конъюнкция & – логическая связка «***и***»,

– дизъюнкция $\vee$ – логическая связка «***или***»,

– импликация $\rightarrow$ – логическая связка «***если …, то***»,

– эквивалентность $\leftrightarrow$ – логическая связка «***тогда и только тогда, когда***»,

– квантор всеобщности $\forall$ – логическая связка «***для всех … выполняется***»,

– квантор существования $\exists$ – логическая связка «***существует …, что***».

Приписывание предметных значений отдельным выражениям языка осуществляется в зависимости от проблематики решаемых задач. Совокупность всех правил предметной интерпретации исчисления предикатов разобьем на следующие группы:

– правила приписывания предметных значений константам;

– правила определения областей интерпретации предметных переменных;

– правила приписывания предметных значений дескриптивным постоянным (переменным, предикатам и функторам) в составе рассматриваемых формул;

– правила приписывания истинностных значений высказываниям.

П2.2.5. Вычислительная интерпретация

В рассматриваемой постановке задача построения какого-то определенного формального языка не ставится. Создается лишь некоторая схема языков определенного типа, в данном случае – логики предикатов первого порядка. Этот тип языков отличается своей

вычислительной семантикой от других языков, даже языков с тем же синтаксисом, например, языка интуиционистской и релевантной логики. Другие расширения (или некоторые иные вычислительные интерпретации) создаваемого языка исчисления предикатов могут быть получены путем определения новых (дополнительных) прагматик для описываемой в настоящем приложении понятийной модели.

Вычислительную интерпретацию будем использовать для нахождения значений выражений языка исчисления предикатов и задавать семантическим описанием предложений понятийной модели. Для этого нам потребуется всего один аспекте, принятый по умолчанию, т.е. в модели будем определять только одну прагматику.

П2.3. Понятийный анализ

Понятийный анализ описанной выше предметной области выполним с учетом проблематики, направленной на решение проблемы разрешимости областей применения квантификаторов.

П2.3.1. Понятие предиката

В исчислении предикатов имена предикатов могут использоваться в двух контекстах. В первом контексте предикат рассматривается как функция, принимающая логические значения при заданных значениях термов. Такие вхождения предикатов описываются синтаксисом высказываний Proposition (см. рис. П2.1) и подлежат вычислению при заданных подстановочных значениях аргументов, выраженных термами. Во втором контексте предикат интерпретируется как имя отношения и описывается конструкцией Principle.

Из анализа грамматики с учетом активной проблематики также следует существование двух типов предикатов: предикатов, которые перечислимы или разрешимы, и предикатов, которые только разрешимы. Перечислимые предикаты могут использоваться в как в правой, так и в левой части правила Principal, а разрешимые – только в левой.

Таким образом, в первом контексте предикаты будем называть разрешимым (Solvable) и рассматривать как множество упорядоченных наборов констант, на которых предикат принимает истинное значение[82]. Второй контекст используется для определения перечислимого предиката (Enumerable), для чего использовать правило типа Principal, где в зависимости от условия в правой части правила происходит определение предиката как отношения, связывающего значения термов. В этом случае предикат следует переопределять, записывая в его тело те упорядоченные наборы констант, на которых он принимает

[82] Будем использовать предположение о замкнутости мира, т.е. считать, что имеет место полное знание о предметной области. Последнее означает, что предикат может принимать только два значения: «истина» и «ложь». В случае реализации модели разомкнутого мира возможно введение третьего значения предиката – «не определено».

истинное значение. В итоге имеем, что перечислимый предикат является множеством упорядоченных наборов констант, которое в процессе выполнения программы может изменяться.

П2.3.2. Домены

Из анализа понятия предиката следует, что для его описания используются подмножества области интерпретации. Для описания таких подмножеств введем понятие домена. ***Домен*** определим как конечное множество констант.

Домен, как и любое множество, может быть задан двумя способами: перечислением или с помощью разрешающих процедур. В первом случае домен будет перечислимым, во втором случае – разрешимым. Разрешающая процедура позволяет для произвольной константы определить ее принадлежность домену.

П2.3.3. Понятийная структура

Основными сущностями, которые нам потребовались для описания исчисления предикатов, являются:

Константа (Constant) – предметная константа, принимающая целые значения;

Имя (Name) – обозначение сущностей предметной области, выраженное константной последовательностью букв и цифр, первым знаком которой является буква;

Переменная (Variable) – именованная константа, изменяющая свои значения;

Функтор (Functor) – вычислимая функция, принимающей предметные значения;

Терм (Term) – аргумент функтора или предиката, имеющий предметное значение;

Разрешимый (Solvable) – вычислимое отношение, принимающее логические значения, например: истина true, ложь false и т.п.;

Домен (Domen) – перечислимое и изменяемое в процессе вычислений множество констант;

Перечислимый (Enumerable) – перечислимое отношение, принимающее логические значения;

Предикат (Predicate) – обобщенное представление предиката как отношения, принимающего логические значения;

Высказывание (Proposition) – предложение, принимающее логические значения и служащее для выражения высказываний о константах, переменных, функторах и предикатах.

Из содержательных представлений относительно моделируемой предметной области следует, что ***Константа*** и ***Имя*** являются первичными понятиями. ***Переменная*** может быть образована как агрегация ***Константы***, а ***Функтор*** – есть обобщение ***Константы*** и

Переменной. В свою очередь, ***Терм*** определяется как обобщение ***Константы***, ***Переменной*** и ***Функтора***.

Понятийная структура исчисления предикатов показана на рис. П2.2. Для описания предиката необходимо использовать понятия ***Домена*** и ***Разрешимого*** предиката. ***Разрешимый*** предикат рассматривается как предикатное имя, истинность которого может быть вычислена. ***Перечислимый*** предикат образуется при агрегации понятий ***Домена***, ***Константы*** и ***Разрешимого*** предиката. В этом случае ***Домен*** необходим для хранения значений термов, на которых перечислимый предикат принимает логические значения, ***Константа*** используется для представления местности предиката, а ***Разрешимый*** предикат необходим для обобщения ***Перечислимого*** и ***Разрешимого*** предиката в новое понятие ***Предикат***, сущности которого принимают логические значения.

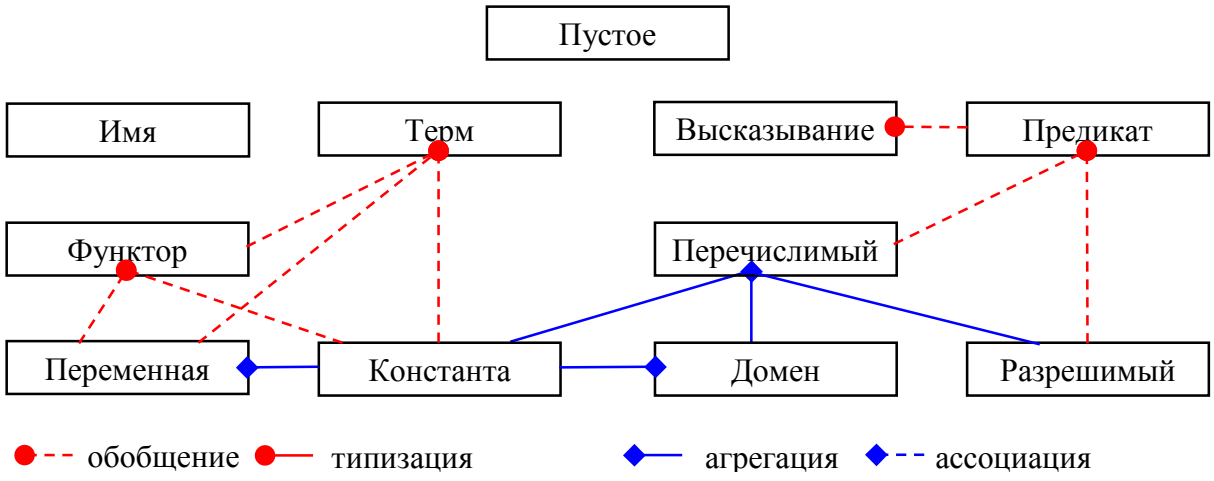

Рис. П2.2. Понятийная структура исчисления предикатов

П2.3.4. Объявление понятий

До определения предложений, описывающих синтаксис и семантику понятий предметной области, зададим ее понятийную структуру. Предварительное целостное описание понятийной структуры не является обязательным, Однако при последовательном описании понятий, могут встретиться предложения, для выражения которых необходимы еще не объявленные понятия.

Ниже приведена понятийная структура исчисления предикатов первого порядка, выраженная на протоязыке системы контекстного программирования.

```
#\-----------------------------------------------------------------------------------------------------------
#\ Объявление понятий и понятийной структры
#\-----------------------------------------------------------------------------------------------------------
        () ()
        () Name ()
        () Constant ()
```

```
() Variable (Constant)
(Constant Variable) Functor ()
(Constant Variable Functor) Term ()
() Solvable ()
() Domain (Constant)
() Enumerable (Domain Constant Solvable)
(Solvable Enumerable) Predicate ()
(Predicate) Proposition ()
#\-----------------------------------------------------------------------------------------------------------
```

Нет никаких препятствий для дополнения этой понятийной структуры новыми понятиями. Последнее может возникнуть в случае, когда потребуется некоторая модификация вычислительной семантики ранее определенных предложений, для выражения которой потребовались дополнительные понятия.

П2.4. Виртуальная машина

Выбор той или иной виртуальной машины определяет вычислительную модель, которая будет использована для реализации задачи. Для исчисления предикатов будем использовать универсальную виртуальную машину, поставляющую необходимые первичные семантические категории. В качестве виртуальной машины выберем машину Kernel, описанную в Приложении 3.

Виртуальная машина Kernel относится к вычислительным устройствам с линейно-ограниченной и стековой вычислительными моделями. Так как объем памяти машины конечен, то ее выразительные способности хуже, чем у машины Тьюринга. Последнее замечание, впрочем, относится ко всем известным вычислительным средствам.

П2.4.1. Структуры данных

Разработаем способы представления сущностей исчисления предикатов. В табл. П2.1 показаны структуры данных, которые получены в результате переноса понятийной структуры исчисления предикатов во внутреннюю форму представления виртуальной машины.

Любая сущность хранится в стеке операндов и представляется содержимым одной ячейки. Если сущность относится к простому понятию, то содержимое этой ячейки однозначно ее определяет. Если сущность интегрирует в себе несколько других понятий, то в стеке операндов она представляется указателем на область локальной памяти, где располагают интегрируемые сущности в порядке их перечисления при объявлении понятия. Если сущность принадлежит понятию, описываемому изменяемым числом ячеек в процессе ее жизненного циклам, то память для такой сущности выделяется в глобальной области.

Из табл. П2.1 видно, что самым сложным по структуре является понятие Enumerable, которое для своего представления требует одной ячейки стековой памяти, области фиксированного размера в локальной памяти и области переменного размера в глобальной памяти.

Заметим, что аналогичные соглашения используются и в объектно-ориентированной технологии. Однако в объектно-ориентированных системах компилятор жестко привязан к вычислительному механизму, задаваемому типом процессора или виртуальной машины, на который ориентирована генерация кода. В контекстной технологии имеется возможность порождать код для одновременно для нескольких виртуальных машин.

Таблица П2.1 Структуры данных для сущностей исчисления предикатов

Определяемое понятие	Стек операндов	Локальная память	Глобальная память
() Name ()	Name		
() Constant ()	Constant		
() Variable (Constant)	Variable →	Constant	
(Constant Variable) Functor ()	Functor {Constant}		
(Constant Variable Functor) Term ()	Term {Constant}		
() Solvable ()	Solvable		
() Domain (Constant)	Domain →	Constant →	Constant … Constant
() Enumerable (Domain Constant Solvable)	Enumerable →	Domain Constant Solvable Constant	Constant … Constant
(Solvable) Predicate ()	Predicate {Solvable}		
(Predicate) Proposition ()	Proposition {Solvable}		

Примечание. Стрелками показаны ссылки на область памяти, а в фигурных скобках – понятие, которому эквивалентно представление в памяти описываемого понятия.

П2.4.2. Генерация кода

Запись команд в область промежуточного кода будем называть ***компиляцией***. Областью ***текущего кода*** назовем элемент предложения, представленный областью промежуточного кода и выраженный текстом в квадратных или фигурных скобках (рис. П2.3).

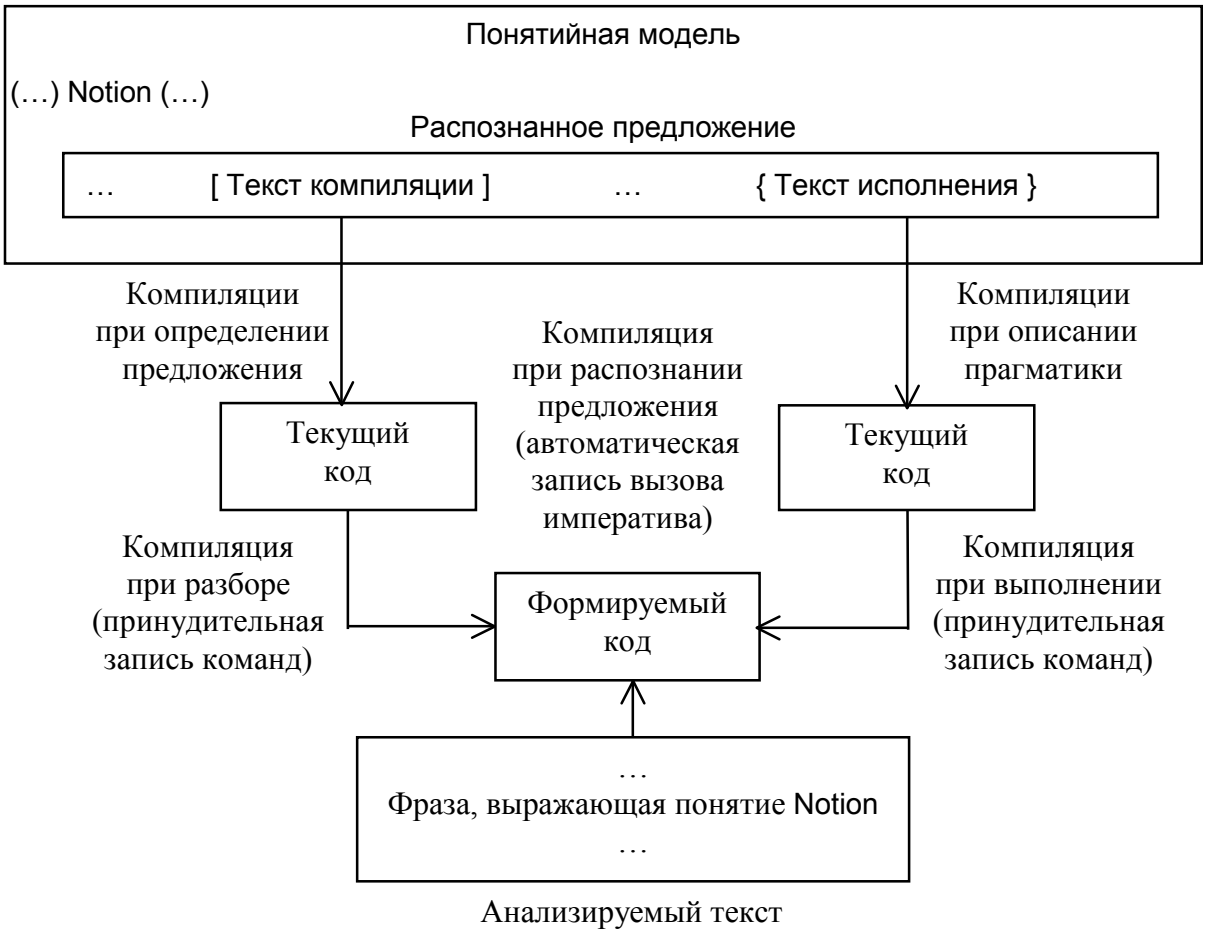

Рис. П2.3. Текущий и формируемый код

Текст в квадратных скобках, или ***текст компиляции***, компилируется во время определения предложения, а исполняется во время грамматического разбора анализируемого текста, где распознано применение этого предложения, выраженное некоторой фразой.

Текст в фигурных скобках, или ***текст исполнения***, также компилируется во время определения предложения, Однако во время грамматического разбора не исполняется. Если во время грамматического разбора распознано применение предложения, то в область формируемого кода вставляется вызов кода, который порожден текстом в фигурных скобках. Исполнение кода, порожденного текстом в фигурных скобках, происходит во время исполнения формируемого кода, при компиляции которого этот вызов был вставлен.

Областью ***формируемого кода*** будем называть тот элемент предложения, который в настоящий момент компилируется. Этот элемент может быть выражен как текстом в квадратных, так и текстом в фигурных скобках. При компиляции фразы исполняется код, представленный текстами компиляции и текстами исполнения того предложения, которое распознано в анализируемом тексте.

Для формирования промежуточного кода определим специальные предложения, позволяющие записывать команды как в область текущего кода (префикс #), так и в область формируемого кода (префикс @).

```
#\-------------------------------------------------------------------------------------------------------------
#\ Средства генерации кода
#\      '#' "[0-2][0-9][0-9]"           Запись байта в область текущего кода
#\      '##' "[0-2][0-9][0-9]"          Запись вызова сервиса в область текущего кода
#\      '@' "[0-2][0-9][0-9]"           Запись байта в область формируемого кода
#\      '@#' "[0-2][0-9][0-9]"          Запись вызова сервиса в область форм. кода
#\-------------------------------------------------------------------------------------------------------------
() ()
        '#' "[0-2][0-9][0-9]"           #\ Запись байта в область текущего кода
        [ ]                             #\ использование аксиомы для записи
        { }
#\-------------------------------------------------------------------------------------------------------------
        '@' "[0-2][0-9][0-9]"           #\ Запись байта в область формируемого кода
        [
                #byte_ #byte_           #\ литерал с кодом литерала байта
                #service_ cbyte_        #\ записать литерал байта в формируемый код
                #i0_ #service_ ivalue_  #\ получить число из шаблона "[0-2][0-9][0-9]"
                #service_ cbyte_        #\ записать число в формируемый код
                #word_                  #\ литерал слово с телом из двух байт
                #service_ cbyte_        #\ два байта команды вызов сервиса
                #service_ cword_        #\ записать слово в формируемый код
        ]
        { }
#\-------------------------------------------------------------------------------------------------------------
        '##' "[0-2][0-9][0-9]"          #\ Запись вызова сервиса в область текущего кода
        [
                @service_               #\ код префикса вызова сервиса
                #i0_ #ivalue_           #\ получить число из шаблона предложения
                #service_ cbyte_        #\ записать байт в область кода
        ]
        { }
#\-------------------------------------------------------------------------------------------------------------
        '@#' "[0-2][0-9][0-9]"          #\ Запись вызова сервиса в область формируемого кода
        [
                @word_                  #\ код литерала слова
                @service_               #\ код первого байта тела литерала
                #i0_ #ivalue_           #\ получить число из шаблона предложения
                #cbyte_                 #\ записать второй байт в область кода
                #word_ #cbyte_          #\ литерал слово (команда вызова сервиса cbyte)
                #cword_                 #\ записать слово в область кода
        ]
```

```
        { }
#\-------------------------------------------------------------------------------------------------------------------
```

Из приведенного выше текста понятийной модели видно, что семантика предложения для записи байта в область текущего кода определяется с помощью аксиомы – предопределенного правила, позволяющего выполнить запись трехзначного десятичного числа в область текущего кода. В свою очередь, семантика записи байта в область формируемого кода определена на основе первого предложения, реализующего запись байт в область текущего кода.

Последние два предложения необходимы для записи в область текущего и формируемого кода вызовов сервисов системы программирования без указания при задании таких вызовов первого байта команды – префикса вызова системного сервиса service_.

П2.4.3. Знаки пунктуации

Знаки пунктуации необходимы для выражения точек последовательности, или останова в последовательности распознавании предложений, выраженных на предметном языке. Для этого, а также для улучшения читаемости текстов будем использовать точку с запятой.

```
#\-------------------------------------------------------------------------------------------------------------------
#\ Пунктуация
#\      '; '                                Точка с запятой – разделитель предложений
#\-------------------------------------------------------------------------------------------------------------------
() ()
        '; '                                #\ Точка с запятой – разделитель
        { }
#\-------------------------------------------------------------------------------------------------------------------
```

П2.4.1. Вспомогательные операторы

Из соображений эффективности семантика некоторых предложений понятийной модели будет определена низкоуровневыми средствами – последовательностью команд виртуальной машины. Однако в понятийной модели будут предложения, реализация которых не столь критична с точки зрения эффективности модели в целом. Семантику таких предложений определим высокоуровневыми средствами – на определяемом предметном языке. Введем в использование вспомогательные предложения, реализующие условный оператор и операторы циклов предметного языка.

```
#\-------------------------------------------------------------------------------------------------------------------
#\ Условный оператор и операторы циклов
#\      'if' Proposition 'then' '' 'else' '' 'end'              Условный оператор
#\      'while' Proposition 'do' '' 'end'                       Цикл с предусловием
#\      'do' '' 'while' Proposition 'end'                       Цикл с постусловием
#\      'for' Variable '=' Term ',' Term ',' Term '' 'end'      Итераторационный цикл
```

```
#\-------------------------------------------------------------------------------------------------------------
() ()
        'if'                            #\ Условный оператор
        Proposition                     #\ Формула, истинность которой проверяется
        [
                @jz_ @001               #\ код[83] переход по нулю на часть «else»
        ]
        'then'
        ' '                             #\ код, выражающий «пустое» понятие
        'else'
        [ #\ Часть «else» распознана
                @jmp_ @002              #\ код перехода на конец оператора
                @label_ @001            #\ код метки части «else»
        ] [[84] #\ Часть «else» не распознана
                @label_ @001            #\ код метки не найденной части «else»
                #i2_ #sitem_            #\ пропустить следующий элемент предложения
        ]
        ' '                             #\ код, выражающий «пустое» понятие
        'end'
        [
                @label_ @002            #\ код метки конца оператора
        ]
        { }
#\-------------------------------------------------------------------------------------------------------------
        'while'                         #\ Цикл с предусловием
        [
                @label_ @001            #\ код метки повторения цикла «while»
        ]
        Proposition                     #\ формула, истинность которой проверяется
        [
                @jz_ @002               #\ код перехода на выход из цикла «while»
        ]
        'do'
        ' '                             #\ код, выражающий «пустое» понятие
        'end'
```

[83] Если в комментарии к тексту программы первым словом является слово «код», то это означает, что в комментируемой строке выполняется запись в область формируемого кода команд, описанных в тексте комментария.

[84] Для реализации условных переходов между элементами предложения при его компиляции разрешено использование второго текста компиляции (в квадратных скобках), который выполняется, если в процессе грамматического разбора предыдущий элемент предложения не был распознан. Последнее необходимо, например, для определения предложений с элементами, которые могут отсутствовать при выражении конкретной сущности некоторого понятия. В тексте выше таким элементом является часть else условного оператора, а сам оператор выражает сущность, принадлежащую пустому понятию.

```
        [
                @jmp_ @001              #\ код перехода на начало цикла «while»
                @label_ @002            #\ код метки конца цикла «while»
        ]
        { }
#\-------------------------------------------------------------------------------------------------------------
        'do'                            #\ Цикл с постусловием
        [
                @label_ @001            #\ код метки начала цикла «do»
        ]
        ' '                             #\ код, выражающий «пустое» понятие
        'while'
        Proposition                     #\ Формула, истинность которой проверяется
        'end'
        [
                @jz_ @001               #\ код перехода на начало цикла «do»
        ]
        { }
#\-------------------------------------------------------------------------------------------------------------
        'for' Variable '=' Term ','     #\ Итераторационный цикл
        [
                @over_                  #\ код извлечения операнда [var min var][85]
                @putd_                  #\ код инициализации переменной [var]
        ]
        Term ','
        [ #\ Если запятая распознана
                #nop_                   #\ ничего не предпринимать
        ] [ #\ Если запятой нет
                @i1_                    #\ код литерала 1 – приращение переменной
                #i2_ #sitem_            #\ пропустить следующий элемент предложения
        ]
        Term
        [
                @rotr_                  #\ код перестановки операндов [add var max]
                @label_ @001            #\ код метки 001 – начало итерации [add var max]
                @over_                  #\ код извлечения операнда [add var max var]
                @getd_                  #\ код чтения переменной [add var max val]
                @cmp_                   #\ код проверки выхода из цикла [add var max val]
                @jl_ @002               #\ код перехода на завершение [add var max val]
                @i4_ @sget_             #\ код получения переменной [add var max val add]
                @add_                   #\ код увеличения на 1 [add var max val+]
                @i3_ @sget_             #\ код получения переменной [add var max val+ var]
```

[85] В комментариях в квадратных скобках показано текущее состояние стека операндов во время выполнения текущего кода, а в фигурных скобках – во время выполнения формируемого кода.

```
        @putd_                  #\ код записи значения переменной [add var max]
    ]
    ' '                         #\ код, выражающий «пустое» понятие
    [
        @jmp_ @001              #\ код перехода на метку 001
    ]
    'end'
    [
        @label_ @002            #\ код метки 002 [add var max val]
        @i4_ @sfree_            #\ код удаления операндов из стека [ ]
    ]
    { }
#\-------------------------------------------------------------------------------------------------------------
```

Объявленные предложения позволяют организовать условную и циклическую интерпретацию текстов, выражающих «пустое» понятие. Формы выражения понятия Proposition, Variable и Term будут определены далее.

П2.4.2. Имена

Одним из общих понятий для многих предметных областей является понятие имени. Имя определим как последовательность букв и цифр, начинающаяся с буквы. Потребуем, чтобы первая буква имени могла быть как прописной, так и строчной, а все другие буквы имени – строчными.

```
#\-------------------------------------------------------------------------------------------------------------
#\ Имя как последовательность букв и цифр, начинающаяся с буквы
#\     "[A-ZA-Я][a-za-я0-9]*"          Имя – строка, начинающаяся с буквы
#\-------------------------------------------------------------------------------------------------------------
() Name ()
    "[A-ZA-Я][a-za-я0-9]*"
    [
        #\ Запись строкового литерала с именем в формируемый код
        @stringb_           #\ код команды строкового литерала
        #i0_ #iterm_        #\ получить имя из шаблона "[A-Za-z][a-z0-9]*"
        #cstring_           #\ записать имя в формируемый код (тело литерала)

        #\ Сохранить имя в атрибуте понятия Name
        #i0_ #iterm_        #\ получить имя из шаблона
        #i1m_ #aput_        #\ записать в атрибут с номером 1
    ]
    { }
#\-------------------------------------------------------------------------------------------------------------
```

После определения, данного выше, понятие Name может быть выражено в тексте, например следующим образом: 'A', 'a0', 'Name', 'Ученый' и т.п. Заметим, что при выраже-

нии имени в тексте никаких проверок на совпадение этого имени с другими именами не делается.

П2.4.3. Таблица идентификаторов

Таблица идентификаторов предназначена для хранения имен сущностей, которые будут использоваться в текстах. Для реализации таблицы идентификаторов будем использовать механизм временных предложений, предоставляемый контекстной технологией.

Определим предложение, которое помещает имя сущности в таблицу идентификаторов, связывает это имя с некоторым понятием и выделяет необходимую этой сущности локальную память.

```
#\-------------------------------------------------------------------------------------------------------------
#\     '#' "%"[86]                    Объявление сущности
#\-------------------------------------------------------------------------------------------------------------
() ()
      '#' "%"                         #\ Объявление сущности [nam] { } → [loc siz] {loc}
      {
            #\ Получить идентификатор понятия по его имени
            #rpush_                   #\ сохранить имя сущности [ ]
            #i1_ #rget_               #\ извлечь имя понятия из шаблона "%" [nam %]
            #inotion_                 #\ получить идентификатор понятия [ntn]

            #\ Определить объем локальной памяти для создаваемой сущности
            #ipush_                   #\ сделать понятие ntn текущим элементом [ ]
            #i0_ #isize_              #\ получить размер сущности [siz]
            #ipop_                    #\ восстановить предыдущий элемент [siz ntn]

            #\ Создать временное предложение для переменной
            #dup_                     #\ копировать идентификатор понятия [siz ntn]
            #xnew_                    #\ создать временное предложение [siz sen]

            #\ Добавить терм имени в созданное предложение
            #dup_                     #\ копирование номера предложения [siz sen sen]
            #rpop_                    #\ восстановить имя сущности [siz sen sen nam]
            #xterm_                   #\ добавить терм [siz sen trm]
            #drop_                    #\ удалить возвращенное значение [siz sen]

            #\ Добавить код компиляции в созданное предложение
            #swap_                    #\ перестановка опернадов [siz sen]
            #i0_                      #\ идентификатор аспекта [siz sen 0]
            #i0_                      #\ иднетификатор нового кода [siz sen 0 0]
```

[86] Символ-сопоставитель % используется для распознания произвольного имени понятия, которое уже определено в понятийной модели. Заметим, что в стандарте языка регулярных выражений такой символ не используется.

```
        #xcode_             #\ добавить код в предложение [siz cod]

        #\ Проверить необходимость выделения локальной памяти
        #over_              #\ получить операнд под вершиной [siz cod siz]
        #jz_ #001           #\ переход на завершение [siz cod] { }

        #\ Зарезервировать локальную память для формируемого кода
        #over_              #\ перестановка операндов [siz cod siz]
        #clocal_            #\ резервирование локальной памяти [siz cod loc]

        #\ Перенаправить генерацию кода в новую область
        #over_              #\ перестановка операндов [siz cod loc cod]
        #ccode_             #\ переключиться в область нового кода [siz cod loc]

        #\ Записать код формирования адреса сущности в новой области
        @dword_             #\ код литерала локального адреса [siz cod loc]
        #dup_               #\ копирование операнда [siz loc loc]
        #cdword             #\ компиляция локального адреса [siz loc]
        @rptr_              #\ код получения глобального адреса

        #\ Восстановить компиляцию в предыдущую область кода
        #i0_                #\ литерал 0 [siz loc 0]
        #ccode_             #\ переключиться на предыдущий код [siz loc]

        #\ Запись кода выделения локальной памяти
        @word_              #\ код литерала размера локальной памяти [siz loc]
        #over_              #\ переставить операнды [siz loc siz]
        #cword              #\ компиляция локального размера [siz loc]
        @rnew_              #\ код выделения локальной памяти [siz loc] {loc}

        #label_ #001        #\ метка завершения 001 [siz cod] { } / [siz loc] {loc}
        #swap_              #\ переставить операнды [cod siz] { } / [loc siz] {loc}
    }
#\-------------------------------------------------------------------------------------------------------------
```

Объяснение особенностей выполнения рассмотренного предложения может быть дано нотацией: [nam] { } → [loc siz] {loc}, которая означает следующее.

Перед использованием определенного выше предложения необходимо на вершину стека записать адрес nam имени сущности, которую необходимо создать. После выхода из императива вместо адреса имени сущности в стек записывается адрес локальной памяти (loc в квадратных скобках), зарезервированной для представления этой сущности. Другим побочным эффектом от использования предложения является запись в области формируемого кода команд выделения реального объема локальной памяти для созданной сущности. При выполнении этого кода на вершине стека появляется адрес этой выделенной

области памяти (loc в фигурных скобках). Здесь nam и loc является сокращения слов name (имя) и local (локальный).

Низкоуровневые средства определения семантики применены в настоящем работе в демонстрационных целях для того, чтобы показать выразительные возможности системы контекстного программирования и сократить объем излагаемых фактов. Однако использование сервисов прямого вызова позволяет уйти от такой степени детализации при объявлении сущностей.

Через сервисы прямого вызова описывается процесс генерации кода в точках жизненного цикла сущности каждого понятия. После такого описания система контекстного программирования автоматически сгенерирует для заданной виртуальной машины в том числе и следующие предложения:

– объявление сущности, принадлежащей понятию;

– преобразование дифференцируемых сущностей в сущность дифференциального понятия;

– преобразование интегрируемой сущности в сущности интегрированных понятий.

Однако для упрощения изложения сервисы прямого вызова далее использовать не будем.

П2.5. Понятийная модель предметной области

П2.5.1. Константы

Константа – первичная сущность, которая используется для выражения предметов в исчислении предикатов. Не ограничивая общности, константы будем представлять целыми числами.

```
#\-------------------------------------------------------------------------------------------------------------
#\ Константа как целое число
#\      "-?[1-9][0-9]*"                 Константа – десятичное целое число
#\-------------------------------------------------------------------------------------------------------------
() Constant ()
      "-?[1-9][0-9]*"
      [
            @long_                    #\ код литерала знакового двойного слова
            #i0_ #ivalue_             #\ получить константу из шаблона "-?[1-9][0-9]*"
            #cdword_                  #\ записать константу в формируемый код

            #\ Сохранить значение константы в атрибуте понятия Constant
            #i0_ #ivalue_             #\ получить константу из шаблона "-?[1-9][0-9]*"
            #i1m_ #aput_              #\ записать в атрибут с номером 1
      ]
```

```
    { }
#\-------------------------------------------------------------------------------------------------------------
```

После определения приведенного выше предложения возможно появление в тексте фраз на предметном языке, выражающих целые числа: '0', '23', '-5' и т.п.

Для предметной интерпретации констант удобно иметь возможность создания именованных констант, т.е. таких констант, которые выражаются именами и имеют неизменное значение.

```
#\-------------------------------------------------------------------------------------------------------------
#\ Именованные константы
#\     'Constant' Name '=' Term      ','     Объявление именованных констант
#\-------------------------------------------------------------------------------------------------------------
() ()
    'Constant' Name             #\ Объявление именованных констант
    [
        #\ Создать сущность с именем Name
        #i1_ #aput_             #\ получить значение атрибута 1 (имя сущности)
        #Constant               #\ создать сущность [cod siz] { }
        #drop_                  #\ удалить операнд из стека [cod] { }
        #i1_ #tput_             #\ сохранить идентификатор кода в ячейке 1 [ ]
    ]
    '=' Term
    [
        #\ Перенаправить генерацию кода в новую область
        #i1_ #tget_             #\ прочитать идентификатор кода из ячейки 1 [cod ]
        #ccode_                 #\ переключиться в область кода константы [ ]

        #\ Запись кода с литералом значения переменной
        @dword_                 #\ код литерала локального адреса [ ]
        #i1_ #aput_             #\ получить значение атрибута 1 (значение) [val]
        #cdword                 #\ компиляция значения переменной [ ]

        #\ Восстановить компиляцию в предыдущую область кода
        #i0_                    #\ литерал 0 [0]
        #ccode_                 #\ переключиться на предыдущий код [ ]
    ]
    ','
    [#\ Если запятая распознана
        #i4m_ #sitem            #\ перейти на элемент Name
    ] [ #\ Если запятая нераспознана
        #nop_                   #\ ничего не делать
    ]
    { }
#\-------------------------------------------------------------------------------------------------------------
```

Таким образом, имеем возможность определять именованные константы, например: 'Constant Иванов = 1, Петров = 3, Сидоров = 0'. После такого определения, если в тексте встретится слово Петров, то это слово будет проинтерпретировано как константа, имеющая целочисленное значение, равное 3.

Нам также потребуется обращение к спискам констант переменой длины, хранящихся в стеке операндов. Для доступа к таким спискам будем использовать следующие предложения.

```
#\-------------------------------------------------------------------------------------------------------------
#\ Доступ к списку констант в стеке операндов
#\      'operand' '(' Term ')'          Константа из стека на заданной глубине
#\      'pop'                           Извлечение из стека константы
#\      'push' Term                     Сохранение константы в стеке
#\      'free' Term                     Удаление из стека заданного числа констант
#\-------------------------------------------------------------------------------------------------------------
() Constant ()
        'operand' '(' Term ')'          #\ Константа из стека на заданной глубине
        [
                @sget_                  #\ код извлечения ячейки из стека
        ]
        { }
#\-------------------------------------------------------------------------------------------------------------
        'pop'                           #\ Извлечение константы из стека
        { }
#\-------------------------------------------------------------------------------------------------------------
() ()
        'push' Term                     #\ Сохранение значения в стеке
        { }
#\-------------------------------------------------------------------------------------------------------------
        'free' Term                     #\ Удаление из стека заданного числа констант
        [
                @sfree_                 #\ код удаления ячейк из стека
        ]
        { }
#\-------------------------------------------------------------------------------------------------------------
```

П2.5.2. Переменные

Переменную опишем как именованную константу. Так как переменная может изменяться, то для хранения переменной выделим ячейку локальной памяти, которую будем использовать для записи и чтения ее текущего значения. Представлять эту ячейку в стеке будем ее глобальным адресом.

```
#\-------------------------------------------------------------------------------------------------------------
#\ Переменная как изменяемая именованная константа
```

```
#\     'Variable' Name '=' Term ','   Объявление и инициализация переменных
#\-------------------------------------------------------------------------------------------------------------
() ()
       'Variable' Name                 #\ Объявление и инициализация переменной
       [
              #\ Создать переменную с именем Name
              #i1_ #aput_              #\ получить значение атрибута 1 (имя сущности)
              #Variable                #\ создать сущность и выделить память [loc] {loc}
              #drop_                   #\ удалить локальный адрес из стека [ ] {loc}
       ]
       '='
       [
              #\ Если инициализатор распознан
              #nop_                    #\ ничего не делать
       ] [
              #\ Если инициализатор не распознан
              @i0_                     #\ код литерала 0 {loc 0}
              #i1_ #sitem_             #\ пропустить следующий элемент предложения
       ]
       Term
       [
              #\ Запись кода инициализации переменной
              @swap_                   #\ код перестановки операндов {0 loc}
              @rput_                   #\ код записи в локальную память { }
       ]
       ','
       [#\ Если запятая распознана
              #i4m_ #sitem             #\ перейти на элемент Name
       ] [ #\ Если запятая нераспознана
              #nop_                    #\ ничего не делать
       ]
       { }
#\-------------------------------------------------------------------------------------------------------------
```

Показанный выше фрагмент понятийной модели служит для определения предложения, необходимого для объявления сущности, принадлежащей понятию Variable, например так: 'Variable x', 'Variable y = 1' или 'Variable x, y = 1'.

Рассмотрим теперь предложения, необходимые для присваивания значений объявленным переменным в процессе вычислений.

```
#\-------------------------------------------------------------------------------------------------------------
#\ Присваивание значений переменной и ее интерпретация как константы
#\     Variable                        Интерпретация переменной как константы
#\     Variable '=' Term               Присваивание переменной нового значения
#\-------------------------------------------------------------------------------------------------------------
() Constant ()
```

```
        Variable                    #\ Интерпретация переменной как константы
        [
                @getd_              #\ код чтения значения переменной
        ]
        { }
#\-------------------------------------------------------------------------------------------------------------
() ()
        Variable '=' Term           #\ Присваивание переменной нового значения
        [
                @swap_              #\ код перестановки операндов {cnst var}
                @putd_              #\ код записи глобальной памяти { }
        ]
        { }
#\-------------------------------------------------------------------------------------------------------------
```

После определения предложений, связанных с переменными, в текстах на предметном языке возможно использование фрагментов:

– выражающих «пустое» понятие ('Variable x', 'Variable y', 'x = 12', 'y = x' и т.п.);

– выражающих понятие Constant ('x', 'y' и т.п.).

При реализации исчисления предикатов нам также потребуется работа со списками переменных, размещенных в стеке операндов. Для этого будем использовать следующее предложение.

```
#\-------------------------------------------------------------------------------------------------------------
#\ Доступ к списку переменным в стеке операндов
#\      'reference' '(' Term ')'        Получение переменной из списка
#\-------------------------------------------------------------------------------------------------------------
() Variable (Constant)
        'reference' '(' Term ')'        #\ Получение переменной из списка по ее номеру
        [
                @sget_              #\ код извлечения ячейки из стека
        ]
        { }
#\-------------------------------------------------------------------------------------------------------------
```

П2.5.3. Функторы

Функтор нами определен как обобщение константы и переменной. Заметим, что наличие в понятийной модели того или иного функтора зависит от предметной области и решаемых в ней задач. Так как предметная область сейчас не задана, то определим только те функторы, которые могут быть общими для всех предметных областей, например реализующие арифметические операции.

```
#\-------------------------------------------------------------------------------------------------------------
#\ Функтор как предметное выражение, принимающее значение константы
#\      '(' Functor ')'                 Арифметические скобки
```

```
#\      '|' Functor '|'                 Модуль
#\      '-' Functor                     Изменение знака
#\      Functor '%' Functor             Остаток от деления
#\      Functor '/' Functor             Целая часть деления
#\      Functor '*' Functor             Умножение
#\      Functor '-' Functor             Вычитание
#\      Functor '+' Functor             Сложение
#\      Functor '?' Functor ':' Functor Арифметическое ветвление
#\-------------------------------------------------------------------------------------------------------------
(Constant Variable) Functor ()
        '(' Functor ')'                 #\ Арифметические скобки
        { }
#\-------------------------------------------------------------------------------------------------------------
        '|' Functor '|'                 #\ Модуль числа
        [
                @tst_                   #\ код установки флагов операнда
                @jns_ @001              #\ код перехода, если положительное
                @neg_                   #\ код изменения знака операнда
                @label @001             #\ код метки 001
        ]
        { }
#\-------------------------------------------------------------------------------------------------------------
        '-' Functor                     #\ Изменение знака
        [
                @neg_                   #\ код команды изменения знака
        ]
        { }
#\-------------------------------------------------------------------------------------------------------------
        Functor '%' Functor             #\ Остаток от деления
        [
                @mod_                   #\ код команды вычисления остатка
        ]
        { }
#\-------------------------------------------------------------------------------------------------------------
        Functor '/' Functor                     #\ Деление
        [
                @div_                   #\ код команды деления
        ]
        { }
#\-------------------------------------------------------------------------------------------------------------
        Functor '*' Functor             #\ Умножение
        [
                @mul_                   #\ код команды умножения
        ]
        { }
```

```
#\-------------------------------------------------------------------------------------------------------------------
        Functor '-' Functor             #\ Вычитание
        [
                @sub_                   #\ код команды вычитания
        ]
        { }
#\-------------------------------------------------------------------------------------------------------------------
        Functor '+' Functor             #\ Сложение
        [
                @add_                   #\ код команды сложения
        ]
        { }
#\-------------------------------------------------------------------------------------------------------------------
        Functor '?' Functor ':' Functor #\ Арифметическое ветвление [prop true false]
        [
                @rotl_                  #\ код циклической перестановки [true false prop]
                @jz_ @001               #\ код перехода по нулю на метку 1
                @swap_                  #\ код перестановки операндов [false true]
                @label_ @001            #\ код метки 1
                @press_                 #\ код уделение операнда под вершиной
        ]
        { }
#\-------------------------------------------------------------------------------------------------------------------
```

Приведем примеры использования предложений, выражающих понятие Functor: '3', '5/-2', '-x', 'y+2', '2*(x-8)'.

П2.5.4. Термы

Терм является обобщением константы, переменной и функтора. Следовательно, терм агрегирует в себе общие признаки этих понятий. Таким признаком, очевидно, является первичное понятие Constant.

```
#\-------------------------------------------------------------------------------------------------------------------
#\ Терм как предметное значение
#\      Constant                Интерпретация константы как терма
#\      Variable                Интерпретация переменной как терма
#\      Functor                 Интерпретация функтора как терма
#\-------------------------------------------------------------------------------------------------------------------
(Constant Variable Functor) Term ()
        Constant                #\ Интерпретация константы как терма
        { }
#\-------------------------------------------------------------------------------------------------------------------
        Variable                #\ Интерпретация переменной как терма
        [
                @getd_          #\ код чтения значения переменной
        ]
```

```
    { }
#\-------------------------------------------------------------------------------------------------------------
    Functor                     #\ Интерпретация функтора как терма
    { }
#\-------------------------------------------------------------------------------------------------------------
```

Если по условию решаемой задачи потребуется определить нестандартные функторы, то это может быть сделано путем дополнения понятийной модели. Например, если потребуется реализовать трехместный функтор с именем Line, вычисляющий линейную функцию от аргументов, то это может быть сделано следующим образом.

```
#\-------------------------------------------------------------------------------------------------------------
(Constant Variable) Functor ()
    'Line' '(' Term `x` ',' Term `a` ',' Term `b` ') '
    {
        a * x + b
    }
#\-------------------------------------------------------------------------------------------------------------
```

После определения функтора Line возможно его использование, например так: 'Line(x-8, 2, -y)'.

П2.5.5. Разрешимые предикаты

Значение любого высказывания будем интерпретировать как истину или ложь. В качестве элементарных вычислимых предикатов определим логические константы false (ложь) и true (истина), которые по своей сути являются нульместными предикатами.

```
#\-------------------------------------------------------------------------------------------------------------
#\ Логические константы
#\    'false'                   Логическая константа «ложь»
#\    'true'                    Логическая константа «истина»
#\-------------------------------------------------------------------------------------------------------------
() Solvable ()
    'false'                     #\ Логическая константа «ложь»
    [
        @tiny_ @000             #\ код литерала 0
    ]
    { }
#\-------------------------------------------------------------------------------------------------------------
    'true'                      #\ Логическая константа «истина»
    [
        @tiny_ @255             #\ код литерала -1
    ]
    { }
#\-------------------------------------------------------------------------------------------------------------
```

Предусмотрим также возможность логической интерпретации констант. Если константа не равна нулю, то припишем ей значение true, в противном случае, ее логическое значение будет false.

```
#\-------------------------------------------------------------------------------------------------------------
#\ Предметная константа как логическое значение
#\      Constant                    Интерпретация константы как предиката
#\-------------------------------------------------------------------------------------------------------------
() Solvable ()
        Constant                    #\ Интерпретация константы как предиката
        [
                @tst_               #\ код проверки значения константы
                @je_ @001           #\ код перехода по равенству константы нулю
                @drop_              #\ код удаления константы из стека
                @tiny_ @255         #\ код литерала –1 (истина)
                @labe_ @001         #\ код метки 001
        ]
        { }
#\-------------------------------------------------------------------------------------------------------------
```

Для любой предметной области стандартными разрешимыми предикатами могут быть отношения порядка и эквивалентности, заданные в области интерпретации (на множестве целых чисел). Определим эти отношения в виде следующих предложений.

```
#\-------------------------------------------------------------------------------------------------------------
#\ Стандартные разрешимые предикаты
#\      Term '=' Term               Отношение «равно»
#\      Term '>' Term               Отношение «больше»
#\      Term '<' Term               Отношение «меньше»
#\-------------------------------------------------------------------------------------------------------------
() Solvable ()
        Term '=' Term               #\ Отношение «равно»
        [
                @sub_               #\ код команды вычитания
                @jz_ @001           #\ код перехода, если равно
                @tiny_ @000         #\ код константы «ложь»
                @jmp_ @002          #\ код безусловного перехода
                @label_ @001        #\ код метки 001
                @tiny_ @255         #\ код константы «истина»
                @label_ @002        #\ код метки 002
        ]
        { }
#\-------------------------------------------------------------------------------------------------------------
        Term '>' Term               #\ Отношение «больше»
        [
                @sub_               #\ код команды вычитания
```

```
            @jg_ @001           #\ код перехода, если больше
            @tiny_ @000         #\ код константы «ложь»
            @jmp_ @002          #\ код безусловного перехода
            @label @001         #\ код метки 001
            @tiny_ @255         #\ код константы «истина»
            @label @002         #\ код метки 002
        ]
        { }
#\-------------------------------------------------------------------------------------------------------------
        Term '<' Term           #\ Отношение «меньше»
        [
            @sub_               #\ код команды вычитания
            @jl_ @001           #\ код перехода, если меньше
            @tiny_ @000         #\ код константы «ложь»
            @jmp_ @002          #\ код безусловного перехода
            @label @001         #\ код метки 001
            @tiny_ @255         #\ код константы «истина»
            @label @002         #\ код метки 002
        ]
        { }
#\-------------------------------------------------------------------------------------------------------------
```

Для определения проблемно-ориентированных разрешимых предикатов описание понятия Solvable следует дополнить предложениями, выражающими необходимые предикаты. Например, если потребуется реализовать трехместный разрешимый предикат с именем Linear, проверяющий линейную зависимость между переменными-термами, то это может быть сделано следующим образом.

```
#\-------------------------------------------------------------------------------------------------------------
() Solvable ()
        'Linear' '(' Term `a` ',' Term `b` ',' Term `c` ') '
        {
            a % b = 0 ?             #\ если a делится на b без остатка
                b % c = 0 ?             #\ и если a делится на b без остатка
                    a / b = b / c ? 1 : 0       #\ то проверить линейную зависимость
                : 0                     #\ иначе 0 (b не делится на c без остатка)
            : 0                     #\ иначе 0 (a не делится на b без остатка)
            = 1                     #\ возврат вычислимого предиката
        }
#\-------------------------------------------------------------------------------------------------------------
```

После определения предиката Linear возможно его использование, например, так: 'Linear(4, 2, x)'. Если значение переменной будет равно 1, то предикат при заданных значениях термов будет выполнимым, в противном случае – невыполнимым.

П2.5.6. Домены

Домен реализуем как именованное множество констант. С каждым доменом свяжем массив ячеек памяти, в которых будем хранить константы, принадлежащие домену. Домен разместим в глобальной памяти, а для ссылок на него будем использовать глобальный адрес его первой ячейки (см. табл. П2.1). Определение количества констант в домене осуществим путем чтения первой ячейки области памяти домена, в которой будем хранить количество записанных констант.

```
#\-------------------------------------------------------------------------------------------------------------
#\ Домен как перечислимое множество констант
#\      Domain '(' ')'              Количество констант в домене
#\      Domain '(' Term ')'         Извлечение константы по номеру (массив констант)
#\-------------------------------------------------------------------------------------------------------------
() Constant ()
        Domain '(' ')'              #\ Количество констант в домене
        [
                @getd_              #\ код чтения адреса глобальной памяти
                @getd_              #\ код чтения адреса глобальной памяти
        ]
        { }
#\-------------------------------------------------------------------------------------------------------------
        Domain                      #\ Извлечение константы по номеру (массив констант)
        [
                #\ Преобразовать адрес домена в адрес глобальной памяти
                @getd_              #\ код получения адреса глобальной памяти
        ]
        '(' Term ')'
        [
                #\ Преобразовать индекс в константу
                @add_               #\ код вычисления адреса константы
                @getd_              #\ код чтения константы из глобальной памяти
        ]
        { }
#\-------------------------------------------------------------------------------------------------------------
```

Константы в домены пронумерованы начиная с 1. Если имеется домен с именем dom, то фраза 'dom()' будет интерпретирована как константа, равная количеству констант в домене dom. В свою очередь, фразы 'dom(1)' и 'dom(2)' будут проинтерпретированы как первая и вторая константы, принадлежащие этому домену.

Для записи в домен констант последний представим как массив переменных с размером, равным числу ячеек глобальной памяти.

```
#\-------------------------------------------------------------------------------------------------------------
#\ Домен как массив переменных
```

```
#\      Domain '[' ']'              Количество переменных в домене
#\      Domain '[' Term ']'         Массив переменных домена
#\-------------------------------------------------------------------------------------------------------------
() Constant ()
        Domain '[' ']'              #\ Количество переменных в домене
        [
                @getd_              #\ код чтения адреса глобальной памяти домена
                @msize_             #\ код получения размера области памяти
        ]
        { }
#\-------------------------------------------------------------------------------------------------------------
() Variable ()
        Domain                      #\ Массив переменных домена
        [
                #\ Преобразовать адрес домена в адрес глобальной памяти
                @getd_              #\ код получения адреса глобальной памяти
        ]
        '[' Term ']'
        [
                #\ Преобразовать индекс в переменную
                @add_               #\ код вычисления адреса переменной
        ]
        { }
#\-------------------------------------------------------------------------------------------------------------
```

В отличие от констант, переменные в домены пронумерованы начиная с 0. Для домена dom фраза 'dom[]' будет выражать число переменных, а фразы 'dom[0]' и 'dom[1]' – будут интерпретироваться соответственно как нулевая и первая переменные домена. Заметим, что нулевая переменная dom[0] ссылается на ячейку, в которой хранится число констант. Это позволяет после добавления или удаления константы в домен изменить их число.

Так как число констант в домене может быть произвольным и меняться в процессе вычислений, предусмотрим предложение, которое позволит изменять размер области памяти домена таким образом, что в каждый момент времени в домене будет свободным от констант как минимум 127 ячеек. Это позволит повысить эффективность реализации путем сокращения интенсивности выделения и освобождения глобальной памяти.

```
#\-------------------------------------------------------------------------------------------------------------
#\ Вспомогательное предложение для работы с глобальной памятью домена
#\      'resize' Domain             Изменение размера массива переменных домена
#\-------------------------------------------------------------------------------------------------------------
() ()
        'resize' Domain `dom`  #\ Изменение размер глобальной памяти домена
        {
```

```
    #\ Объявление константы – размера свободной области домена
        Constant Stock = 128
    #\ Объявление и инициализация временных переменных
        Variable cons = dom(), vars = dom[], ptr = 0
    #\ Вычислить требуемый объем памяти домена
        if
            vars – cons – 1 < Stock
        then
            ptr = vars + Stock
        end
        if
            vars – cons – 1 > 2 * Stock
        then
            ptr = vars – Stock
        end
    #\ Проверка необходимости изменения размера области памяти, если ¬ ptr = 0
        if
            ptr
        then
        #\ Выделить новую область глобальной памяти
            push( ptr )          #\ Затолкнуть в стек новый размер области
            #mnew_               #\ Выделить глобальную область памяти
            ptr = pop            #\ Извлечь из стека указатель на новую область
        #\ Переслать константы в новую область
            #i1_ #rget_          #\ Получить адрес домена dom
            #getd_               #\ Затолкнуть в стек адреса памяти домена
            push( ptr )          #\ Затолкнуть в стек адреса новой области
            push( cons +1 )      #\ Затолкнуть в стек число занятых ячеек
            #moved_              #\ Переслать ячейки в новую область
        #\ Освободить старую область памяти
            #i1_ #rget_          #\ Получить адрес домена dom
            #getd_               #\ Затолкнуть в стек адрес памяти домена
            #mfree_              #\ Освободить область глобальной памяти
        #\ Записать указатель на новую область в домен
            push( ptr )          #\ Затолкнуть в стек новый размер области
            #i1_ #rget_          #\ Затолкнуть в стек адрес домена dom
            #putd_               #\ Записать новый адрес
        end
    }
#\-----------------------------------------------------------------------------------------------------------
```

Для демонстрации возможностей описанной выше понятийной модели определим предложение, позволяющее инициализировать домен списком констант.

```
#\-----------------------------------------------------------------------------------------------------------
#\ Инициализация домена списком контстант
```

```
#\      Domain '=' Term ',' ...          Инициализация домена списком термов
#\-------------------------------------------------------------------------------------------------------------------
        Domain `dom` '='                 #\ Инициализация домена списком констант
        Term `ters` ','
        [
                #\ Если запятая распознана, то вернуться на элемент Term
                @swap_                   #\ код перестановки операндов […, term, domain]
                #i1_ #tget_              #\ увеличить количество констант в ячейке 2
                #inc_                    #\ на единицу
                #i1_ #tput_              #\ сохранить новое значение в той же ячейке
                #i2m_ #item_             #\ повторить распознание элемента Term
        ] [
                #\ Если запятая не распознана, то записать число термов под алиасом ters
                @word_                   #\ код целочисленного литерала
                #i1_ #tget_              #\ количество констант из ячейки 1
                #cword_                  #\ запись в тело литерала (количество термов)
        ]
        {
                #\ Очистка домена от констант
                dom[0] = 0
                #\ Объявление и инициализация переменной цикла
                Variable ter = ters
                #\ Запись констант
                while
                        ter > 0
                do
                        dom[ ter ] = operand( ter )
                        ter = ter –1
                end
                #\ Обновление размера памяти домена
                resize dom
                #\ Удаление списка термов из стека
                free ters
        }
#\-------------------------------------------------------------------------------------------------------------------
```

При необходимости может быть определено понятие разрешимого домена, описывающее другой способ задания множеств – путем определения разрешающей процедуры. В этом случае потребуется реализовать предикат, позволяющий проверить принадлежность некоторой константы разрешимому домену. Однако в настоящей реализации этого не требуется, так как имеется механизм определения разрешимых предикатов, которое, по своей сути, определяют разрешимые множества.

П2.5.7. Перечислимые предикаты

Перечислимые предикаты перед использованием будем объявлять. При объявлении такого предиката помимо имени следует задавать его местность.

```
#\-------------------------------------------------------------------------------------------------------------
#\ Предикат как перечислимое множествонаборов констант
#\      'Predicate' "[1-9][0-9]" Name ','       Объявление перечислимых предикататов
#\-------------------------------------------------------------------------------------------------------------
() ()
        'Predicate' "[1-9][0-9]*"       #\ Объявление перечислимых предикатов
        [
                #\ Сохранить местность предиката во временной памяти
                #i0_ #ivalue_           #\ местность предиката из шаблона (значение)
                #i1_ #tput_             #\ запись во временную ячеку 1
        ]
        Name
        [
                #\ Создать перечислимый предикат с именем nam [ ] { }
                #i1_ #aget_             #\ получить значение атрибута 1 (имя сущности)
                #Enumerable             #\ создать сущность и выделить память [loc] {loc}

                #\ Код выделения глобальной памяти [loc] {loc}
                @i128_                  #\ код литерала 128 (количество ячеек)
                @mnew_                  #\ код выделения глобальной памяти

                #\ Код инициализации домена предиката [loc] {loc ptr}
                @i0_                    #\ код литерала 0 (количество констант)
                @over_                  #\ код извлечения глобального адреса на вершину
                @putd_                  #\ код записи в глобальную память

                #\ Код инициализации локальной памяти [loc] {loc ptr}
                @over_                  #\ код получения локального адреса {loc ptr loc}
                @rput_                  #\ код записи в адреса домена {loc}
                @inc_                   #\ код адреса следующей ячейки {loc+}
                @word_                  #\ код литерала местности предиката {loc ary}
                #i1_ #tget_             #\ чтение местности предиката
                #cword_                 #\ запись в тело литерала
                @swap_                  #\ код перестановки операндов {ary loc}
                @rput_                  #\ код записи в адреса домена { }

                #\ Освобождение глобальной памяти при выходе из зоны видимости [loc]
                #cepilog_               #\ переключиться в эпилог формируемого кода [loc]
                @word_                  #\ код литерала местности предиката [loc] { }
                #cword_                 #\ запись в тело литерала [ ] {loc}
                @rget_                  #\ код чтения указателя глобальной памяти {ptr}
```

```
        @mfree_             #\ код освобождения глобальной памяти { }
    ]
    ','
    [#\ Если запятая распознана
        #i3m_ #sitem        #\ перейти на элемент "[1-9][0-9]*"
    ] [ #\ Если запятая нераспознана
        #nop_               #\ ничего не делать
    ]
    { }
#\-------------------------------------------------------------------------------------------------------------
```

Определенное выше предложение позволяет объявлять перечислимые предикаты. Например, после объявлений 'Predicate 1 P' и 'Predicate 2 Q, 3 R' для каждого предиката создается временное предложение, позволяющее использовать имена P, Q и R как идентификаторы перечислимых предикатов, местность которых равна 1, 2 и 3 соответственно.

После объявления перечислимого предиката предусмотрим возможность получения его местности и домена. Для этого определим предложения, которые будем использовать для доступа к сущностям понятий, агрегированных в понятии Enumerable.

```
#\-------------------------------------------------------------------------------------------------------------
#\ Извлечение сущностей, агрегированных в перечислимый предикат
#\      'Arity' Enumerable          Местность предиката
#\      'Domain' Enumerable         Домен предиката
#\-------------------------------------------------------------------------------------------------------------
() Constant ()
    'Arity' Enumerable              #\ Местность предиката
    [
        @inc                        #\ код увеличения адреса предикта
        @getd_                      #\ код чтения глобальной памяти
    ]
    { }
#\-------------------------------------------------------------------------------------------------------------
() Domain ()
    'Domain' Enumerable             #\ Домен предиката
    [
        @getd_                      #\ код чтения глобальной памяти
    ]
    { }
#\-------------------------------------------------------------------------------------------------------------
```

Приведем примеры использования последних двух предложений. После объявления 'Predicate 2 Q' возможно получение местности предиката Q путем использования фразы 'Arity Q', а домен предиката выражается фразой 'Domain Q'.

П2.5.8. Обобщение перечислимых и разрешимых предикатов

Определение разрешимого предиката Solvable в виде предложения понятийной модели выражает его выполнимость. Однако перечислимый предикат Enumerable задается только именем и не может быть вычислен, так как не заданы его термы. Для вычисления перечислимого предиката при заданных значениях его термов, необходимо создать предложение, выражающее понятие Solvable и позволяющее использовать в тексте, например, следующие фразы:

– 'P(2)' – вычисление одноместного перечислимого предиката P;

– 'Q(x , y-2)' – вычисление двуместного перечислимого предиката Q;

– 'R(x+y, x-y, 5)' – вычисление трехместного перечислимого предиката R.

```
#\-----------------------------------------------------------------------------------------------------------
#\ Перечислимый предикат как предметное значение
#\     Enumerable '(' Term ',' …')'   Выполнимость перечислимого предиката
#\-----------------------------------------------------------------------------------------------------------
(Solvable Enumerable) Predicate ()
    Enumerable `enum` '('       #\ Определение выполнимости предиката
    [
        #i0_  #i1_ #tput_       #\ записать количество термов в ячейку 1
    ]
    Term `ters` ','
    [
        #\ Если запятая распознана, то вернуться на элемент Term
        @swap_                  #\ код перестановки операндов […, term, predicate]
        #i1_ #tget_             #\ увеличить количество констант в ячейке 2
        #inc_                   #\ на единицу
        #i1_ #tput_             #\ сохранить новое значение в той же ячейке
        #i2m_ #item_            #\ повторить распознание элемента Term
    ] [
        #\ Если запятая не распознана, то записать число термов под алиасом ters
        @word_                  #\ код целочисленного литерала
        #i1_ #tget_             #\ количество констант из ячейки 1
        #cword_                 #\ запись в тело литерала (количество термов)
    ]
    ')'
    {
    #\ Объявление временных переменных
        Variable kor, ter
    #\ Вычислить максимальный номер кортежа
        kor = Domain enum () / ters
    #\ Просмотреть все кортежи предиката
        while
            kor > 0
```

```
        do
        #\ Сравнить поэлементно список термов и текущий кортеж предиката
            ter = ters
            while
                ter > 0
            do
                if
                    operand( ter ) = Domain enum ( kor * ters – ter )
                then
                    ter = ter –1
                else
                    ter = -1
                end
            end
            kor = ( ter = -1 ) ? 0 : kor – 1
        end
    #\ Удаление списка термов из стека
        free ters
    #\ Возврат разрешимого предиката
        ( ter = -1 )
    }
#\----------------------------------------------------------------------------------------------------------
```

Заметим, что определение семантики предыдущего предложения на предметном языке порождает код, эффективность которого меньше, чем при использовании команд виртуальной машины. Это связано с тем, что в последнем случае можно уменьшить время вычисления предиката путем реализации прямого сравнения областей памяти низкоуровневыми средствами – с помощью команд виртуальной машины, специально предназначенных для работы со строками.

П2.5.9. Высказывания

Высказывание или формулу будем рассматривать как обобщение разрешимого и перечислимого предикатов, выраженное понятием Proposition. Как и предикат, высказывание может быть истинным или ложным.

Рассмотрим выразительные средства для создания сложных высказываний на основе более простых. Если окажется, что для определения выполнимости сложного высказывания достаточно уже полученного результата вычисления простого высказывания, то другие простые высказывания вычислять не будем.

```
#\----------------------------------------------------------------------------------------------------------
#\ Создание сложных высказываний на основе простых
#\     '(' Proposition ')'              Скобки (порядок интерпретации)
#\     '¬' Proposition                  Отрицание (логическое «не»)
#\     Proposition '&' Proposition      Конъюнкция (логическое «и»)
```

```
#\      Proposition '∨' Proposition          Дизъюнкция (логическое «или»)
#\      Proposition '→' Proposition          Импликация (логический вывод)
#\      Proposition '↔' Proposition          Эквиваленция (логическая равносильность)
#\-------------------------------------------------------------------------------------------------------------
(Predicate) Proposition ()
        '(' Proposition ')'                  #\ Логические скобки (порядок интерпретации)
        { }
#\-------------------------------------------------------------------------------------------------------------
        '¬' Proposition                      #\ Отрицание (логическое «не»)
        [
                @not_                        #\ код отрицания
        ]
        { }
#\-------------------------------------------------------------------------------------------------------------
        Proposition                          #\ Конъюнкция (логическое «и»)
        [
                @tst_                        #\ код проверки выполнимости первого высказывания
                @je_ @001                    #\ код обхода вычисления второго высказывания
        ]
        '&' Proposition
        [
                @and_                        #\ код команды логического умножения
                @label_ @001                 #\ код метки 001
        ]
        { }
#\-------------------------------------------------------------------------------------------------------------
        Proposition                          #\ Дизъюнкция (логическое «или»)
        [
                @tst_                        #\ код проверки выполнимости первого высказывания
                @jne_ @001                   #\ код обхода вычисления второго высказывания
        ]
        '∨' Proposition
        [
                @or_                         #\ код логического сложения
                @label_ @001                 #\ код метки 001
        ]
        { }
#\-------------------------------------------------------------------------------------------------------------
        Proposition                          #\ Импликация (логический вывод)
        [
                @not_                        #\ код команды логического отрицания
                @jne_ @001                   #\ код обхода вычисления второго высказывания
        ]
        '→' Proposition
        [
```

```
        @or_                #\ код логического сложения
        @label_ @001        #\ код метки 001
    ]
    { }
#\-------------------------------------------------------------------------------------------------------------
    Proposition '↔' Proposition   #\ Эквиваленция (логическая равносильность)
    [
        @xor_               #\ код команды исключающего ИЛИ
        @not_               #\ код команды логического отрицания
    ]
    { }
#\-------------------------------------------------------------------------------------------------------------
```

Для сокращения количества скобок в формулах логические связки определены в следующем порядке: ¬, &, ∨, →, ↔, ∃, ∀. Это позволяет использовать приоритеты соответствующих предложений как приоритеты связок, выражаемых этими предложениями. В итоге, наибольший приоритет имеет логическая связка ***не***, самый низкий приоритет – квантор всеобщности ∀.

П2.5.10. Простые и вычислимые факты

Приступим к определению предложения, предназначенного для добавления фактов относительной предметной области. Для накопления фактов будем использовать перечислимый предикат, рассматриваемый как отношение, заданное на множестве констант. Так как используется предположение о замкнутом мире, то выражение вида 'Q(a, b) ← true' эквивалентно добавлению кортежа[87] констант (a, b) в домен предиката, а 'Q(a, b) ← false' – удалению этого кортежа.

```
#\-------------------------------------------------------------------------------------------------------------
#\ Добавление (удаление) элементов перечислимого предиката
#\     Enumerable '(' Term ',' … ')' '←' Proposition     Добавить факт
#\-------------------------------------------------------------------------------------------------------------
() ()
    Enumerable `enum` '('       #\ Добавить факт
    [
        #i0_  #i1_ #tput_       #\ записать количество термов в ячейку 1
    ]
    Term `ters` ','
    [
        #\ Если запятая распознана, то вернуться на элемент Term
        @swap_                  #\ код перестановки операндов […, term, predicate]
        #i1_ #tget_             #\ увеличить количество констант в ячейке 2
        #inc_                   #\ на единицу
```

[87] Под кортежем здесь понимается упорядоченный набор констант.

```
        #i1_ #tput_             #\ сохранить новое значение в той же ячейке
        #i2m_ #item_            #\ повторить распознание элемента Term
] [
        #\ Если запятая не распознана, то записать число термов под алиасом ters
        @word_                  #\ код целочисленного литерала
        #i1_ #tget_             #\ количество констант из ячейки 1
        #cword_                 #\ запись в тело литерала (количество термов)
]
')' '←' Proposition `prop`
[
        #\ Если высказывание распознано, завершить разбор предложения
        #nop_                   #\ нет никаких действий
] [
        #\ Если высказывание не распознано, то подставить true
        @i1m_                   #\ коди литерала логической единицы (-1)
]
{
#\ Объявление временных переменных
        Variable ter, kor, ind
#\ Вычислить количество кортежей в домене предиката
        kor = Domain enum () / ters
#\ Поиск кортежа из заданных термов в домене предиката
        while
                kor > 0
        do
                ter = ters
                while
                        ter > 0
                do
                        ind = kor
                        if
                                operand( ter ) = Domain enum ( ind * ters – ter )
                        then
                                ter = ter –1
                        else
                                ter = -1
                        end
                end
                kor = ( ter = -1) ? kor – 1 : -1
        end
#\ Проверка необходимости изменения предиката
        if
                ¬ ( ter = -1 ↔ prop )
        then
                #\ Проверка на добавления или удаление
```

```
                if
                        prop
                then
                #\ Добавление нового кортежа
                        ind = Domain enum () + ters
                        for ter = 1, ters
                                Domain enum [ ind - ter ] = operand( ter )
                        end
                #\ Записать новое число констант
                        Domain enum [ 0 ] = Domain enum () + ters
                else
                #\ Удаление существующего кортежа (ind – удаляемый кортеж)
                        ind = ( ind – 1 ) * ters
                        for ter = 1, Domain enum () – ind
                                Domain enum [ ind ] = Domain enum [ ind + 1]
                        end
                #\ Записать новое число констант
                        Domain enum [ 0 ] = Domain enum () – ters
                end
        end
    #\ Удаление списка термов из стека
        free ters
    }
#\-------------------------------------------------------------------------------------------------------------------
```

Приведем примеры добавления фактов. Для рассмотренного выше предиката Q возможно употребление следующих фраз в тексте программы:

– 'P(1) ← true' или 'P(1) ← ' – ввод простого факта;

– 'Q(2, -1) ← false' – удаление простого факта;

– 'R(0, x-1, 5*(x+y)) ← ¬Q(-x, 2) & y > x' – ввод или удаление вычислимого факта (предполагается, что переменным x и y присвоены некоторые предметные значения).

При использовании разомкнутой модели мира возможна другая реализация механизма накопления фактов. В этом случае с каждым кортежем следует хранить значение истинности: если факт выполним, то значение истинности кортежа будет равно true, а если факт невыполним, то false. Тогда отсутствие кортежа (a, b) в домене перечислимого предиката Q будет означать, что факт не определен, и суждение вида 'Q(a, b)' следует рассматривать как принимающее третье логическое значение, например, undefined.

П2.5.11. Высказывания с кванторами

Наиболее сложной частью исчисления предикатов является реализация высказываний с кванторами. Определим предложения, которые позволят выразить высказывания следующего вида:

– '∃ x ∈ P x > 0';

– '∃ x, y ∈ Q ¬ x = 0 & P(y)';

– '∀ x, y, z ∈ R x < 0 ∨ P(y) → ¬Q(1, z)';

– '∀ x ∈ P ∀ y, z ∈ Q x / 2 = 0 & P(y) ↔ R(x, y, z)'.

```
#\-----------------------------------------------------------------------------------------------------------
#\ Высказывания с кванторами
#\     "(∃)|(∀)" Variable ',' …'∈' Enumerable Proposition     Высказывание с кванторами
#\-----------------------------------------------------------------------------------------------------------
      "(∃)|(∀)"                     #\ Высказывание с кванторами
      [
            @byte_                  #\ код байтового литерала
            #i0_ #imatch_           #\ получить номер распознанного квантора
            #cbyte_                 #\ записать в тело литерала номера квантора
            #i0_ #i1_ #tput_        #\ записать количество переменных 0 в ячейку 1
      ]
      Variable `vars` ','
      [
            #\ Если запятая распознана, то вернуться на элемент Variable
            #i1_ #tget_             #\ увеличить количество констант в ячейке 2
            #inc_                   #\ на единицу
            #i1_ #tput_             #\ сохранить новое значение в той же ячейке
            #i2m_ #item_            #\ повторить распознание элемента Variable
      ] [
            #\ Если не распознана, то записать число переменных под алиасом vars
            @stringd_               #\ код строкового литерала
            #i1_ #tget_             #\ количество переменных из ячейки 1
            #cdword_                #\ запись двойного слова в тело литерала
            #i0_ #cdword_           #\ запись конца строки в тело литерала
      ]
      '∈' Enumerable `enum`
      [
            #\ Перенаправить компиляцию в новую область кода
            #cnew_                  #\ создать новую область кода
            #dup_                   #\ скопировать идентификатор нового кода
            #i2_ #tput_             #\ сохранить идентификатор кода в ячейке 2
            #ccode_                 #\ переключится на новую область кода
      ]
      Proposition `prop`
      [
            #\ Восстановить компиляцию в текущую область кода
            #i0_ #ccode_            #\ восстановить старую область кода

            #\ Записать идентификатор новой области кода под алиасом prop
            @dword_                 #\ код целочисленного литерала
```

```
        #i2_ #tget_             #\ идентификатор кода для высказывания
        #cdword_                #\ запись в тело литерала
    ]
    {
    #\ Объявление временных переменных
        Variable var, kor, kors
    #\ Количество кортежей в предикате kors
        kors = Domain enum () / vars
    #\ Цикл по всем кортежам
        kor = kors;
        while
            kor > 0
        do
        #\ Уменьшить номер кортежа
            kor = kor – 1
        #\ Присвоить значения переменным
            for var = 1, vars
                reference( var ) = Domain enum ( kor*vars + var )
            end
        #\ Вычислить предикат
            push( prop )
            #exec_
            var = pop
        #\ Проверить результат вычисления предиката
            if
                var ↔ operand( vars+1 ) = 1
            then
                kor = -1
            end
        end
    #\ Удаление списка переменных из стека
        free vars
    #\ Возврат разрешимого предиката
        ( kor = -1 )
    }
#\-----------------------------------------------------------------------------------------------------------
```

Заметим, что как и при определении логических связок, при вычислении выражений с предикатами ищется первый случай, опровергающий или подтверждающий высказывание. Так, при вычислении высказывания с квантором существования происходит завершение итераций, если найдена комбинация переменных, на которой высказывание выполнимо. В свою очередь, при вычислении выказывания с квантором всеобщности первая комбинация переменных, на которой предикат невыполним, приводит к прекращению итерации.

П2.6. Демонстрационный пример

П2.6.1. Описание задачи

Определим выполнимость формулы $\forall x \in Q \;\; P(x,a) \& P(x,b) \to P(x,y)$ при условии, что областью интерпретации переменных x и y являются одноместные предикаты Q и R, константам a, b приписаны некоторые значения, а предикат $P(x, y)$ является отношением «x делится без остатка на y».

Рассматриваемая формула интерпретируется как высказывание «Для всякого $x \in Q$ верно, что если x делится на a и на b, то x делится на y». Ясно, что это высказывание и соответствующая ему формула могут быть как истинными, так ложными.

Например, если предикат Q содержит целые положительные числа от 1 до 15, $a = 2$, $b = 3$ и $y = 6$, то формула будет выполнимой. Однако при $y = 5$ формула становится невыполнимой. В свою очередь при $R = \{1, 2, 3, 6\}$ формула является тождественно истинной или общезначимой.

П2.6.2. Дополнение понятийной модели

Поставим целью так дополнить имеющуюся понятийную модель исчисления предикатов, чтобы можно было вычислять истинность формул, построенных на основе разрешимого предиката $P(x, y)$. Для этого определим следующее предложение, которое позволяет выражать $P(x, y)$ в тексте, и опишем семантику этого предложения.

```
#\-------------------------------------------------------------------------------------------------------------
() Solvable ()
        'P' '(' Term `a` ',' Term `b` ')'          #\ Отношение «делится без остатка»
        {
                a % b = 0
        }
#\-------------------------------------------------------------------------------------------------------------
```

П2.6.3. Ситуационное описание

После определения предиката возможно приведенное ниже ситуационное описание, в котором перечисляются известные факты относительно рассматриваемой предметной области и приводится решение некоторой задачи. В частности, решается задача определения выполнимости и общезначимости формул на различных множествах интерпретации.

```
#\-------------------------------------------------------------------------------------------------------------
<       #\ Ситуационное описание задачи

        #\ Объявление переменных
```

```
Variable x, y

#\ Объявление перечислимых одноместных предикатов Q и R
Predicate 1 Q, 1 R

#\ Ввод фактов (прямое перечисление констант)
Domain Q = 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15

#\ Ввод фактов (пропозиционная форма)
R(0) ← false   R(1) ← true   R(2) ←;   R(3) ←;   R(6) ←;

#\ Ввод фактов (в процессе вычислений)
Variable i = 6
do
        R(i) ← 6 % i = 0
        i = i – 1
while
        i > 0
end

#\ Проверка выполнимости формулы при y = 6
y = 6
if
        ∀ x ∈ Q  P(x, 2) & P(x, 3) → P(x, y)
then
        message 'при y = 6 формула выполнима'[88]
else
        message 'при y = 6 формула не выполнима'
end

#\ Проверка выполнимости формулы при y = 5
y = 5
if
        ∀ x ∈ Q  P(x, 2) & P(x, 3) → P(x, y)
then
        message 'при y = 5 формула выполнима'
else
        message 'при y = 5 формула не выполнима'
end

#\ Проверка общезначимости формулы при y ∈ R
```

[88] В тексте ситуационной части используется не определенное в модели понятие String (строка), при описании которого определены предложения для вывода сообщений () () 'message' String, задания строковых констант () String () " '.* ' ", конкатенации строк () String () String '&' String и форматирования констант () String () Constant.

```
        if
                ∀ y ∈ R  (∀ x ∈ Q  P(x, 2) & P(x, 3) → P(x, y))
        then
                message 'формула общезначима при y ∈ R'
        else
                message 'формула не общезначима при y ∈ R'
        end

        #\ Проверка общезначимости формулы при y ∈ Q
        if
                ∀ y ∈ Q  (∀ x ∈ Q  P(x, 2) & P(x, 3) → P(x, y))
        then
                message 'формула общезначима при y ∈ Q'
        else
                message 'формула не общезначима при y ∈ Q'
        end

        #\ Проверка выполнимости формулы при y ∈ P
        if
                ∃ y ∈ Q  (∀ x ∈ Q  P(x, 2) & P(x, 3) → P(x, y))
        then
                message 'формула выполнима при y = ' & y
        else
                message 'формула не выполнима'
        end
>
#\-----------------------------------------------------------------------------------------------------------
```

П2.6.4. Результат вычислений

В результате интерпретации ситуационной части в соответствии с заданной ранее понятийной моделью получим следующие сообщения:

– «при y = 6 формула выполнима»;

– «при y = 5 формула не выполнима»;

– «формула общезначима y ∈ R»;

– «формула не общезначима при y ∈ Q»

– «формула выполнима при y = 1».

П2.7. Вывод в логике предикатов

До сих пор нами были определены предложения выражающие:

– высказывания, принимающие логические значения;

– конструкции, применяемые для ввода и накопления фактов.

Однако в логике предикатов используются еще два типа выражений: правила вывода и целевые высказывания.

П2.7.1. Правила вывода

Правилами вывода в исчислении предикатов (см. рис. П2.1) называются высказывания вида Predicate '←' Proposition, где предикат определен не на списке термов, принимающих предметные значения, а на переменных, значения которых не известны или не заданы. Такие переменные называются ***свободными***.

Например, правилом вывода может быть фраза 'P(x) ← ∀ y ∈ Q x>y → R(x, 1, y)'. В рассматриваемом контексте свободной переменной будет переменная x. В свою очередь значения переменной y задаются квантором. Такая переменная называется ***связанной***, в данном случае – связанной квантором всеобщности.

Заметим, что нами разработана понятийная модель, которая приведенную выше фразу будет интерпретировать как ввод фактов, если предикат P объявлен как перечислимый. Если предикат P не объявлен или определен как разрешимый, то текст фразы распознан не будет, что вызовет сообщение об ошибке.

По своей сути правила вывода являются разрешимыми предикатами и задают высказывания, которые после подстановки значений свободных переменных могут быть вычислены. Такие предикаты мы назвали вычислимыми. Более того, для объявления и определения вычислимых предикатов нами использованы выразительные возможности понятийной модели, а именно, определение синтаксиса и семантики соответствующих предложений для выражения разрешимых предикатов в тексте.

П2.7.2. Определение правил вывода

Понятийная модель может быть дополнена произвольными правилами вывода. Однако для совместимости с распространенной интерпретацией текстов на языках логического программирования разработаем механизм, позволяющий задавать правила вывода в традиционной форме, как это принято в исчислении предикатов.

```
#\-------------------------------------------------------------------------------------------------------------
#\ Определение правил вывода
#\      'Δ' Name '(' Name ',' …')' '←' Proposition          Задать правило вывода
#\-------------------------------------------------------------------------------------------------------------
() ()
      'Δ' Name `enum` '('              #\ Задать правило вывода
      [
            #\ Инициализация временной памяти
            #i0_ #i1_ #tput_          #\ записать количество аргументов 0 в ячейку 1

            #\ Сформировать временное предложение для предиката
```

```
        #byte_ #007             #\ идентификатор понятия Solvable
        #xnew_                  #\ создать временное предложение
        #i2_ #tput_             #\ записать идентификатор предложения в ячейку 2

        #\ Добавить терм имени предиката в созданное предложение
        #i2_ #tget_             #\ получить идентификатор предложения [sen]
        #i1_ #aget_             #\ извлечь имя сущности [sen enum]
        #xterm_                 #\ добавить терм [trm]
        #drop_                  #\ удалить возвращенное значение [ ]

        #\ Добавить терм левой скобки в созданное предложение
        #i2_ #tget_             #\ получить идентификатор предложения [sen]
        #i0_ #iterm_            #\ получить терм скобки [sen str]
        #xterm_                 #\ добавить терм [trm]
        #drop_                  #\ удалить возвращенное значение [ ]
]
Name `nams`
[
        #\ Добавить понятие Term в созданное предложение
        #i2_ #tget_             #\ получить идентификатор предложения [sen]
        #byte_ #006             #\ идентификатор понятия Term [sen ntn]
        #i1_ #aget_             #\ извлечь имя аргумента как алиаса [sen ntn als]
        #xnotion_               #\ добавить понятие [ntn]
        #drop_                  #\ удалить возвращенное значение [ ]
]
','
[
        #\ Если запятая распознана, то вернуться на элемент Name
        @swap_                  #\ код перестановки операндов […, term, predicate]
        #i1_ #tget_             #\ получить количество констант в ячейке 1
        #inc_                   #\ увеличить на единицу
        #i1_ #tput_             #\ сохранить новое значение в той же ячейке

        #\ Добавить терм запятой в созданное предложение
        #i2_ #tget_             #\ получить идентификатор предложения [sen]
        #i0_ #iterm_            #\ получить терм скобки [sen str]
        #xterm_                 #\ добавить терм [trm]
        #drop_                  #\ удалить возвращенное значение [ ]
        #i2m_ #item_            #\ повторить распознание элемента Name
] [
        #\ Если запятая не распознана, то записать число аргументов как nams
        @word_                  #\ код целочисленного литерала
        #i1_ #tget_             #\ количество аргументов из ячейки 1
        #cword_                 #\ запись в тело литерала (количество аргументов)
]
```

```
')'
[
        #\ Добавить терм правой скобки в созданное предложение
        #i2_ #tget_             #\ получить идентификатор предложения [sen]
        #i0_ #iterm_            #\ получить терм скобки [sen str]
        #xterm_                 #\ добавить терм [trm]
        #drop_                  #\ удалить возвращенное значение [ ]
]
'←'
[
        #\ Перенаправить компиляцию в новую область кода
        #cnew_                  #\ создать новую область кода
        #dup_                   #\ скопировать идентификатор нового кода
        #i2_ #tput_             #\ сохранить идентификатор кода в ячейке 2
        #ccode_                 #\ переключится на новую область кода
]
Proposition `prop`
[
        #\ Восстановить компиляцию в текущую область кода
        #i0_ #ccode_            #\ восстановить предыдущую область кода

        #\ Добавить императив в созданное предложение
        #i2_ #tget_             #\ получить идентификатор предложения [sen]
        #i0_                    #\ идеантификатор аспекта по умолчанию [sen 0]
        #i2_ #tput_             #\ сохранить идентификатор кода [sen 0 cod]
        #ximper_                #\ добавить императив [trm]
        #drop_                  #\ удалить возвращенное значение [ ]

        #\ Записать идентификатор новой области кода под алиасом prop
        @dword_                 #\ код целочисленного литерала
        #i2_ #tget_             #\ идентификатор кода для высказывания
        #cdword_                #\ запись в тело литерала
]
{
        #\ Очистить стек от списка аргументов
        free nams
}
#\-------------------------------------------------------------------------------------------------------------
```

Определенное выше предложение избавляет пользователя от дополнения понятийной модели предложений для выражения разрешимых предикатов средствами контекстной технологии. Вместо этого может быть использована традиционная для логического программирования форма задания правил вывода. Надо заметить, что использование этого предложения возможно как при определении семантики предложений, так и в ситуационной части программы.

Если в тексте программы встретятся две фразы 'P(x) ← ∀ y ∈ Q x>y → R(x, 1, y)' и 'Δ P(x) ← ∀ y ∈ Q x>y → R(x, 1, y)', то первая фраза интерпретируется как ввод факта, а вторая – задает правило вывода. В первом случае переменная x предполагается связанной, во втором случае – свободной. Для отличия ввода фактов от определения правил вывода служит квантор определения (дефиниции) Δ.

Рассмотрим демонстрационный пример. Пусть требуется определить правило вывода в виде разрешимого трехместного предиката Q, интерпретируемого как «быть связанным линейным уравнением». Вычислительная интерпретация предиката Q выражается так: a*x + y = 0. В ситуационной части программы запишем правило вывода в следующей форме: Δ Q(x, y, a) ← a*x + y = 0. После такого определения в понятийной модели появится предложение, позволяющее выражать этот предикат далее по тексту, например так: Q(1, 2, 3). Более того, при исполнении ситуационной части в соответствующей формуле выполнимость предиката Q будет определена путем вычисления выражения 3*1 + 2 = 0.

П2.8. Целевые высказывания

Последним типом выражений, применяемых в исчислении предикатов первого порядка, является предложением вида '←' Proposition. Это предложение называется целевым и определяет высказывание, которое надо доказать. Как и в случае предложений с правилами вывода, высказывание Proposition может содержать или не содержать свободные переменные.

П2.8.1. Цель со связанными переменными

Если целевое предложение не содержит свободных переменных, то задача его вычисления сводится к простой проверке выполнимости предложения. В этом случае высказывание может быть представлено как разрешимый нульместный предикат (логическое значение).

Приведем пример разрешимого целевого предложения. Пусть необходимо решить задачу $\leftarrow Q(x, y, a)$, где переменная x свободна, а переменная y связанная, т.е. имеет некоторое значение. Областью интерпретации переменной x является множество целых чисел Z, $x \in Z$. Если предикат $Q(x, y, a)$ является отношением «быть решением уравнения $ax + y = 0$», то полный перебор возможных значений переменной не требуется: зная константу a и значение переменной y, можно легко установить выполнимость предиката Q и определить множество, на котором этот предикат выполним.

Сложнее дело обстоит при равенстве нулю константы a и переменной y. В этом случае предикат также выполним, но его выполнимость происходит не при единственном значении переменной x, а на всех возможных ее значениях.

Решением указанной проблемы в любой системе логического программирования будет исключение такого предложения из списка целей, создание нового вычислимого предиката, эквивалентного удаленному предложению, и некоторая модификация целевых предложений. Заметим, что такого рода действия могут быть осуществлены только исходя их содержательных представлений о решаемой задаче, т.к. в общей постановке проблема определения перечислимости множеств по их формальному описанию неразрешима.

П2.8.2. Цель со свободными переменными

Если целевое предложение содержит вхождения свободных переменных, то результатом вычисления такого предложения является перечислимый предикат, заданный на свободных переменных. Вычисление такого предиката позволяет определить наборы значений переменных, на которых целевое утверждение выполнимо. В общем случае области интерпретации свободных переменных не могут быть актуально бесконечными, так как результат становится практически невычислимым.

Для решения проблемы вычислимости разработано множество стратегий (эвристик), методов и алгоритмов сокращения пространства поиска[89], в том числе и путем предварительного вычисления (ограничения) предметных множеств, на которых выполняется вывод. Однако далее будем придерживаться прагматического подхода, заключающегося в предоставлении пользователю понятийной модели возможности заранее, до проверки выполнимости целевого утверждения, ограничить (задать) области интерпретации свободных переменных. Это позволит разумно учесть содержательные представления относительно предметной области и сократить пространство поиска до приемлемого размера, вытекающего из особенностей решаемой задачи.

П2.8.3. Итерации по перечислимым предикатам

Формальное использование целевых предложений как со свободными, так и со связанными переменными без учета особенностей предметной области и решаемых в ней задач, приводит к проблеме вычислимости итогового результата. Однако при конечном числе элементов в областях интерпретации свободных переменных такие проблемы не возникают.

Поставим целью так дополнить разработанную нами понятийную модель, чтобы появилась возможность связывания свободных переменных конечными множествами. Это

[89] См. *Лорьер Ж.-Л.* Системы искусственного интеллекта. – М.: Мир, 1991.

позволит реализовать целевое предложение в виде итеративного (циклического) ввода фактов в некоторый перечислимый предикат.

```
#\-------------------------------------------------------------------------------------------------------------------
#\ Итерация по перечислимым предикатам
#\      '∇' Variable ',' …'∈' Enumerable ' ' '.'        #\ Итератор свободных переменных
#\-------------------------------------------------------------------------------------------------------------------
() ()
        "∇|for"                         #\ Итератор свободных переменных
        [
                #\ Инициализация счетчика переменных
                #i0_ #i1_ #tput_        #\ записать количество переменных 0 в ячейку 1

                #\ Запись кода литерала для идентификатора тела итератора
                @dword_                 #\ код целочисленного литерала
                #csize_                 #\ получить текущий размер формируемого кода
                #dec_                   #\ вычислить смещение тела литерала
                #i2_ #tput_             #\ сохранить смещение тела литерала в ячеке 2
                @i0_ #cdword_           #\ запись в тело литерала 4 нулевых байта
        ]
        Variable `vars` ','
        [
                #\ Если запятая распознана, то вернуться на элемент Variable
                #i1_ #tget_             #\ увеличить количество констант в ячейке 2
                #inc_                   #\ на единицу
                #i1_ #tput_             #\ сохранить новое значение в той же ячейке
                #i2m_ #item_            #\ повторить распознание элемента Variable
        ] [
                #\ Если не распознана, то записать число переменных под алиасом vars
                @stringd_               #\ код строкового литерала
                #i1_ #tget_             #\ количество переменных из ячейки 1
                #cdword_                #\ запись двойного слова в тело литерала
                #i0_ #cdword_           #\ запись конца строки в тело литерала
        ]
        '∈' Enumerable `enum`
        [
                #\ Перенаправить компиляцию в новую область кода
                #cnew_                  #\ создать новую область кода
                #dup_                   #\ скопировать идентификатор нового кода
                #i3_ #tput_             #\ сохранить идентификатор кода в ячейке 3
                #ccode_                 #\ переключится на новую область кода
        ]
        ' '
        [
                #\ Восстановить компиляцию в текущую область кода
```

```
        #i0_ #ccode_            #\ восстановить старую область кода

        #\ Записать идентификатор новой области кода в тело литерала
        #i3_ #tget_             #\ идентификатор кода для тела итератора
        #i2_ #tget_             #\ смещение литерала в формируемом коде
        #cput_                  #\ запись в область формируемого кода
    ]
    '.|end'
    {
    #\ Объявление временных переменных
        Variable var, kor, kors, body
    #\ Идентификатор кода тела итератора
        body = operand( vars + 1 )
    #\ Количество кортежей в предикате
        kors = Domain enum () / vars
    #\ Цикл по всем кортежам
        kor = kors
        while
            kor > 0
        do
        #\ Уменьшить номер кортежа
            kor = kor – 1
        #\ Присвоить значения переменным
            for var = 1, vars
                reference( var ) = Domain enum ( kor*vars + var )
            end
        #\ Выполнить код тела итератора
            push( body )
            #exec_
        end
    #\ Удаление переменных и идентификатора тела итератора из стека
        free vars + 1
    }
#\-----------------------------------------------------------------------------------------------------------
```

Фраза целевого высказывания начинается квантором цели (итерации) ∇ и завершается точкой. Этот позволяет отличать целевые высказывания от ввода фактов. Заметим, что в предложении определены два синтаксиса выражения цели: '∇ ... .' и 'for ... end'.

Рассмотрим пример. Пусть необходимо определить выполнимость формулы $x * y > b \,\&\, Q(x, y, a)$ при $a = -1$ и $b = 2$, где $Q(x, y, a)$ – разрешимый предикат «быть решением уравнения $ax + y = 0$», а взаимосвязанные значения свободных переменных, определяющих область интерпретации, заданы перечислимым предикатом R.

```
#\-----------------------------------------------------------------------------------------------------------
<       #\ Ситуационное описание
```

```
        #\ Объявление констант
        Constant a = -1, b = 2

        #\ Объявление переменных
        Variable x, y

        #\ Объявление предикатов
        Predicate 2 P, 2 R

        #\ Ввод фактов
        R(6, 0) ←;
        R(5, 0) ←; R(5, 1) ←;
        R(4, 0) ←; R(4, 1) ←; R(4, 2) ←;
        R(3, 0) ←; R(3, 1) ←; R(3, 2) ←; R(3, 3) ←;
        R(4, 0) ←; R(2, 1) ←; R(2, 2) ←; R(2, 3) ←; R(2, 4) ←;
                    R(1, 1) ←; R(1, 2) ←; R(1, 3) ←; R(1, 4) ←; R(1, 5) ←;
                                R(0, 2) ←; R(0, 3) ←; R(0, 4) ←; R(0, 5) ←; R(0, 6) ←;

        #\ Правило вывода
        Δ Q(u, v, w) ← w*u + v = 0

        #\ Цель
        ∇ x, y ∈ R   P(x, y) ← x * y > b & Q(x, y, a).

        #\ Отображение результата
        for x, y ∈ P
                message 'P(' & x & ', ' & y & ')'
        end
>
#\----------------------------------------------------------------------------------------------------
```

В результате вычислительной интерпретации ситуационной части получим следующие сообщения:

– «P(2, 2)»;

– «P(3, 3)».

Теперь осталось произвести предметную интерпретацию полученного результата: «прямая a x + y = 0 при a=–1 пересекает заданную в условии задачи предметную область R выше гиперболы, определенной уравнением x * y = b при b=2, в точках (2, 2) и (3, 3)». Заметим, что если бы область не была ограничена, то решением задачи было бы бесконечное множество точек.

П2.9. Предложения понятийной модели

Для систематизации определенных ранее предложения приведем их полный список с привязкой к тем понятиям, которые они выражают.

```
#\-----«Пустое» понятие ----------------------------------------------------------------------------
() ()
        '; '                                    #\ Знак пунктуации
        'if' Proposition 'then' '' 'else' '' 'end'      #\ Условный оператор
        'while' Proposition 'do' '' 'end'       #\ Цикл с предусловием
        'do' '' 'while' Proposition 'end'       #\ Цикл с постусловием
        'for' Variable '=' Term ',' Term '' 'end'       #\ Итераторационный цикл
        'Variable' Name '=' Term ','            #\ Объявление и инициализация переменной
        Variable '=' Term ','                   #\ Присваивание переменной значения
        Domain '=' Term ','                     #\ Инициализация домена списком констант
        'Predicate' "[1-9][0-9]" Name ','       #\ Объявление предиката
        Enumerable '(' Term ',' ')' '←' Proposition#\ Добавить факт
        'Δ' Name '(' Name ',' ')' '←' Proposition   #\ Определить правило вывода
        '∇' Variable ',' '∈' Enumerable '' '.'      #\ Доказать целевое предложение

#\------ Имя ------------------------------------------------------------------------------------------
() Name ()
        "[A-ZA-Я][a-za-я0-9]*"                  #\ Синтаксис имен

#\------ Константа ------------------------------------------------------------------------------------
() Constant ()
        "-?[1-9][0-9]*"                         #\ Константа – десятичное целое число
        Domain '(' ')'                          #\ Количество констант в домене
        Domain '(' Term ')'                     #\ Извлечение константы по номеру
        Domain '[' ']'                          #\ Количество переменных в домене
        'Arity' Enumerable                      #\ Местность перечислимого предиката

#\------ Переменная -----------------------------------------------------------------------------------
() Variable (Constant)
        Domain '[' Term ']'                     #\ Домен как массив переменных

#\------ Функтор --------------------------------------------------------------------------------------
(Constant Variable) Functor ()
        '(' Functor ')'                         #\ Арифметические скобки
        '|' Functor '|'                         #\ Модуль
        '-' Functor                             #\ Изменение знака
        Functor '%' Functor                     #\ Остаток от деления
        Functor '/' Functor                     #\ Целая часть деления
        Functor '*' Functor                     #\ Умножение
        Functor '-' Functor                     #\ Вычитание
        Functor '+' Functor                     #\ Сложение
```

```
        Functor '?' Functor ':' Functor              #\ Арифметическое ветвление

#\------ Терм ----------------------------------------------------------------------------------------------
(Constant Variable Functor) Term ()

#\------ Разрешимый предикат -------------------------------------------------------------------------------
() Solvable ()
        'false'                                      #\ Логическая константа «ложь»
        'true'                                       #\ Логическая константа «истина»
        Term '=' Term                                #\ Отношение «равно»
        Term '>' Term                                #\ Отношение «больше»
        Term '<' Term                                #\ Отношение «меньше
        Enumerable '(' Term ',' ')'                  #\ Выполнимость перечислимого предиката

#\------ Домен ---------------------------------------------------------------------------------------------
() Domain (Constant)
        'Domain' Enumerable                          #\ Домен перечислимого предиката

#\------ Перечислимый предикат -----------------------------------------------------------------------------
() Enumerable (Domain Constant Solvable)

#\------ Предикат ------------------------------------------------------------------------------------------
(Solvable Enumerable) Predicate ()

#\------ Высказывание --------------------------------------------------------------------------------------
(Predicate) Proposition ()
        '(' Proposition ')'                          #\ Скобки (порядок интерпретации)
        '¬' Proposition                              #\ Отрицание (логическое «не»)
        Proposition '&' Proposition                  #\ Конъюнкция (логическое «и»)
        Proposition '∨' Proposition                  #\ Дизъюнкция (логическое «или»)
        Proposition '→' Proposition                  #\ Импликация (логический вывод)
        Proposition '↔' Proposition                  #\ Эквиваленция (логическая равносильность)
        "∃|∀" Variable',' '∈' Enumerable Proposition     #\ Высказывания с кванторами
#\-------------------------------------------------------------------------------------------------------------
```

В понятийной модели определена только прагматика по умолчанию, реализующая вычислительную семантику классического исчисления предикатов. В случае необходимости, могут быть определены дополнительные прагматики, выражающие иные вычислительные семантики, например, семантику исчисления предикатов при разомкнутом мире. Заметим, что для любой новой прагматики тексты логических программ останутся синтаксически корректными, изменится лишь их вычислительная интерпретация.

П2.10. Предметная декомпозиция

На практике встречаются постановки задач, в которых используются формулы, определяемые на счетных или актуально бесконечных (практически не перечислимых) множествах. Проверка выполнимости таких формул не представляется возможной, ибо связана с перебором большого числа решений.

Иногда для решения сложных прикладных задач используется не прямое вычисление таких формул, а их тождественные преобразования, позволяющие свести сложную задачу к более простой и эквивалентной исходной. Ожидается, что решение, полученное в результате тождественных преобразований, потребует, в конечном итоге, вычисления формул на конечных множествах.

Известно, что проблема выводимости в логике предикатов первого порядка неразрешима. Более того, принято считать[90], что автоматические рассуждения на основе чисто синтаксических методов преобразования формул не способны обработать огромное пространство поиска при решении задач практической сложности. Следовательно, для решения проблемы вычислимости формул в исчислении предикатов необходимо по-другому осуществить общую постановку задачи.

Особенностью рассмотренной выше реализации исчисления предикатов является перечислимость областей интерпретации свободных переменных. Это обстоятельство обусловлено тем, что для вычисления выполнимости формул с кванторами необходимо повторение вычислений при всех возможных значениях как свободных, так связанных переменных.

На основе понятийного анализа исчисления предикатов найдено, что для решения проблемы вычислимости формул требуется рассматривать два типа предикатов:

– вычислимые предикаты, которые могут использоваться для определения как конечных, так и актуально бесконечных разрешимых множеств;

– перечислимые предикаты, которые используются только для задания конечных (перечислимых) множеств.

Такое разделение позволит во время описания задачи так определить формулы, что проверка их выполнимости будет сведена к итеративным вычисления на конечных множествах; или, путем использования известных аналитических методов определения выполнимости предикатов, – на актуально бесконечных множествах.

В этом случае особое значение приобретает не формализация логического вывода в общей постановке задачи, которая приводит к проблеме вычислимости, а разумная пред-

[90] *Вагин В. Н., Головина Е. Ю., Загорянская А. А. Фомина М. В.* Достоверный и правдоподобный вывод в интеллектуальных системах. М.: Физматлит, 2004.

метная декомпозиция предметной области, следующая из содержательных представлений о решаемых задачах и свойствах их решения. В последнем случае основной целью предметной декомпозиции является получение вычислимого решения стоящей прикладной задачи.

В отличие от известных языков логического программирования, разработанная понятийная модель содержит выразительные средства, которые побуждают пользователя так формулировать задачи, чтобы они имели конечное решение. Для выполнения этого требования в созданном языке логики предикатов в предложениях цели запрещено использование свободных переменных, а для их связывания введен квантор ∇.

Однако, если потребуется получить решения прикладных задач, описываемых бесконечными и актуально бесконечными множествами, то единственным конструктивным путем решения будет использование синтаксических методов, которые, путем преобразования текста исходной задачи в текст решения, позволят выразить такие множества в виде конечной последовательности знаков. Но даже в этом случае, нет гарантии успеха, так как существуют неразрешимые множества, которые невозможно описать конечными средствами. Для решения такого рода задач необходимо по-другому осуществить предметную декомпозицию, т.е. так выполнить постановку задачи, чтобы задача имела конечное решение[91].

Таким образом, применение понятийного анализа и контекстной технологии видится перспективным, так как позволяет в естественной форме выразить результат предметной декомпозиции. В этом случае на долю человека выпадают задачи содержательной постановки и корректного описания решаемых задач на существующем или создаваемом предметном языке, а вычислительной системе остаются формальные манипуляции на основе заданной или определяемой пользователем вычислительной семантики.

[91] Прием, который используется в таких случаях, может быть продемонстрирован на примере иррациональных чисел: если конечными средствами невозможно выразить некоторые числа (основание натурального логарифма, коэффициент пропорциональности между длиной окружности и диаметром), то введем для них новое понятие – иррациональное число, а сущности, принадлежащие этому понятию, обозначим специальными знаками (e, π).

Приложение 3.
Виртуальная машина

Настоящее приложении является справочным. В нем описана универсальная виртуальная машина Kernel, которая использована для демонстрации возможностей контекстной технологии на примере реализации исчисления предикатов первого порядка.

Виртуальная машина предназначена для создания специальной среды исполнения мобильного, безопасного и устойчивого программного кода. Под мобильностью кода понимается возможность выполнения одного и того же адресуемого фрагмента программы (императива) на разных вычислительных платформах или типах виртуальных машин. Под безопасностью кода понимается встроенная в реализацию базовых примитивов вирутальной машины (в ее команды) защита от несанкционированного доступа к ресурсам вычислительной платформы. Устойчивость кода предполагает возможность обнаружения ошибок, связанных с некоррекностью операций, совершаемых в процессе выполнения программы.

Базовый механизм, позволяющий обеспечить перечисленные выше свойства среды исполнения, основывается на использовании специального формата представления кода – промежуточного кода, который является результатом работы семантического анализатора системы программирования и сохраняет многие семантические свойства программ. Аналогичный подход в промышленных масштабах впервые использовался в технологии Java (Java Virtual Machine).[92] Технология компилирующих виртуальных машин разрабатывалась также в рамках проекта «Эльбрус».[93]

П3.1. Вычислительный механизм

Рассматриваемая виртуальная машина относится к вычислительным устройствам с линейно-ограниченной и стековой вычислительными моделями и реализует возможности современных вычислительных средств с неймановской архитектурой.

На рис. П3.1 показано различие между промежуточным и исполняемым кодом. Промежуточный код формируется системой контекстного программирования и состоит из команд виртуальной машины. Исполняемый код создается виртуальной машиной для каждой области промежуточного кода путем его преобразования в команды аппаратной платформы и, возможно, в вызовы операционной системы.

---

[92] Engel J. Programming for the Java Virtual Machine. Addison-Wesley, 1999.

[93] (Babayan B. Main Principles of E2K Architecture // Free Software Magazine. 2002. Vol. 1, Issue 02. PP.13-26).

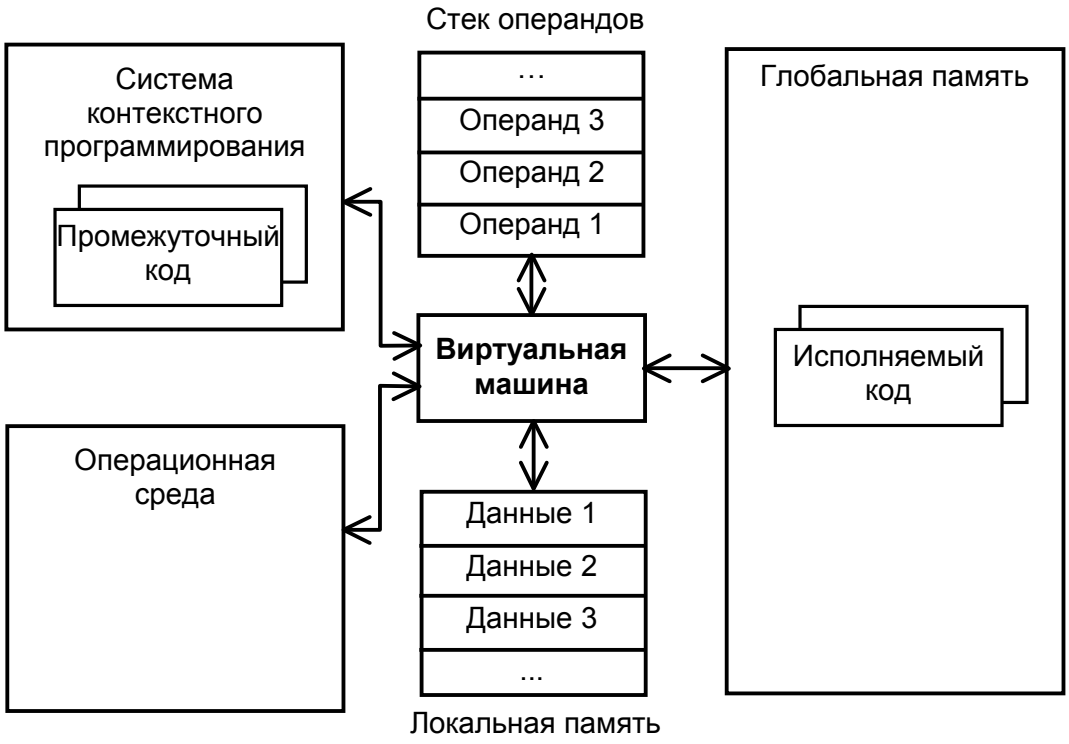

Рис. П3.1. Организация вычислений

Непосредственному исполнению подлежит только область исполняемого кода. Виртуальная машина организует преобразование промежуточного кода в исполняемый, если такое преобразование еще не было выполнено. После преобразования осуществляется запуск сформированного исполняемого кода средствами аппаратной платформы и/или операционной среды.

П3.2. Система команд

Рассмотрим систему команд виртуальной машины Kernel. Каждая команда кодируется одним байтом и описана директивой препроцессора контекстной системы программирования.

Директива '#define имя значение # комментарий' устанавливает текстовую эквивалентность имени его значению. Имена команд заданы мнемоническими обозначениями, а значения – числами в десятичной системе счисления, которые являются кодами этих команд.

Для пояснения действий, выполняемых командами, используется комментарий. В комментариях также содержится нотация команды – состояние стека операндов до и после выполнения команды. Операнды обозначаются содержательными именами и заключаются в квадратные скобки. Если в комментарии присутствует обозначение операнда в наклонных скобках /xxx/, то это сигнализирует о том, что указанный в скобках операнд xxx

является непосредственными данными команды. Непосредственные данные располагаются в области кода сразу после кода команды.

В заключительной части приложения описаны сервисы обратного вызова, реализуемые системой контекстного программирования. Сервисы обратного вызова предназначены для выполнения тех действий, которые не могут быть реализованы командами аппаратной платформы или операционной средой.

При описании команд в комментариях и нотациях использованы следующие стандартные сокращения:

a, b, c – операнды одинарной точности (одна ячейка);

A, B, C – операнды двойной точности (две ячейки);

bit – Bit (номер бита);

byt – Byte (байт);

cod – Code (идентификатор области промежуточного кода);

dst – Destination (приемник данных, указатель на область памяти);

dwd – Double Word (двойное слово, четыре байта);

fil – File (указатель на имя файла);

han – Handle (дескриптор ресурса);

lab – Label (идентификатор метки в области промежуточного кода);

loc – Local (локальный адрес);

num – Number (количество сдвигов, номер ячейки);

obj – Object (указатель на объект и/или интерфейс);

ofs – Offset (смещение ячейки памяти);

pro – Procedure (указатель на процедуру);

ptr – Pointer (указатель на ячейку памяти);

qwd – Quadric Word (четверное слово, восемь байт);

siz – Size (размер области памяти);

src – Source (источник данных, указатель на область памяти);

stk – Stack (номер ячейки в стеке);

val – Value (значение, записанное в ячейке памяти);

wrd – Word (слово, два байта).

П3.3. Команды организация исполнения

П3.3.1. Структурирование и исполнение

```
#define     error_      000         # Генерация ошибки [ ]→[ ]
#define     debug       001         # Запуск отладчика [ ]→[ ]
#define     code_       002         # Вызов императива /cod/ [ ]→[ ]
```

```
#define   exec_      003     # Выполнение императива [cod]→[ ]
#define   try_       004     # Начало секции перехвата исключений /lab/ [ ]→[ ]
#define   throw_     005     # Генерация исключения [val ... val siz]→[ ]
#define   catch_     006     # Окончание секции перехвата исключений /lab/ [ ]→[ ]
#define   label_     007     # Метка в секции кода [ ]→[ ]
```

П3.3.2. Обращение к внешним модулям

```
#define   load_      008     # Загрузка внешнего модуля [fil]→[han]
#define   free_      009     # Выгрузка внешнего модуля [han]→[ ]
#define   proc_      010     # Получить адрес процедуры [nam han]→[pro]
#define   call_      011     # Вызов процедуры [... par par pro]→[res]
#define   construct_ 012     # Создание внешенего объекта [nam]→[obj]
#define   destruct_  013     # Уничтожение внешнего объекта [obj]→[ ]
#define   interface_ 014     # Получение указателя на интерфейс [obj]→[obj]
#define   invoke_    015     # Вызов метод интерфейса [method obj]→[res]
```

П3.4. Стековая память

П3.4.1. Выделение и освобождение памяти, пересылка данных

```
#define   snew_      016     # Выделение стековой памяти [siz]→[stk]
#define   sfree_     017     # Освобождение стековой памяти [stk]→[ ]
#define   sget_      018     # Чтение ячеки памяти [stk]→[val]
#define   sput_      019     # Запись в ячеку памяти [val stk]→[ ]
#define   scopy_     020     # Копирование ячеек памяти [stk siz]→[val ... val]
#define   smove_     021     # Пересылка ячеек памяти [stk siz]→[ ]
#define   sptr_      022     # Глобальный адрес [stk]→[ptr]
#define   ssize_     023     # Размер (глубина) стека [ ]→[siz]
```

П3.4.2. Операции с ячейками стековой памяти

```
#define   nop_       024     # Нет операции [ ]→[ ]
#define   swap_      025     # Перестановка верхних ячеек стека [b a]→[a b]
#define   dup_       025     # Копирование ячейки на вершине стека [a]→[a a]
#define   drop_      027     # Удаление ячейки на вершине стека [b a]→[b]
#define   over_      028     # Копирование ячейки под вершиной [b a]→[b a b]
#define   press_     029     # Удаление ячейки под вершиной [b a]→[a]
#define   rotr_      030     # Циклическая перестановка ячеек вправо [c b a]→[ a c b]
#define   rotl_      031     # Циклическая перестановка ячеек вправо [c b a]→[b a c]
```

П3.5. Локальная память

П3.5.1. Выделение и освобождение памяти, пересылка данных

```
#define   rnew_      032     # Выделение локальной памяти [siz]→[loc]
#define   rfree_     033     # Освобождение локальной памяти [loc]→[ ]
#define   rget_      034     # Чтение ячеки локальной памяти [loc]→[val]
#define   rput_      035     # Запись в ячеку локальной памяти [val loc]→[ ]
#define   rcopy_     036     # Копирование ячеек памяти [loc siz]→[val … val]
#define   rmove_     037     # Пересылка ячеек памяти [val val ... val loc siz ]→[ ]
```

```
#define     rptr_       022     # Глобальный адрес [±loc]→[ptr]
#define     rsize_      039     # Размер локальной памяти [ ]→[siz]
```

П3.5.2. Операции с ячейками локальной памяти

```
#define     rpush_      040     # Сохранить в локальной памяти [a]→[ ]
#define     rpop_       041     # Восстановить из локальной памяти [ ]→[a]
#define     rdup_       042     # Копировать в локальную память [a]→[a]
#define     rover_      043     # Копировать локальную память [b a]→[b a]
#define     rread_      044     # Прочитать ячеку локальной памяти [ ]→[a]
#define     rwrite_     045     # Записать ячеку локальной памяти [a]→[ ]
#define     rload_      046     # Прочитать ячеку с глубины [loc]→[a]
#define     rstore_     047     # Записать ячеку на глубину [a loc]→[ ]
```

П3.6. Глобальная память

П3.6.1. Выделение и освобождение памяти, пересылка данных

```
#define     mnew_       048     # Выделение памяти [siz]→[ptr]
#define     mfree_      049     # Освобождение памяти [ptr]→[ ]
#define     mget_       050     # Чтение ячеки [ptr ofs]→[val]
#define     mput_       051     # Запись в ячеку [val ptr ofs]→[ ]
#define     mcopy_      052     # Копирование ячеек в стек [ptr ofs siz]→[val … val]
#define     mmove_      053     # Пересылка ячеек из стека [val val ... val ptr ofs siz]→[ ]
#define     mptr_       054     # Получить указатель [ptr ofs]→[ptr]
#define     msize_      055     # Размер глобальной памяти [ptr]→[siz]
```

П3.6.2. Операции с ячейками глобальной памяти

```
#define     getb_       056     # Переслать байт из памяти в стек [ptr]→[val]
#define     putb_       057     # Переслать байт из стека в память [val ptr]→[ ]
#define     getw_       058     # Переслать слово глобальной памяти в стек [ptr]→[val]
#define     putw_       059     # Переслать слово из стека в память [val ptr]→[ ]
#define     getd_       060     # Переслать двойное слово в стек [ptr]→[val]
#define     putd_       061     # Переслать двойное слово из стека [val ptr]→[ ]
#define     getq_       062     # Переслать два двойных слова в стек [ptr]→[val val]
#define     putq_       063     # Переслать два двойных слова из стека [val val ptr]→[ ]
```

П3.7. Литералы

П3.7.1. Литералы фиксированной длинной

```
#define     byte_       064     # Байт (безнаковое целое) /byt/ [ ]→[val]
#define     tiny_       065     # Байт (целое со знаком) /byt/ [ ]→[ val]
#define     word_       066     # Слово (безнаковое целое) /wrd/ [ ]→[ val]
#define     integer_    067     # Слово (целое со знаком) /wrd/ [ ]→[val]
#define     dword_      068     # Двойное слово (безнаковое целое) /dwd/ [ ]→[val]
#define     long_       069     # Двойное слово (целое со знаком) /dwd/ [ ]→[val]
#define     qword_      070     # Четверное слово (безнаковое целое) /qwd/ [ ]→[val val]
#define     large_      071     # Четверное слово (целое со знаком) /qwd/ [ ]→[val val]
```

П3.7.2. Литералы переменной длины

```
#define     stringb_    072     # Строка байт /byt … byt 0/ [ ]→[ptr]
#define     stringw_    073     # Строка слов /wrd … wrd 0/ [ ]→[ptr]
#define     stringd_    074     # Строка двойных слов /dwd … dwd 0/ [ ]→[ptr]
#define     stringq_    075     # Строка четверных слова /qwd … qwd 0/ [ ]→[ptr]
#define     datab_      076     # Данные (байты) /siz byt … byt/ [ ]→[ptr siz]
#define     dataw_      077     # Данные (слова) /siz wrd … wrd/ [ ]→[ptr siz]
#define     datad_      078     # Данные (двойные слова) /siz dwd … dwd/ [ ]→[ptr siz]
#define     dataq_      079     # Данные (четверные слова) /siz qwd … qwd/ [ ]→[ptr siz]
```

П3.8. Целые константы

П3.8.1. Положительные константы

```
#define     i0_         080     # Целое 0 [ ]→[0]
#define     i1_         081     # Целое 1 [ ]→[1]
#define     i2_         082     # Целое 2 [ ]→[2]
#define     i3_         083     # Целое 3 [ ]→[3]
#define     i4_         084     # Целое 4 [ ]→[4]
#define     i5_         085     # Целое 5 [ ]→[5]
#define     i6_         086     # Целое 6 [ ]→[6]
#define     i7_         087     # Целое 7 [ ]→[7]
```

П3.8.2. Отрицательные константы

```
#define     i7m_        088     # Целое -7 [ ]→[-7]
#define     i6m_        089     # Целое -6 [ ]→[-6]
#define     i5m_        090     # Целое -5 [ ]→[-5]
#define     i4m_        091     # Целое -4 [ ]→[-4]
#define     i3m_        092     # Целое -3 [ ]→[-3]
#define     i2m_        093     # Целое -2 [ ]→[-2]
#define     i1m_        094     # Целое -1 [ ]→[-1]
#define     i128_       095     # Целое 128 [ ]→[128]
```

П3.9. Битовые операции

П3.9.1. Операции с адресуемыми битами

```
#define     btst_       096     # Проверка бита [val bit]→[val] + CF=bit
#define     bcom_       097     # Инверсия бита [val bit]→[val] + CF=bit
#define     bset_       098     # Установка бита [val bit]→[val] + CF=bit
#define     bclr_       099     # Сброс бита [val bit]→[val] + CF=bit
#define     bsf_        100     # Поиск бита вперед [val]→[val bit]
#define     bsb_        101     # Поиск бита назад [val]→[val bit]
#define     bget_       102     # Прочитать флаги условий [ ]→[val]
#define     bput_       103     # Записать флаги условий [val]→[ ]
```

П3.9.2. Операции с флагом переноса

```
#define     cclr_       104     # Сброс флага переноса [ ]→[ ] CF=0
```

```
#define   cset_    105    # Установка флага переноса [ ]→[ ] CF=1
#define   ccom_    106    # Инверсия флага переноса [ ]→[ ] CF=¬CF
#define   cbit_    107    # Записать флаг переноса в ячейку [val bit]→[val]
#define   cinc_    108    # Сложить флаг переноса с ячейкой [val]→[val+CF]
#define   cdec_    109    # Вычесть флаг переноса из ячейки [val]→[val-CF]
#define   cadd_    110    # Сложение с учетом флага переноса [b a]→[b+a+CF]
#define   csub_    111    # Вчитание с учетом флага переноса [b a]→[b-a-CF]
```

П3.10. Поразрядные операции

П3.10.1. Операции сдвигов

```
#define   shl_     112    # Логический сдвиг влево [val num]→[val]
#define   shr_     113    # Логический сдвиг вправо [val num]→[val]
#define   sal_     114    # Арифметический сдвиг влево [val num]→[val]
#define   sar_     115    # Арифметический сдвиг вправо [val num]→[val]
#define   rol_     116    # Циклический сдвиг влево [val num]→[val]
#define   ror_     117    # Циклический сдвиг вправо [val num]→[val]
#define   rcl_     118    # Циклический сдвиг влево с переносом [val num]→[val]
#define   rcr_     119    # Циклический сдвиг вправо с переносом [val num]→[val]
```

П3.10.2. Логические операции

```
#define   cvs_     120    # Преобразовать бит [a]→[b]
#define   cvb_     121    # Преобразовать байт [a]→[b]
#define   cvw_     122    # Преобразовать слово [a]→[v]
#define   cvd_     123    # Преобразовать двойное слово [a]→[b]
#define   not_     124    # Поразрядное отрицание [a]→[¬a]
#define   and_     125    # Поразрядная конъюнкция [b a]→[b&a]
#define   xor_     126    # Поразрядная неэквиваленция [b a]→[b⊕a]
#define   or_      127    # Поразрядная дизъюнкция [b a]→[b∨a]
```

П3.11. Целочисленная арифметика

П3.11.1. Проверка условий и унарные операции

```
#define   cnd_     128    # Сформировать условия [a]→[ ]
#define   tst_     129    # Сравнить с нулем [a]→[a]
#define   abs_     130    # Модуль числа [a]→[|a|]
#define   neg_     131    # Изменение знака [a]→[–a]
#define   inc_     132    # Увеличение на единицу [a]→[a+1]
#define   dec_     133    # Уменьшение на единицу [a]→[a-1]
#define   umul_    134    # Беззнаковое умножение [b a]→[b∗a]
#define   udiv_    135    # Беззнаковое деление [b a]→[b/a]
```

П3.11.2. Бинарные операции

```
#define   add_     136    # Сложение [b a]→[b+a]
#define   sub_     137    # Вычитание [b a]→[b–a]
#define   cmp_     138    # Сравнение [b a]→[b]
```

```
#define     bound_      139     # Проверка диапазона [c b a]→[c]
#define     mul_        140     # Знаковое умножение [b a]→[b*a]
#define     div_        141     # Знаковое деление [b a]→[b/a]
#define     mod_        142     # Остаток от деления [b a]→[b%a]
#define     rmd_        143     # Целая часть и остаток деления [b a]→[b/a b%a]
```

П3.12. Условные и безусловные переходы

П3.12.1. Переходы по состоянию вершины стека

```
#define     jz_         144     # По нулю ячейки на вершине /lab/ [val]→[ ]
#define     jnz_        145     # По не нулю ячейки на вершине /lab/ [val]→[ ]
#define     js_         146     # По знаку ячейки на вершине /lab/ [val]→[ ]
#define     jns_        147     # По не знаку ячейки на вершине /lab/ [val]→[ ]
#define     jp_         148     # По паритету ячейки на вершине /lab/ [val]→[ ]
#define     jnp_        149     # По не паритету ячейки на вершине /lab/ [val]→[ ]
#define     jodd_       150     # По четности ячейки на вершине /lab/ [val]→[ ]
#define     jeven_      151     # По нечетности ячейки на вершине /lab/ [val]→[ ]
```

П3.12.2. Переходы по флагам 1

```
#define     je_         152     # По флагу равенства /lab/ [ ]→[ ]
#define     jne_        153     # По флагу не неравенства /lab/ [ ]→[ ]
#define     ja_         154     # По флагам больше (беззнаковый) /lab/ [ ]→[ ]
#define     jbe_        155     # По флагам меньше или равно (беззнаковый) /lab/ [ ]→[ ]
#define     jl_         156     # По флагам меньше (знаковый) /lab/ [ ]→[ ]
#define     jge_        157     # По флагам больше или равно (знаковый) /lab/ [ ]→[ ]
#define     jg_         158     # По флагам больше (знаковый) /lab/ [ ]→[ ]
#define     jle_        159     # По флагам меньше или равно (знаковый) /lab/ [ ]→[ ]
```

П3.12.3. Переходы по флагам 2

```
#define     jc_         160     # По флагу переноса /lab/ [ ]→[ ]
#define     jb_         160     # По флагам мешьше (беззнаковый) /lab/ [ ]→[ ]
#define     jnc_        161     # По флагу не переноса /lab/ [ ]→[ ]
#define     jae_        161     # По флагам больше или равно (беззнаковый) /lab/ [ ]→[ ]
#define     jo_         162     # По флагу переполнения /lab/ [ ]→[ ]
#define     jno_        163     # По флагу не переполнения /lab/ [ ]→[ ]
#define     jpe_        164     # По флагу четного паритета /lab/ [ ]→[ ]
#define     jpo_        165     # По флагу нечетного паритета /lab/ [ ]→[ ]
#define     jaux_       166     # По флагу дополнительного переноса /lab/ [ ]→[ ]
#define     jmp_        167     # Переход безусловный /lab/ [ ]→[ ]
```

П3.13. Константы с плавающей запятой

```
#define     f0_         168     # Нуль 0.0 [ ]→[A]
#define     f1_         169     # Единица 1.0 [ ]→[A]
#define     f2_         170     # Двойка 2.0 [ ]→[A]
#define     fpi_        171     # Число пи [ ]→[A]
```

```
#define    floge_    172    # Двоичный логарифм числа e [ ]→[A]
#define    flog10_   173    # Двоичный логарифм 10 [ ]→[A]
#define    flg2_     174    # Десятичный логарифм 2 [ ]→[A]
#define    fln2_     175    # Натуральный логарифм 2 [ ]→[A]
```

П3.14. Операции над числами с плавающей запятой

П3.14.1. Операции с ячейками стека

```
#define    fexam_    176    # Проверка типа числа [A]→[A]
#define    fswap_    177    # Перестановка чисел [B A]→[A B]
#define    fdup_     178    # Копирование числа [A ]→[A A]
#define    fdrop_    179    # Удаление числа [B A]→[B]
#define    fover_    180    # Копирование числа [B A]→[B A B]
#define    fpress_   181    # Удаление числа [B A]→[ A]
#define    frotr_    182    # Цикл вправо [C B A]→[A C B]
#define    frotl_    183    # Цикл влево [C B A]→[B A C]
```

П3.14.2. Унарные операции

```
#define    fcnd_     184    # Заменить число на флаги [A]→[ ]
#define    ftst_     185    # Установить флаги [A]→[A]
#define    fabs_     186    # Модуль [A]→[B]
#define    fneg_     187    # Изменение знака [A]→[B]
#define    finc_     188    # Увеличить на 1 [A]→[B]
#define    fdec_     189    # Уменьшить на 1 [A]→[B]
#define    fpush_    190    # Сохранить в локальной памяти [A]→[ ]
#define    fpop_     191    # Восстановить из локальной памяти [ ]→[A]
```

П3.14.3. Бинарные операции

```
#define    fadd_     192    # Сложение чисел прямое [B A]→[C]
#define    fsub_     193    # Вычитание чисел прямое [B A]→[C]
#define    fsubr_    194    # Вычитание обратное [B A]→[C]
#define    fcmp_     195    # Сравнение чисел [B A]→[B]
#define    fcmpr_    196    # Сравнение обратное [B A]→[A]
#define    fmul_     197    # Умножение чисел [B A]→[C]
#define    fdiv_     198    # Деление чисел прямое [B A]→[C]
#define    fdivr_    199    # Деление чисел обратное [B A]→[C]
```

П3.14.4. Элементарные функции

```
#define    finv_     200    # Вычисление обратного числа [A]→[B]
#define    fsqrt_    201    # Квадратный корень [A]→[B]
#define    flog_     202    # Двоичный логарифм [A]→[lB]
#define    fpow_     203    # Возведение 2 в степень [A]→[B]
#define    fsin_     204    # Синус [A]→[B]
#define    fcos_     205    # Косинус [A]→[B]
#define    ftg_      206    # Тангенс [A]→[B]
#define    fctg_     207    # Котангенс [A]→[B]
```

П3.14.5. Преобразования чисел

```
#define   fcvf_     208     # Преобразовать из целого [val]→[A]
#define   fcvi_     209     # Преобразовать целое [A]→[val]
#define   fcvd_     210     # Преобразовать в двойную точность [a0]→[A]
#define   fcvs_     211     # Преобразовать в одинарную точность [A]→[a]
#define   fscale_   212     # Масштабирование числа [A val]→[B]
#define   fround_   213     # Округление числа [A val]→[B]
#define   fexp_     214     # Порядок числа [A]→[val]
#define   fman_     215     # Мантиса числа [A]→[B]
```

П3.15. Операции со строками

П3.15.1. Очистка и поиск

```
#define   fillb_    216     # Запись байта [val dst siz]→[ ]
#define   fillw_    217     # Запись слова [val dst siz]→[ ]
#define   filld_    218     # Запись двойного слова [val dst siz]→[ ]
#define   fillq_    219     # Запись четверного слова [val val dst siz]→[ ]
#define   findb_    220     # Поиск байта [val dst siz]→[ofs]
#define   findw_    221     # Поиск слова [val dst siz]→[ofs]
#define   findd_    222     # Поиск двойного слова [val dst siz]→[ofs]
#define   findq_    223     # Поиск четверного слова [val val dst siz]→[ofs]
```

П3.15.2. Загрузка и выгрузка

```
#define   loadb_    224     # Загрузка байт [src siz]→[val … val]
#define   loadw_    225     # Загрузка слов [src siz]→[val … val]
#define   loadd_    226     # Загрузка двойных слов [src siz]→[val … val]
#define   loadq_    227     # Загрузка четверных слов [src siz]→[val … val]
#define   storeb_   228     # Выгрузка байт [val … val dst siz]→[ ]
#define   storew_   229     # Выгрузка слов [val … val dst siz]→[ ]
#define   stored_   230     # Выгрузка двойных слов [val … val dst siz]→[ ]
#define   storeq_   231     # Выгрузка четверных слов [val … val dst siz]→[ ]
```

П3.15.3. Сравнение и пересылка

```
#define   cmpb_     232     # Сравнение байт [src dst siz]→[ofs]
#define   cmpw_     233     # Сравнение слов [src dst siz]→[ofs]
#define   cmpd_     234     # Сравнение двойных слов [src dst siz]→[ofs]
#define   cmpq_     235     # Сравнение четверных слов [src dst siz]→[ofs]
#define   movb_     236     # Пересылка байт [src dst siz]→[ ]
#define   movw_     237     # Пересылка слов [src dst siz]→[ ]
#define   movd_     238     # Пересылка двойных слов [src dst siz]→[ ]
#define   movq_     239     # Пересылка четверных слов [src dst siz]→[ ]
```

П3.16. Резерв

```
#define   reserv1_  240     # Резервная команда 1 [ ]→[ ]
#define   reserv2_  241     # Резервная команда 2 [ ]→[ ]
#define   reserv3_  242     # Резервная команда 3 [ ]→[ ]
```

```
#define   reserv4_   243      # Резервная команда 4 [ ]→[ ]
#define   reserv5_   244      # Резервная команда 5 [ ]→[ ]
#define   reserv6_   245      # Резервная команда 6 [ ]→[ ]
#define   reserv7_   246      # Резервная команда 7 [ ]→[ ]
#define   reserv8_   247      # Резервная команда 8 [ ]→[ ]
```

П3.17. Средства расширения

```
#define   service_   248      # Префикс сервисов обратного вызова
#define   prefix1_   249      # Префикс расширения 1
#define   prefix2 _  250      # Префикс расширения 2
#define   prefix3_   251      # Префикс расширения 3
#define   prefix4_   252      # Префикс расширения 4
#define   prefix5_   253      # Префикс расширения 5
#define   prefix6_   254      # Префикс расширения 6
#define   escape_    255      # Префикс расширения команд
```

П3.18. Сервисы обратного вызова

Для вызова сервиса используются средства расширения команд. Вызов сервиса состоит из префикса service_ (код команды 248), после которого идут непосредственные данные – байт номера сервиса. Параметры, необходимые для выполнения сервисных функций, система программирования извлекает из стека виртуальной машины.

Для взаимодействия с виртуальной машиной используется стандартный интерфейс, состоящий из следующих точек входа:.

1). Создание и уничтожение экземпляра виртуальной машины:

```
long    EngineOpen   (void pro, long stk, long loc)[94];
long    EngineClose  (long han);
```

2). Компиляция и исполнение кода:

```
long    EngineImage  (long han, long ptr, long siz, long stk, long loc, void*ptr);
long    EngineExec   (long han, long ptr, long stk);
```

3). Чтение и запись глобальной памяти:

```
long    EngineByte   (long han, long dst, long src);
long    EngineWord   (long han, long dst, long src);
long    EngineDword  (long han, long dst, long src);
long    EngineQword  (long han, long dst, long src);
```

4). Управление глобальной памятью:

```
long    EngineAlloc  (long han, long *ptr, long siz);
```

[94] Описание стандартного интерфейса виртуальной машины дано на языке программирования Си, где для аргументов функций использованы те же обозначения, что и при описании команд виртуальной машины.

```
long    EngineFree    (long han, long ptr, long *num);
long    EngineSize    (long han, long ptr, long *siz);
```

5). Точки входа для работы со строками и данными:

```
long    EngineLength (long han, long src, long *siz);
long    EngineCopy    (long han, long dst, long src);
long    EngineMove    (long han, long dst, long src, long siz);
```

6). Операции со стеком:

```
long    EnginePush    (long han, long val);
long    EnginePop     (long han, long *val);
```

П3.19.1. Сервис компиляции

```
#define    cnew_     #000     # Создать новую область кода [ ]→[cod]
#define    ccode_    #001     # Переключение в область кода [cod]→[ ]
#define    clabel_   #002     # Выделение меткок в области кода [cnt]→[lab]
#define    clocal_   #003     # Выделение локальной памяти для кода [cnt]→[loc]

#define    cbyte_    #004     # Запись байта в область кода [val] →[ ]
#define    cword_    #005     # Запись слова в область кода [val] →[ ]
#define    cdword_   #006     # Запись двойного слова в область кода [val] →[ ]
#define    cqword_   #007     # Запись четверного слова в область кода [val val] →[ ]

#define    cstring_  #008     # Запись строки в область кода [ptr]→[ ]
#define    cdata_    #009     # Запись данных в область кода [dat siz]→[ ]
#define    cprolog_  #010     # Переключится в область пролога кода [ ]→[ ]
#define    cepilog_  #011     # Переключится  в область эпилога кода [ ]→[ ]

#define    csize_    #012     # Получить размер области кода [ ]→[siz]
#define    cresize_  #013     # Установить размер области кода [siz]→[ ]
#define    cput_     #014     # Записать байт в область кода [val siz]→[ ]
#define    cget_     #015     # Прочитать байт из области кода [siz]→[val]
```

П3.20.2. Информационный сервис

```
#define    ipush_    #016     # Установить текущий элемент [val]→[ ]
#define    ipop_     #017     # Удалить текущий элемент [ ]→[val]
#define    iget_     #018     # Получить флаги текущего элемента [ ]→[val]
#define    iset_     #019     # Записать флаги текущего элемента [val]→[ ]

#define    iterm_    #020     # Получить строку текущего элемента [ ]→[ptr]
#define    ivalue_   #021     # Получить значение текущего элемента [ ]→[val]
#define    iopcode_  #022     # Получить идентификатор текущего элемента [ ]→[val]
#define    isize_    #023     # Получить размер текущего элемента [ ]→[val]

#define    ileft_    #024     # Получить левое свойство текущего элемента [ ]→[val]
#define    iright_   #025     # Получить правое свойство текущего элемента [ ]→[val]
```

```
#define    iup_        #026     # Получить верхнее свойство текущего элемента [ ]→[val]
#define    idown_      #027     # Получить нижнее свойство текущего элемента [ ]→[val]

#define    inotion_    #028     # Получить идентификатор понятия [val]→[val]
#define    isentence_  #029     # Получить идентификатор предложения [val]→[val]
#define    iaspect_    #030     # Получить идентификатор аспекта [val]→[val]
#define    icode_      #031     # Получить идентификатор кода [val]→[val]
```

П3.21.3. Сервис контекстных условий

```
#define    xnew_       #032     # Создать предложение [ ]→[ val]
#define    xfree_      #033     # Удалить предложение [val]→[ ]
#define    x02_        #034     #  [ ]→[ ]
#define    x03_        #035     #  [ ]→[ ]

#define    xterm_      #036     # Добавить терм [val val]→[ val]
#define    xpattern_   #037     # Добавить шаблон [val val]→[val]
#define    xnotion_    #038     # Добавить понятие [val val]→[val]
#define    x07_        #039     # [ ]→[ ]

#define    xcode_      #040     # Добавить код компиляции [val val]→[val]
#define    ximper_     #041     # Добавить императив [val val]→[val]
#define    x10_        #042     # [ ]→[ ]
#define    x11_        #043     # [ ]→[ ]
```

П3.22.4. Сервис управления и состояния

```
#define    scomp_      #050     # Версия компилятора [ ]→[val]
#define    sengine_    #051     # Версия виртуальной машины [ ]→[val]
#define    sget_       #048     # Получить режимы компиляции [ ]→[val]
#define    sput_       #049     # Установить режимы компиляции [val]→[ ]

#define    sbreak_     #052     # Прервать компиляцию [ ]→[ ]
#define    scode_      #053     # Выполнить код [cod]→[…]
#define    spriority_  #054     # Установить текущий приоритет предложений [val]→[ ]
#define    sitem_      #055     # Установить текущий элемент предложения [val]→[ ]
```

П3.23.5. Сервис временной и атрибутной памяти

```
#define    tget_       #064     # Чтение временной ячейки по номеру [num]→[val]
#define    tput_       #065     # Запись во временную ячейку по номеру [val num]→[ ]
#define    tload_      #066     # Загрузка временных ячеек [num siz]→[val … val]
#define    tstore_     #067     # Выгрузка временных ячеек [val … val num siz]→[ ]

#define    aget_       #068     # Чтение атрибута по номеру [num]→[val]
#define    aput_       #069     # Запись атрибута по номеру [val num]→[ ]
#define    aload_      #070     # Загрузка атрибутов [num siz]→[val … val]
#define    astore_     #071     # Выгрузка атрибутов [val … val num siz]→[ ]
```

Приложение 4.
Полиномиальные и неполиномиальные формы

Имеется круг задач, решение которых основывается на обработке дискретных данных. К таким задачам можно отнести логическое управление, дискретную оптимизацию, обработку сигналов и изображений, распознавание образов и прогнозирование, принятие решений, моделирование дискретных устройств и т.д.

Традиционная архитектура ЭВМ не предусматривает специализированных устройств для дискретных преобразований, а современное состояние информационной технологии отражает неадекватность используемых методов обработки данных структуре вычислительных средств. В связи с этим наблюдается закономерный интерес к разработке новых форм представления дискретных данных и нетрадиционным методам вычисления. Наименее изученными остаются вопросы, связанные с выбором оптимальных форм представления дискретных данных с учетом операционных возможностей применяемого вычислительного средства.

Известные методы логических вычислений своей структурой повторяют формульное описание функции. Опишем дискретную обработку данных дискретной функцией некоторой значности k_f:

$$f(x_0, x_1, \ldots, x_{n-1}) \in N_{k_f}, \quad x_t \in N_{k_t}, \tag{П4.1}$$

зависящей от n аргументов $X = \{x_0, x_1, \ldots, x_{n-1}\}$, значности которых, в общем случае, различны и равны, соответственно, $K = \{k_0, k_1, \ldots, k_{n-1}\}$. Здесь $N_k = \{0, 1, \ldots, k-1\}$ – конечное множество произвольной природы из k элементов.

Представим дискретную функцию разложением, полученным при спектральной

$$f(X) = \sum_{i=0}^{m-1} \theta_i(X) \times a_i, \quad m = k_0 k_1 \ldots k_{n-1}, \tag{П4.2}$$

или алгебраической декомпозиции

$$f(X) = \sum_{i=0}^{k'-1} \theta_i(X') \times a_i(X''), \quad k' = k'_0 k'_1 \ldots k'_{n'-1}, \tag{П4.3}$$

выполняющихся в одной из образующих алгебр $R = \langle N_k, +, \times \rangle$, где θ_i – система спектральных функций, а a_i – коэффициенты разложения. В последнем случае множество переменных X разбивается на два непересекающихся множества $X' = \{k'_0, k'_1, \ldots, k'_{n'-1}\}$ и $X'' = \{k''_0, k''_1, \ldots, k''_{n''-1}\}$.

Для синтеза разложений (П4.2) и (П4.3) воспользуемся дискретным преобразованием

$$\begin{cases} \mathbf{A} = \mathbf{D} \times \mathbf{F}; \\ \mathbf{F} = \mu\mathbf{D} \times \mathbf{A}, \end{cases} \qquad \text{(П4.4)}$$

где $\mathbf{F}$ – вектор функции, $\mathbf{A}$ – вектор коэффициентов формы, $\mathbf{D}$ и $\mu\mathbf{D}$ – матрицы прямого и обратного преобразования размерности $k' \times k'$. Здесь знаком μ обозначена матричная операция обращения матрицы, выполняемая в одной из образующих алгебр.

Задача конструирования формы (П4.2) и (П4.3) сводится к разложению некоторой дискретной функции $f(X)$ в ряд по заданным спектральным функциям $\theta_i(X)$. Для этого вычисляется спектр $\mathbf{A}$ вектора данных $\mathbf{F}$ с использованием дискретного преобразования (П4.4) таким образом, что значения функции $f(X)$ и значения формы совпадают во всех точках области определения. Разнообразие форм для одной и той же функции определяется возможностью получения различных полных систем функций $\{\theta_i(X)\}$.

П4.1. Полиномиальные формы

Практический интерес представляют полиномиальные формы, которые имеют однородную алгебраическую структуру и хорошо реализуются микроэлектронными средствами.

П4.1.1. Полиномиальные функции

Рассмотрим обобщенную аналитическую конструкцию полиномиальных функций:

$$\theta_i(x_0, x_1, \ldots, x_{n-1}) = x_0^{i_0} \circ_0 x_1^{i_1} \circ_1 \ldots \circ_{n-2} x_{n-1}^{i_{n-1}} \circ_{n-1} c_i, \qquad \text{(П4.5)}$$

где $i = \overline{0, k'-1}$, c_i – произвольные константы; $\circ_j$ – функции двух переменных, $i = (i_0\, i_1 \ldots i_{n-1})_{k_{n-1}\ldots k_1 k_0}$ – представление числа i в системе счисления с основаниями, равными значностям переменных; $x_j^{i_j} = x_j \bullet_j i_j$, $\bullet_j$ – степенные функции, задаваемые для каждой переменной в отдельности.

С учетом $x_j^{i_j} = x_j \bullet_j i_j$ конструкцию (П4.5) можно записать так,

$$\theta_i(x_0, x_1, \ldots, x_{n-1}) = (x_0 \bullet_0 i_0) \circ_0 (x_1 \bullet_1 i_1) \circ_1 \ldots \circ_{n-2} (x_{n-1} \bullet_{n-1} i_{n-1}) \circ_{n-1} c_i, \qquad \text{(П4.6)}$$

Зададим порядок вычисления выражения слева направо при приоритетном выполнении функций $\bullet_j$ по отношению к $\circ_j$.

В свою очередь, константы c_i будем рассматривать как результат вычисления некоторой фиктивной степенной операции над фиктивной переменной.

П4.1.2. Методика синтеза

Синтез полиномиальной формы сводится к построению матрицы $\mathbf{D}$ с учетом вида аналитической конструкции спектральных функций и нахождению вектора коэффициентов $\mathbf{A}$ для каждого характеристического вектора $\mathbf{F}$.

Рассмотрим методику синтеза полиномиальных форм в произвольной образующей алгебре R.

Структура выражения (П4.5) позволяет использовать при построении $\mathbf{D}$ операцию кронекеровского произведения матриц.

Определение П4.1. *Пусть заданы двуместная операция* $\circ$ *и две матрицы:* $\mathbf{A}=[a_{i_0 j_0}]$ *размерности* $n_0 \times m_0$ *и* $\mathbf{B}=[b_{i_1 j_1}]$ *размерности* $n_1 \times m_1$.

Кронекеровским *произведением матрицы* $\mathbf{A}$ *на матрицу* $\mathbf{B}$ *относительно операции* $\circ$ *называется матрица* $\mathbf{C}=\mathbf{A}\otimes_{\circ}\mathbf{B}$ *с элементами* $c_{ij}=a_{i_0 j_0}\circ b_{i_1 j_1}$, *где* $i=(i_1\, i_0)_{n_1 n_0}$, $j=(j_1\, j_0)_{m_1 m_0}$ *– представление индексов* i *и* j *в системе счисления с различными основаниями цифр.*

Из определения П4.1 непосредственно следует, что $\mathbf{C}$ является блочной матрицей, состоящей из $n_1 \times m_1$ подматриц $\mathbf{C}_{i_1 j_1}=\mathbf{A}\circ b_{i_1 j_1}$ размерности $n_0 \times m_0$. Заметим, что кронекеровское произведение можно определить и так: $\mathbf{C}_{i_0 j_0}=a_{i_0 j_0}\circ\mathbf{B}$.

В первом случае операцию будем называть ***левой*** и обозначать $\otimes$ или $\overleftarrow{\otimes}$, во втором – ***правой*** и обозначать $\overrightarrow{\otimes}$. Очевидно $\mathbf{A}\overleftarrow{\otimes}\mathbf{B}\neq\mathbf{A}\overrightarrow{\otimes}\mathbf{B}$. Правое кронекеровское произведение еще известно как прямое (внешнее) произведение матриц.

Ядро преобразования. Зададим ***ядро*** преобразования (П4.4), которое определяет степенные и соединительные операции полиномиальной аналитической конструкции (П4.5).

Степенные операции представим матрицами $\mathbf{\Gamma}_t$ с элементами $\bullet_t(i,j)=i^j$, где $i,j=\overline{0,k_t-1}$ – номер строки и столбца $\mathbf{\Gamma}_t$, $t=\overline{0,n-1}$. Соединительные операции $\circ_t$, как и степенные, зададим в виде матриц $\mathbf{\Delta}_t$ с элементами $\circ_t(i,j)=i\circ_t j$, где $i=\overline{0,k_t-1}$ – номер строки $\mathbf{\Delta}_t$; $j=\overline{0,k_{t+1}-1}$ – номер столбца $\mathbf{\Delta}_t$, $t=\overline{0,n-1}$.

Рекуррентная процедура. С учетом структуры аналитической конструкции (П4.6) и определения П4.1 матрицу $\mathbf{D}$ получаем по рекуррентному правилу:

$$\begin{cases}\mathbf{D}_0 &=\mathbf{\Gamma}_0;\\ \mathbf{D}_{t+1}&=\mathbf{D}_t\otimes_{\circ_t}\mathbf{\Gamma}_{t+1} \quad (t=\overline{0,n-2});\\ \mathbf{D} &=\mathbf{D}_{n-1}\circ_{n-1}\mathbf{C},\end{cases} \tag{П4.7}$$

где $\mathbf{\Gamma}_t$ – матрица степенной операции $\bullet_t$; $\otimes_{\circ_t}$ – обобщенная операция кронекеровского произведения матриц, выполняющиеся относительно соединительной операции $\circ_t$; $\mathbf{C}$ – матрица констант, состоящая из k' одинаковых строк k' произвольных констант.

На каждом шаге вычисления (П4.7) имеем матрицу $\mathbf{D}_{t+1}$, состоящую из $k_t \times k_t$ подматриц $\mathbf{D}_t$, каждая из которых поэлементно преобразована операцией $\circ_t$, правым операндом которой является соответствующий элемент матрицы $\mathbf{\Gamma}_{t+1}$.

Фундаментальность и ортогональность. Достаточные условия, необходимые для получения фундаментальной матрицы в используемой образующей алгебре, реализуются выбором операций и их последовательностью. Если эти условия выполнены, то в результате вычисления (П4.7) получаем $\mathbf{D}$ – фундаментальную матрицу образующей алгебры R.

Если матрица прямого преобразования $\mathbf{D}$ окажется ортогональной, то может быть получена матрица обратного преобразования $\mathbf{Q}$, вычисляемая из условия ортогональности $\mathbf{D} \times \mathbf{Q} = \mathbf{E}$ обращением $\mathbf{D}$, $\mathbf{Q} = \mu_R \mathbf{D}$, где μ_R – унарная матричная операция обращения матриц в R, $\mathbf{E}$ – единичная матрица.

Пример П4.1. Для демонстрации методики синтеза полиномиальных форм рассмотрим функцию, заданную следующей таблицей.

Таблица П4.1 Дискретная функция 1

x_0	x_1	$f(x_0, x_1)$
0	0	2
1	0	1
0	1	3
1	1	3
0	2	0
1	2	1

Из табл. П4.1 видно, функция имеет значность $k_f = 4$, а переменные – значности из $\mathrm{K} = \{3, 2\}$. Найдем спектральное разложение функции в мультипликативной алгебре с операциями:

$$+ = \begin{bmatrix} 0 & 1 & 2 & 3 \\ 1 & * & * & * \\ 2 & * & * & * \\ 3 & * & * & * \end{bmatrix}, \quad \times = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 2 & 3 & 1 \\ 0 & 3 & 1 & 2 \\ 0 & 1 & 2 & 3 \end{bmatrix},$$

откуда видно, что нуль алгебры $\sigma = 0$, а единица – $\tau = 3$.

В соответствии с (П4.2) и (П4.5), искомая полиномиальная форма имеет вид:

$$f(X)=\sum_{i=0}^{5}(x_0^{i_0}\circ_0 x_1^{i_1}\circ_1 c_i)\times a_i=\sum_{i=0}^{5}(x_0\bullet_0 i_0)\circ_0(x_1\bullet_1 i_1)\times a_i\,,$$

где операция $\circ_1$ – тривиальна и может быть опущена, а оставшиеся операции определим так:

$$\bullet_0=\begin{bmatrix}2&3&2\\2&2&0\\1&2&2\end{bmatrix},\quad \circ_0=\begin{bmatrix}3&0&1&2\\2&0&1&1\\0&0&0&0\\3&0&2&2\end{bmatrix},\quad \bullet_1=\begin{bmatrix}1&0\\3&1\end{bmatrix}.$$

Из (П4.7) получаем

$$\mathbf{D}=\mathbf{\Gamma}_0\otimes_0\mathbf{\Gamma}_1=\left[\begin{array}{ccc|ccc}0&0&0&0&3&0\\0&0&0&0&0&3\\0&0&0&2&0&0\\\hline0&2&0&0&0&0\\0&0&2&0&0&0\\1&0&0&0&0&0\end{array}\right].$$

Матрица прямого преобразования $\mathbf{D}$ является ортогональной и может быть использована в мультипликативной алгебре для нахождения матрицы обратного преобразования $\mathbf{Q}$,

$$\mathbf{Q}=\widetilde{\mathbf{D}}^{T}=\left[\begin{array}{ccc|ccc}0&0&0&0&0&3\\0&0&0&2&0&0\\0&0&0&0&2&0\\\hline0&0&2&0&0&0\\1&0&0&0&0&0\\0&1&0&0&0&0\end{array}\right],$$

где $\widetilde{\mathbf{D}}$ – матрица, полученная из $\mathbf{D}$ заменой ненулевых элементов на обратные, а T – операция транспонирования матриц.

Далее находим вектор коэффициентов формы $\mathbf{A}$, $\mathbf{A}=\mathbf{Q}\times\mathbf{F}=[1\,2\,0\,2\,3\,2]^T$, куда из табл. П4.1 подставлен вектор функции $\mathbf{F}=[2\,1\,3\,3\,0\,1]^T$. Вектор $\mathbf{A}$ использует для конструирования искомой полиномиальной формы функции:

$$f(X)=(x_0^0\circ_0 x_1^0)\times 1+(x_0^1\circ_0 x_1^0)\times 2+(x_0^1\circ_0 x_1^1)\times 2+(x_0^0\circ_0 x_1^2)+(x_0^1\circ_0 x_1^2)\times 2\,.$$

П4.1.3. Форма Рида-Маллера

Некоторым обобщением полиномиального представления является представление функции в форме Рида-Маллера, у которой спектральные функции

$$\theta_i(x_0,x_1,\ldots,x_{n-1})=(\neg_0 x_0^{i_0})\circ_0(\neg_1 x_1^{i_1})\circ_1\ldots\circ_{n-2}(\neg_{n-1}x_{n-1}^{i_{n-1}})\circ_{n-1}c_i\,,\qquad(\text{П4.8})$$

где $\neg_j$ – унарные операции, задающие полярность вхождения переменных в аналитическую конструкцию формы, $j = \overline{0, n-1}$.

По своей сути, операции ***полярности*** определяют предварительное преобразование переменных, которое осуществляется перед вычислением степенных операций.

Заметим, что операции $\neg_j$ разбивают все функции на классы эквивалентности. Это позволяет синтезировать полиномиальные формы с точностью до взаимно однозначного их преобразования. Если найдено компактное (эффективное) представление f, то такое же представление будут иметь $k_0!k_1!\ldots k_{n-1}!$ функций, получаемых из f после всевозможных преобразований переменных операциями полярности.

Пример П4.2. Представим функцию, заданную в табл. П4.1, в обобщенной форме Рида-Маллера. Будем использовать ее полиномиальное представление из примера на с. 414.

Зададим полярность вхождения переменных векторами

$$\neg_0 = \begin{bmatrix} 2 \\ 0 \\ 1 \end{bmatrix}, \quad \neg_1 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

и найдем модифицированные матрицы степенных операций путем перестановки и копирования строк в соответствии с полярностью переменных:

$$\mathbf{\Gamma}_0^{\neg} = \begin{bmatrix} 1 & 2 & 2 \\ 2 & 3 & 2 \\ 2 & 2 & 0 \end{bmatrix}, \quad \mathbf{\Gamma}_1^{\neg} = \begin{bmatrix} 3 & 1 \\ 1 & 0 \end{bmatrix}.$$

Далее получим матрицы прямого и обратного преобразования:

$$\mathbf{D} = \mathbf{\Gamma}_0^{\neg} \otimes_0 \mathbf{\Gamma}_1^{\neg} = \left[\begin{array}{ccc|ccc} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 0 & 0 & 3 \end{array}\right], \quad \mathbf{Q} = \widetilde{\mathbf{D}}^T = \left[\begin{array}{ccc|ccc} 2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 0 & 0 & 3 \end{array}\right].$$

Затем находим вектор коэффициентов формы $\mathbf{A} = \mathbf{Q} \times \mathbf{F} = [1\,2\,1\,1\,0\,1]^T$ и конструируем итоговое выражение:

$$\begin{aligned} f(X) &= (\neg_0 x_0^0 \circ_0 \neg_1 x_1^0) \times 1 + (\neg_0 x_0^1 \circ_0 \neg_1 x_1^0) \times 2 + (\neg_0 x_0^0 \circ_0 \neg_1 x_1^1) \times 1 + \\ &+ (\neg_0 x_0^1 \circ_0 \neg_1 x_1^1) \times 1 + (\neg_0 x_0^1 \circ_0 \neg_1 x_1^2) \times 1. \end{aligned}$$

П4.1.4. Мультипликативные формы

Формирование базиса дискретного преобразования (П4.4) сопряжено со значительными вычислительными трудностями, которые связаны с обращением матриц большой размерности.

В алгебре логики и мультипликативной алгебре нахождение матрицы обратного преобразования сводится к операции транспонирования, в то время как в аддитивной алгебре и конечном поле (целостном кольце) требуется традиционное обращение матриц. Следовательно, основную трудность при синтезе полиномиальных форм в аддитивной и фундаментальной алгебрах вызывает обращение матриц больших размерностей.

Рассмотрим частный случай полиномиального представления – мультипликативную форму, позволяющую решить описанную проблему. Для этого дадим ее определение.

Определение П4.2. *Мультипликативными будем называть полиномиальные формы, у которых в выражении (П4.6) все или часть операций* $\circ_j$ *(*$j = \overline{0, n-1}$*) совпадают с операцией умножения образующей алгебры* R *в области значений степенных операций* $\bullet_j$ *и* $\bullet_{j+1}$ *(такие соединительные операции* $\circ_j$ *также будем также называть мультипликативными).*

На практике в качестве мультипликативных операций в булевой алгебре используется конъюнкция, а в многозначной логике – арифметическое умножение. Синтез мультипликативных форм осуществляется на основе следующей теоремы.

Теорема П4.1. *Пусть* $\circ_j$ *(*$j = \overline{0, n-1}$*) – мультипликативные операции образующей алгебры* R *и для каждой фундаментальной матрицы* $\mathbf{\Gamma}_j$ *в* R *существует обратная матрица* $\mu\mathbf{\Gamma}_j$*, тогда*

$$\mu\left(\bigotimes_{j=0}^{n-1}\mathbf{\Gamma}_j\right) = \bigotimes_{j=0}^{n-1}\mu\mathbf{\Gamma}_j. \qquad \text{(П4.9)}$$

Пусть имеется возможность обращать матрицы до некоторого размера m'. Будем использовать чередование в аналитической конструкции полиномиальной формы мультипликативных и немультипликативных операций,

$$\theta_i(x) = \theta_i^{(0)}(x_0,\ldots,x_{t_0-1})\times\theta_i^{(t_0)}(x_{t_0},\ldots,x_{t_1-1})\times\ldots\times\theta_i^{(t_{r-1})}(x_{t_{r-1}},\ldots,x_{n-1})\circ_{n-1} c_i, \qquad \text{(П4.10)}$$

где $\theta_i^{(t)} = x_t^{i_t}\circ_t x_{t+1}^{i_{t+1}}\circ_{t+1}\ldots\circ_{t+s-1} x_{t+s}^{i_{t+s}}$, причем таким образом, чтобы подряд идущих немультипликативных операций, начиная с $\circ_t$ было быть не более s, где s находим из неравенства $m' \geq k_t k_{t+1}\ldots k_{t+s}$.

Тогда при синтезе мультипликативной форму сначала получаем матрицы $\mathbf{\Gamma}_t$ ($t=\overline{0,r-1}$), соответствующие функциям $\theta_i^{(t)}$ из (П4.10), которые, в общем случае, могут и не иметь полиномиального представления.

Затем, после обращения $\mathbf{\Gamma}_t$, находим матрицу обратного преобразования $\mathbf{Q}$ в виде кронекеровского произведения матриц $\mu\mathbf{\Gamma}_t$. Очевидно, что при вычислениях θ_i необходимо учитывать новый порядок выполнения операций.

В итоге, рекуррентное правило для формирования базиса дискретного преобразования (П4.7) может быть представлено в следующем виде:

$$\begin{cases} \mathbf{D}_0=\mathbf{\Gamma}_0, & \mathbf{D}_t=\mathbf{\Gamma}_{t-1}\otimes\mathbf{D}_{t-1} \quad (t=\overline{0,r-1}), & \mathbf{D}=\mathbf{D}_{n-2}; \\ \mathbf{Q}_0=\mu\mathbf{\Gamma}_0, & \mathbf{Q}_t=\mu\mathbf{\Gamma}_{t-1}\otimes\mathbf{Q}_{t-1} \quad (t=\overline{0,r-1}), & \mathbf{Q}=\mathbf{Q}_{n-2}. \end{cases} \qquad (\text{П4.11})$$

Пример П4.3. Представим функцию из Табл. П4.1 в мультипликативной форме. Разложение выполним в аддитивной алгебре, образованной поразрядными операциями неэквиваленции и конъюнкции:

$$+=\begin{bmatrix} 0 & 1 & 2 & 3 \\ 1 & 0 & 3 & 2 \\ 2 & 3 & 0 & 1 \\ 3 & 2 & 1 & 0 \end{bmatrix}, \quad \times=\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 2 & 2 \\ 0 & 1 & 2 & 3 \end{bmatrix},$$

с нулем $\sigma=0$ и единицей $\tau=3$.

В соответствии с (П4.5), искомая полиномиальная форма имеет вид:

$$f(X)=\sum_{i=0}^{5}(x_0^{i_0}\circ_0 x_1^{i_1}\circ_1 c_i)\times a_i\,,$$

где степенные операции определим на основе поразрядной дизъюнкции и сложения по модулю 2, а соединительную операцию – как поразрядную конъюнкцию:

$$\mathbf{\Gamma}_0=\begin{bmatrix} 0 & 3 \\ 3 & 3 \end{bmatrix}, \quad \mathbf{\Delta}_0=\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 2 & 2 \\ 0 & 1 & 2 & 3 \end{bmatrix}, \quad \mathbf{\Gamma}_1=\begin{bmatrix} 0 & 3 & 0 \\ 3 & 0 & 3 \\ 0 & 3 & 3 \end{bmatrix}.$$

В области значений степенных операций матрица $\mathbf{\Delta}_0$ совпадает с алгебраическим умножением, следовательно, операция $\circ_1$ – мультипликативная. Так как $\mathbf{\Gamma}_0$ и $\mathbf{\Gamma}_1$ обращаемые матрицы в заданной алгебре,

$$\mu\mathbf{\Gamma}_0=\begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}, \quad \mu\mathbf{\Gamma}_1=\begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 0 \\ 1 & 0 & 1 \end{bmatrix},$$

с $\Delta_0=1$ и $\Delta_1=1$, то в соответствии с (П4.11), найдем матрицу $\mathbf{Q}$:

$$\mathbf{Q}=\begin{bmatrix}1&1\\1&0\end{bmatrix}\otimes\begin{bmatrix}1&1&1\\1&0&0\\1&0&1\end{bmatrix}=\left[\begin{array}{ccc:ccc}1&1&1&1&1&1\\1&0&0&1&0&0\\1&0&1&1&0&1\\\hdashline 1&1&1&0&0&0\\1&0&0&0&0&0\\1&0&1&0&0&0\end{array}\right],\quad \Delta=\Delta_0\Delta_1=1.$$

Получаем вектор коэффициентов $\Delta\cdot\mathbf{A}=\mathbf{Q}\cdot\mathbf{F}=[1\,1\,3\,0\,2\,1]^T$, где точкой обозначена операция циклической суммы, а $\mathbf{F}$ – характеристический вектор функции, $\mathbf{F}=[2\,1\,3\,3\,0\,1]^T$. По вектору $\mathbf{A}$ конструируем искомую мультипликативную форму:

$$f(X)=(x_0^0\circ_0 x_1^0)\times 1+(x_0^1\circ_0 x_1^0)\times 1+(x_0^0\circ_0 x_1^1)+(x_0^0\circ_0 x_1^2)\times 2+(x_0^1\circ_0 x_1^2)\times 1.\ \blacklozenge$$

П4.1.5. Булева мультипликативная форма

Представление логических функций в булевом мультипликативном базисе основано на использовании в качестве логических операций булевой конъюнкции &, совпадающей с операцией арифметического умножения на множестве $B=\{0,1\}$:

$$f(X)=\sum_{i=0}^{m-1}(x_0^{i_0}\,\&\,x_1^{i_1}\,\&\ldots\&\,x_{n-1}^{i_{n-1}})\times a_i\ . \qquad (\text{П4.12})$$

Для согласования области определения булевой конъюнкции с областями значений степенных операций, последние необходимо задавать в виде булевых матриц, т.е. матриц, элементы которых принимают значения на B.

Пример П4.4. Представим функцию из табл. П4.1 в булевой форме. Разложение выполним в алгебре логики, образованной операциями

$$+=\begin{bmatrix}0&1&2&3\\1&1&3&3\\2&3&2&3\\3&3&3&3\end{bmatrix},\quad \times=\begin{bmatrix}0&0&0&0\\0&1&2&3\\*&*&*&*\\*&*&*&*\end{bmatrix},$$

с нулем $\sigma=0$ и единицей $\tau=1$, где сложение – поразрядная дизъюнкция, а умножение задано так, что вычисление формы сводится к суммированию коэффициентов, для которых логическая часть не равна нулю.

Для существования разложения степенные операции должны выражаться матрицами перестановок:

$$\mathbf{\Gamma}_0=\begin{bmatrix}0&1\\1&0\end{bmatrix},\quad \mathbf{\Delta}_0=\begin{bmatrix}0&0\\0&1\end{bmatrix},\quad \mathbf{\Gamma}_1=\begin{bmatrix}1&0&0\\0&1&0\\0&0&1\end{bmatrix},$$

где первая степенная операция – неэквиваленция, а вторая – литеральная операция

$$x_1^j = \begin{cases} 0, & \text{при } x_1 \neq j; \\ 1, & \text{при } x_1 = j. \end{cases}$$

Обратим степенные операции путем транспонирования матриц, а затем вычислим матрицу $\mathbf{Q}$ и вектор коэффициентов $\mathbf{A}$:

$$\mathbf{Q} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \otimes_{\&} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \left[\begin{array}{ccc:ccc} 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ \hdashline 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{array}\right], \quad \mathbf{A} = \mathbf{Q} \times \mathbf{F} = \begin{bmatrix} 3 \\ 0 \\ 1 \\ 2 \\ 1 \\ 3 \end{bmatrix}.$$

В итоге имеем

$$f(X) = (x_0 \,\&\, x_1^0) \times 3 + (x_0 \,\&\, x_1^1) + (\bar{x}_0 \,\&\, x_1^1) \times 2 + (x_0 \,\&\, x_1^2) + (\bar{x}_0 \,\&\, x_1^2) \times 3,$$

где $\bar{x}_0$ – отрицание x_0. ♦

Минимизация булевой мультипликативной формы осуществляется путем подбора степенных операций для каждой из переменных. Заметим, что количество обращаемых булевых матриц размерности $k_t \times k_t$ может быть подсчитано по формуле:

$$N_B(k_t) = k_t! \left(2^{\frac{k_t(k_t-1)}{2}+1} - 1 \right),$$

где $k_t!$ – факториал числа k_t.

Использование булевой мультипликативной формы позволяет эффективно реализовать вычисления в многозначной логике на вычислительных средствах, использующих двоичное кодирование данных.

П4.1.6. Мультипликативная форма Уолша

Пусть задана образующая алгебра с арифметической операцией умножения. Определим степенные операции мультипликативной формы в виде матриц, состоящих из элементов множества $W = \{1, -1\}$. В результате получим расширение полиномиального представления Уолша:

$$f(X) = \sum_{i=0}^{m-1} x_0^{i_0} \times x_1^{i_1} \times \ldots \times x_{n-1}^{i_{n-1}} \times a_i = \sum_{i=0}^{m-1} a_i (-1)^{b_i(X)}, \qquad (\text{П4.13})$$

где $b_i(X)$ – количество отрицательных значений степенных операций для каждого индекса i на заданном наборе данных X. Вычисление полинома (П4.13) сводится к подсчету $b_i(X)$ и суммированию коэффициентов со знаками, равными знакам выражений $(-1)^{b_i(X)}$.

Для уменьшения времени вычисления полиномиальных функций определим степенные операции так, как это делается в классической полиномиальной форме Уолша:

$$x_t^{i_t} = (-1)^{(x_t \times i_t) \bmod k_t},$$

откуда находим, что форма (П4.13) существует только при $k_t \in \{2, 3\}$, что вызвано линейной зависимостью строк (столбцов) матриц степенных операций. При $k_t = 2$ получаем классический базис Уолша.

Задавая степенные операции в виде

$$x_t^{i_t} = (-1)^{(x_t + i_t) \bmod k_t},$$

получаем аналитическую конструкцию, существующую при всех простых k_t, и в которой в качестве операции арифметического умножения может быть использована более быстрая операция арифметического сложения:

$$f(X) = \sum_{i=0}^{m-1} a_i \times (-1)^{b(X)}, \quad b(X) = \sum_{t=0}^{n-1} (x_t + i_t)_{\bmod k_t}. \qquad (П4.14)$$

К достоинствам формы (П4.14) можно отнести регулярность структуры аналитической конструкции и небольшое время вычисления. С другой стороны, обратные матрицы степенных операций легко находятся по исходным матрицам дискретного преобразования и имеют достаточно простой вид:

$$\mathbf{\Gamma}_t = \begin{bmatrix} 1 & -1 & 1 & -1 & . & 1 \\ -1 & 1 & -1 & . & 1 & . \\ 1 & -1 & . & 1 & . & 1 \\ -1 & . & 1 & . & 1 & -1 \\ . & 1 & . & 1 & -1 & 1 \\ 1 & . & 1 & -1 & 1 & -1 \end{bmatrix}, \quad \mu\mathbf{\Gamma}_t = \frac{1}{2^{k_t - 2}} \begin{bmatrix} 1 & 0 & 0 & . & 0 & 1 \\ 0 & 0 & . & 0 & 1 & 1 \\ 0 & . & 0 & 1 & 1 & 0 \\ . & 0 & 1 & 1 & 0 & . \\ 0 & 1 & 1 & 0 & . & 0 \\ 1 & 1 & 0 & . & 0 & 0 \end{bmatrix}.$$

Минимизация мультипликативной формы Уолша осуществляется путем подбора степенных операций для каждой из переменных. Количество обращаемых матриц $N_w(k_t)$ размерности $k_t \times k_t$, полученное в результате вычислительного эксперимента, представлено в таблице (там же для сравнения приведены значения для N_B):

Таблица П4.2 Сравнение количества обращаемых матриц

k_t	2	3	4
$N_w(k_t)$	8	192	22272
$N_B(k_t)$	6	90	3048

Из табл. П4.2 видно, что возможности минимизации логической функции в мультипликативной форме Уолша превосходят аналогичные возможности в булевой форме.

П4.1.7. Другие частные случаи

Использование значений степенных операций из множества $\{-1, 0, 1\}$ позволяет обобщить рассмотренные ранее мультипликативные формы. Разложение произвольной функции в этом случае производится по трехуровневым полиномиальным функциям. Количество степенных операций для такой формы превосходит количество степенных операций для мультипликативной формы Уолша и булевой мультипликативной формы. Понятно, что матрицами степенных операций в этом случае могут быть все $N_B(k_t)$ булевы матрицы и все $N_W(k_t)$ матрицы Уолша, а также все обращаемые матрицы, получаемые заменой произвольного числа единиц в матрицах Уолша на ноль. Это определяет широкие возможности минимизации в классе трехуровневых мультипликативных форм.

С целью повышения эффективности вычислений нами наложены дополнительные ограничения на выбор базиса, которые возможны по причине определенной произвольности в выборе операций аналитической конструкции полиномиальной формы.

Среди таких ограничений – возможность факторизации базиса, т.е. представления матриц прямого (обратного) преобразования в виде произведения слабо заполненных матриц, что позволяет построить быстрый алгоритм дискретного преобразования.

Иногда важным является уменьшение сложности формирования базиса, поскольку это отражается на программных и аппаратных затратах. Варьируя базис при неизменных входных данных можно получить множество спектров, некоторые из которых являются более предпочтительными по условию решаемой задачи, или содержа много нулевых компонентов. В последнем случае решается задача минимизации функции в заданном классе операций.

Можно также ввести ограничения на выбор базиса, связанные с минимизацией коэффициентов формы, что приводит к уменьшению памяти, необходимой для их хранения, или включить в конструкцию те операции, которые реализуются на используемом вычислительном средстве с наибольшей эффективностью.

На практике получили развитие методы вычисления с помощью арифметических полиномов в булевой алгебре. Выражение (П4.5) дает возможность свести вычисления в многозначной логике к вычислению арифметико-логического полинома в булевой алгебре путем соответствующего выбора степенных функций. В этом случае ядро дискретного преобразования задается в виде булевых матриц, а логические вычисления сводятся к выполнению булевых операций над совокупностью промежуточных переменных, полученных в результате возведения в степень входных переменных.

Существует несколько других частных случаев мультипликативных форм, при использовании которых повышается эффективность вычисления функций.

Например, если последняя соединительная операция формы принимает значения на двоичном множестве, то операция алгебраического умножения не выполняется, а вычисление функции сводится к суммированию коэффициентов a_i, у которых θ_i не равны нулю.

В случае, если степенные операции принимают значения на двоичном множестве, т.е. преобразуют многозначные значения переменных в двоичные, последующие вычисления спектральных функций можно производить в булевой алгебре.

При $k > k_f$ увеличивается количество операций, которые можно использовать при синтезе. При прочих равных условиях это позволяет получить более компактное представление функции.

И, наконец, количество операций, необходимых для вычисления полиномиальных функций, можно уменьшить, задав все или часть степенных операций $\bullet_j$ так, чтобы существовали такие y_j, для которых $x_j \bullet_j y_j = x_j$. Тогда при $i_j = y_j$ вычисление операции $\bullet_j$ не выполняются, так как их результат равен значению переменной.

П4.2. Неполиномиальные формы

В настоящее время разработано большое количество алгоритмов быстрых преобразований и сверток. Большинство этих разработок преследует одну цель: увеличить скорость выполнения ортогональных преобразований, в частности, имеются работы по теоретико-числовым преобразованиям, алгоритмам вычисления свертки и дискретного преобразования Кули-Тьюки и Винограда, полиномиальным преобразованиям Нуссбаумера.

Много работ посвящено разработке быстрых алгоритмов дискретных ортогональных преобразований на циклических группах и над конечными алгебраическими структурами. Вместе с тем, развивается специфическое направление, связанное с построением дискретных преобразований, заданных на конечных как абелевых, так и неабелевых группах и определенных над конечными кольцами (полями).

На практике используются два класса дискретных преобразований: полиномиальные и неполиномиальные. Примером полиномиального базиса может служить спектральный базис Уолша. Классическим примером неполиномиального базиса является базис Хаара, функции которого не имеют полиномиального представления. Однако благодаря низкой вычислительной сложности последние получили широкое распространение в области распознавания образов, обработки изображений, в коммуникационных технологиях для кодирования данных, коммутации и цифровой фильтрации.

П4.2.1. Неполиномиальная аналитическая конструкция

Пусть спектральные функции имеют аналитическую конструкцию

$$\theta_i(x_0, x_1, \ldots, x_{n-1}) = (\neg_0^{(i)} x_0) \circ_0^{(i)} (\neg_1^{(i)} x_1) \circ_1^{(i)} \ldots \circ_{t-2}^{(i)} (\neg_{n-1}^{(i)} x_{n-1}) \circ_{n-1}^{(i)} c^{(i)}, \qquad (П4.15)$$

где $i = \overline{0, k'-1}$, $j = \overline{0, n-1}$, $c^{(i)}$ – произвольные константы; $\circ_j^{(i)}$ ($\neg_j^{(i)}$) – система бинарных (унарных) операций.

В отличие от полиномиальной формы, операции в (П4.15) зависят от индекса функции, т.е. каждая спектральная функция θ_i имеет свой собственный набор операций. Такие функции будем называть ***неполиномиальными***. Очевидно, что полиномиальные формы являются частным случаем неполиномиальных.

Заметим, что регулярность аналитической конструкции у неполиномиальными форм меньше, чем у полиномиальных.

П4.2.2. Методика синтеза

Синтез неполиномиальных форм осуществим путем получения всевозможных спектральных функций, которые в алгебре образующих операций могут быть столбцами матрицы прямого преобразования $\mathbf{D}$. Далее из них сформируем невырожденную матрицу $\mathbf{D}$ в соответствии с условиями теоремы разложения. По завершению конструирования $\mathbf{D}$ найдем обратную матрицу $\mathbf{Q}$, если она существует.

В главе 5 на примере мультипликативной и аддитивной алгебры показано, что могут существовать представления дискретных функций, для которых матрица обратного преобразования не существует. Методика синтеза таких форм определяется свойствами образующей алгебры. В этом случае при оперировании дискретным преобразованием матричные операции не используются, а все преобразования выполняются над исходной системой алгебраических уравнений путем ее тождественного преобразования в образующей алгебре, например, как это осуществлялось для разложений в мультипликативной алгебре.

Пример П4.5. Синтезируем неполиномиальную форму для функции из табл. П4.3. Из (П4.2) и (П4.15) получаем:

$$f(x_0, x_1) = \sum_{i=0}^{k_0 k_1 - 1} (\neg_0^{(i)} x_0 \circ_0^{(i)} \neg_1^{(i)} x_1) \times a_i .$$

Разложение осуществим в конечном поле $R_F = \langle N_3, +, \times \rangle$ с операциями сложения и умножения по модулю 3:

$$+ = \begin{bmatrix} 0 & 1 & 2 \\ 1 & 2 & 0 \\ 2 & 3 & 0 \end{bmatrix}, \quad \times = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 2 \\ 0 & 2 & 1 \end{bmatrix}.$$

Нулем алгебры является элемент $\sigma = 0$, а единицей – $\tau = 1$.

Таблица П4.3. Дискретная функция 2

x	x_0	x_1	f
0	0	0	1
1	1	0	0
2	2	0	1
3	0	1	2
4	1	1	2
5	2	1	0

Пусть $\boldsymbol{\theta} = [\theta_i]$ – вектор функций, зависящих от переменных x_0 значности $k_0 = 3$ и x_1 значности $k_1 = 2$. Зададим конкретный вид этих функций в виде формул,

$$\boldsymbol{\theta} = [1 \;\; x_0 \;\; x_1 \;\; \widetilde{x}_0 \;\; x_1 \triangleright x_0 \;\; x_0 \triangleleft x_1],$$

где используемые операции определены так:

$$\sim = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}, \quad \triangleright = \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \end{bmatrix}, \quad \triangleright = \begin{bmatrix} 0 & 0 \\ 0 & 1 \\ 0 & 2 \end{bmatrix}.$$

Располагая вектора функций, полученные вычислением приведенных выше формул, в порядке их перечисления в векторе $\boldsymbol{\theta}$, конструируем матрицу прямого преобразования $\mathbf{D}$ и находим матрицу обратного преобразования $\mathbf{Q} = \mathbf{D}^{-1}$ в поле R_F:

$$\mathbf{D} = \begin{array}{c|c|c} x_0 & x_1 & \theta_0 \;\; \theta_1 \;\; \theta_2 \;\; \theta_3 \;\; \theta_4 \;\; \theta_5 \\ \hline 0 & 0 & 1 \;\; 0 \;\; 0 \;\; 0 \;\; 1 \;\; 0 \\ 1 & 0 & 1 \;\; 1 \;\; 0 \;\; 0 \;\; 0 \;\; 0 \\ 2 & 0 & 1 \;\; 2 \;\; 0 \;\; 1 \;\; 0 \;\; 0 \\ 0 & 1 & 1 \;\; 0 \;\; 1 \;\; 0 \;\; 2 \;\; 1 \\ 1 & 1 & 1 \;\; 1 \;\; 1 \;\; 0 \;\; 1 \;\; 2 \\ 2 & 1 & 1 \;\; 2 \;\; 1 \;\; 1 \;\; 0 \;\; 0 \end{array}, \quad \mathbf{Q} = \begin{array}{c|c|c} x_0 & x_1 & \vartheta_0 \;\; \vartheta_1 \;\; \vartheta_2 \;\; \vartheta_3 \;\; \vartheta_4 \;\; \vartheta_5 \\ \hline 0 & 0 & 0 \;\; 2 \;\; 2 \;\; 1 \;\; 1 \;\; 1 \\ 1 & 0 & 0 \;\; 2 \;\; 1 \;\; 2 \;\; 2 \;\; 2 \\ 2 & 0 & 0 \;\; 0 \;\; 2 \;\; 0 \;\; 0 \;\; 1 \\ 0 & 1 & 0 \;\; 0 \;\; 0 \;\; 1 \;\; 1 \;\; 1 \\ 1 & 1 & 1 \;\; 1 \;\; 1 \;\; 2 \;\; 2 \;\; 2 \\ 2 & 1 & 1 \;\; 2 \;\; 0 \;\; 2 \;\; 1 \;\; 0 \end{array},$$

где $\vartheta = [\vartheta_i]$ – система функций, образующих матрицу обратного преобразования $\mathbf{Q}$.

Далее, находим вектор коэффициентов $\mathbf{A} = \mathbf{Q} \times \mathbf{F}$, $\mathbf{A} = [0\,0\,2\,1\,1\,1]$ и синтезируем формулу для заданной функции,

$$f(x_0, x_1) = 2x_1 + \widetilde{x}_0 + x_1 \triangleright x_0 + x_0 \triangleleft x_1 \,. \;\blacklozenge$$

П4.2.3. Консонансное представление

Рассмотрим синтез некоторой частной аналитической конструкции неполиномиальной формы, которую назовем ***консонансной***. Из содержательных соображений потребуем присутствия в разложении функции, выполненной в рамках консонансной аналити-

ческой конструкции, спектральных функций со всеми возможными частотами, т.е. функций, которые изменяются с различной интенсивностью в области своего определения, возможно не совпадающих с областью определения разлагаемой функции. Использование консонансных конструкций позволит адекватным образом передать слитное, согласованное и одновременное звучание различных тонов в спектре сигнала.

Для решения проблемы обращения больших матриц, будем использовать мультипликативное разбиение аналитической конструкции (с. 417). Мультипликативное разбиение заключается в таком чередовании в аналитической конструкции формы мультипликативных и немультипликативных операций, при котором матрицы дискретного преобразования представляются в виде произведения матриц меньшей размерности, т.е. допускают свою факторизацию.

Синтез аналитической конструкции. Разложим дискретную функцию f в образующей алгебре R по последним $s+1$ переменным,

$$f(x_0,\ldots,x_{n-1}) = \sum_{i=0}^{k_t k_t \ldots k_{t+s-1}-1} \varphi_i^{(t)}(x_t,\ldots,x_{t+s}) \times a_i^{(t)}(x_0,\ldots,x_{t-1}), \qquad (\text{П4.16})$$

или, положив в (П4.16) $x_t^{[s]} = \{x_t, x_{t+1}, \ldots, x_{t+s-1}\}$ и $k_t^{[s]} = k_t k_{t+1} \ldots k_{t+s-1}$,

$$f(x_0^{[n]}) = \sum_{i=0}^{k_t^{[s+1]}-1} \varphi_i^{(t)}(x_t^{[s+1]}) \times a_i^{(t)}(x_0^{[t]}), \qquad (\text{П4.17})$$

где $t = n - s - 1$.

Сокращение $x_t^{[s]}$ используется для обозначения подмножества переменных X, состоящего из s переменных, начиная с переменной x_t в порядке их нумерации:

$$x_t^{[0]} = \varnothing, \quad x_t^{[1]} = \{x_t\}, \quad x_t^{[2]} = \{x_t, x_{t+1}\}, \quad \ldots$$

В свою очередь, выражение $k_t^{[s]}$ обозначает произведение значностей s переменных, начиная со значности переменной x_t:

$$k_t^{[0]} = 1, \quad k_t^{[1]} = k_t, \quad k_t^{[2]} = k_t k_{t+1}, \quad \ldots$$

Для обеспечения наличия спектральных функций с различными частотами положим для всех значений y и z переменной x_{t+s}, $y, z \in N_{k_{t+s}}$,

$$\varphi_i^{(t)}(x_t^{[s]}, y) = \varphi_i^{(t)}(x_t^{[s]}, z), \qquad (\text{П4.18})$$

где $i = \overline{0, k_t^{[s]} - 1}$.

Равенство (П4.18) задает ограничение на выбор спектральных функций $\varphi_i^{(t)}$ и утверждает, что первые $k_t^{[s]}$ из $k_t^{[s+1]}$ спектральных функций не зависят от переменной x_{t+s}.

Это возможно во всех алгебрах, кроме мультипликативной и алгебры логики. Тогда, после перегруппировки слагаемых в (П4.17), получим

$$f(x_0^{[n]}) = \sum_{i=0}^{k_t^{[s]}-1} \varphi_i^{(t)}(x_t^{[s]}, x_{t+s}) \times a_i^{(t)}(x_0^{[t]}) + \sum_{i=k_t^{[s]}}^{k_t^{[s+1]}-1} \varphi_i^{(t)}(x_t^{[s+1]}) \times a_i^{(t)}(x_0^{[t]}). \qquad (П4.19)$$

В связи с тем, что спектральные функции $\varphi_i^{(t)}(x_t^{[s]}, x_{t+s})$ при $i = \overline{0, k_t^{[s]} - 1}$ не зависят от переменной x_{t+s}, имеем

$$f(x_0^{[n]}) = g_t(x_0^{[t+s]}) + \sum_{i=k_t^{[s]}}^{k_t^{[s+1]}-1} \varphi_i^{(t)}(x_t^{[s+1]}) \times a_i^{(t)}(x_0^{[t]}), \qquad (П4.20)$$

где g_t – некоторая функция от $t+s$ переменных. Напомним, что пока подразумевается $t = n - s - 1$. Следовательно, $g_t(x_0^{[t+s]}) = g_t(x_0^{[n-1]})$, т.к. в (П4.19) первая часть спектральной суммы не зависит от последней переменной x_{n-1}.

Теперь выполним разложение g_t по последним $s+1$, как это было сделано в (П4.17)-(П4.20) при $t = \overline{n-s-2, 0}$. Объединив полученные выражения и проведя очевидные тождественные преобразования, получим:

$$f(x_0^{[n]}) = \sum_{i=0}^{k_0^{[s]}-1} \phi_i(x_0^{[s]}) \times a_i + \sum_{t=0}^{n-s-1} \sum_{i=k_t^{[s]}}^{k_t^{[s+1]}-1} \varphi_i^{(t)}(x_t^{[s+1]}) \times a_i^{(t)}(x_0^{[t]}), \qquad (П4.21)$$

где ϕ_i и $\varphi_i^{(t)}$ $(t = \overline{0, n-s-1})$ – некоторая система спектральных функций.

Так как в (П4.21) коэффициенты $a_i^{(t)}$ являются функциями от $x_0^{[t]}$ переменных, представим их в той же образующей алгебре, но уже по системе ортогональных функций

$$\psi_{ij}^{(t)} \ (, j = \overline{0, k_0^{[t]} - 1}),$$

$$f(x_0^{[n]}) = \sum_{i=0}^{k_0^{[s]}-1} \phi_i(x_0^{[s]}) \times a_i + \sum_{t=0}^{n-s-1} \sum_{i=k_t^{[s]}}^{k_t^{[s+1]}-1} \varphi_i^{(t)}(x_t^{[s+1]}) \times \left\{ \sum_{j=0}^{k_0^{[t]}-1} \psi_{ij}^{(t)}(x_0^{[t]}) \times a_{ij}^{(t)} \right\}, \qquad (П4.22)$$

где a_i, $a_{ij}^{(t)}$ – коэффициенты разложения (константы). Для определенности положим $\psi_{ij}^{(0)} = \tau$, где τ – единица образующей алгебры.

В итоге получили некоторую обобщенную неполиномиальную форму дискретной функции с различающимися значностями переменных.

Из построения видно, что отличительной особенностью синтезированной формы является ее чувствительность к разложению функций, зависящим не от всех, а от части переменных множества $X = \{x_0, x_1, \ldots, x_{n-1}\}$. Это равносильно наличию в спектральном базисе функций с различными частотными свойствами и с различными областями изме-

нения. Другая отличительная особенность формы состоит в том, что матрицы дискретного преобразования факторизуются в произведение матриц меньшей размерности.

Факторизация консонансного преобразования. Заметим, что из ортогональности ϕ_i, $\varphi_i^{(t)}$ и $\psi_{ij}^{(t)}$ следует их фундаментальность. Тогда, используя структуру выражения (П4.22) матрицу прямого преобразования $\mathbf{D} = \mathbf{D}_{n-s}$, определим из рекуррентного уравнения

$$\mathbf{D}_{t+1} = (\mathbf{T}_{k_0^{[t]}} \otimes \mathbf{\Phi}^{(t)}) \times \begin{bmatrix} \mathbf{D}_t & \boldsymbol{\sigma}_{k_t^{[s]}} & \cdots & \boldsymbol{\sigma}_{k_t^{[s]}} \\ \boldsymbol{\sigma}_{k_t^{[s]}} & \mathbf{\Psi}_1^{(t)} & \cdots & \boldsymbol{\sigma}_{k_t^{[s]}} \\ \vdots & \vdots & \ddots & \vdots \\ \boldsymbol{\sigma}_{k_t^{[s]}} & \boldsymbol{\sigma}_{k_t^{[s]}} & \cdots & \mathbf{\Psi}_{k_t-1}^{(t)} \end{bmatrix} \qquad (t = \overline{0, n-s-1}), \qquad \text{(П4.23)}$$

а матрицу обратного преобразования $\mathbf{Q} = \mathbf{Q}_{n-s}$ из уравнения

$$\mathbf{Q}_{t+1} = \begin{bmatrix} \mathbf{Q}_t & \boldsymbol{\sigma}_{k_t^{[s]}} & \cdots & \boldsymbol{\sigma}_{k_t^{[s]}} \\ \boldsymbol{\sigma}_{k_t^{[s]}} & \mu\,\mathbf{\Psi}_1^{(t)} & \cdots & \boldsymbol{\sigma}_{k_t^{[s]}} \\ \vdots & \vdots & \ddots & \vdots \\ \boldsymbol{\sigma}_{k_t^{[s]}} & \boldsymbol{\sigma}_{k_t^{[s]}} & \cdots & \mu\,\mathbf{\Psi}_{k_t-1}^{(t)} \end{bmatrix} \times (\mathbf{T}_{k_0^{[t]}} \otimes \mu\,\mathbf{\Phi}^{(t)}) \quad (t = \overline{0, n-s-1}), \qquad \text{(П4.24)}$$

с начальными условиями

$$\mathbf{D}_0 = \mathbf{\Phi}, \qquad \mathbf{\Psi}_i^{(0)} = \mathbf{T}_{k_0^{[s]}};$$
$$\mathbf{Q}_0 = \mu\mathbf{\Phi}, \quad \mu\,\mathbf{\Psi}_i^{(0)} = \mathbf{T}_{k_0^{[s]}},$$

где $\boldsymbol{\sigma}_{k_t^{[s]}}$ и $\mathbf{T}_{k_t^{[s]}}$ – соответственно нулевая и единичная матрицы размерности $k_t^{[s]}$, μ – операция обращения матрицы,

$$\begin{cases} \mathbf{\Phi} & = & [\phi_i(j)] & (i, j = \overline{0, k_0^{[s]} - 1}); \\ \mathbf{\Phi}^{(t)} & = & [\varphi_i^{(t)}(j)] & (i, j = \overline{0, k_t^{[s+1]} - 1}); \\ \mathbf{\Psi}_i^{(t)} & = & [\psi_{ij}^{(t)}(l)] & (i = \overline{1, k_t - 1},\ j, l = \overline{0, k_0^{[t]} - 1}). \end{cases} \qquad \text{(П4.25)}$$

а $\varphi_0^{(t)}(j) = \tau$ для всех j.

Для доказательства (П4.23) и (П4.24) достаточно найти $\mathbf{Q} \times \mathbf{D}$ и удостовериться, что произведение этих матриц равно $\mathbf{T}_{k_0^{[n]}}$. При этом используется теорема П4.1 и известное свойство обращения произведения квадратных матриц $\mathbf{A}$ и $\mathbf{B}$:

$$\mu(\mathbf{A} \times \mathbf{B}) = \mu\mathbf{B} \times \mu\mathbf{A},$$

где μ – операция обращения матрицы.

Разрешение спектрального базиса. Форма (П4.22) является параметрической, т.к. зависит от параметра $s = \overline{0, n}$, задающего ***разрешение*** ее спектрального базиса. Величина

s определяет минимальное количество переменных, от которых существенным образом зависят спектральные функции.

При $s = 0$ конструкция (П4.22) имеет наибольшее разрешение:

$$f(x_0^{[n]}) = a + \sum_{t=0}^{n-1} \sum_{i=1}^{k_t - 1} \varphi_i^{(t)}(x_t) \times \left\{ \sum_{j=0}^{k_0^{[t]}-1} \psi_{ij}^{(t)}(x_0^{[t]}) \times a_{ij}^{(t)} \right\}. \qquad \text{(П4.26)}$$

В этом случае в спектральном базисе имеются функции, не зависящие ни от одной переменной $x_0^{[0]} = \varnothing$, существенным образом зависящие от переменной $x_0^{[1]} = \{x_0\}$, от переменных $x_0^{[2]} = \{x_0, x_1\}$, и т.д., вплоть до существенной зависимости от всего множества переменных $x_0^{[n]} = \{x_0, x_1, \ldots, x_{n-1}\}$.

При $s = 1$ разрешение аналитической конструкции уменьшается,

$$f(x_0^{[n]}) = \sum_{i=0}^{k_0 - 1} \varphi_i(x_0) \times a_i + \sum_{t=0}^{n-2} \sum_{i=k_t}^{k_t^{[2]}-1} \varphi_i^{(t)}(x_t^{[2]}) \times \left\{ \sum_{j=0}^{k_0^{[t]}-1} \psi_{ij}^{(t)}(x_0^{[t]}) \times a_{ij}^{(t)} \right\},$$

и в спектральном базисе используются функции, зависящие от одной и более переменных.

В предпоследнем случае, когда $s = n - 1$, из (П4.22) получаем конструкцию

$$f(x_0^{[n]}) = \sum_{i=0}^{k_0^{[n-1]}-1} \varphi_i(x_0^{[n-1]}) \times a_i + \sum_{i=k_0^{[n-1]}}^{k_0^{[n]}-1} \varphi_i^{(0)}(x_0^{[n]}) \times a_{i0}^{(0)},$$

которая состоит из спектральных функций, зависящих от $x_0^{[n-1]}$ и $x_0^{[n]}$ переменных.

И, наконец, когда $s = n$ имеем спектральную форму с наименьшим разрешением,

$$f(x_0^{[n]}) = \sum_{i=0}^{k_0^{[n]}-1} \varphi_i(x_0^{[n]}) \times a_i. \qquad \text{(П4.27)}$$

Следует заметить, что разрешение спектрального базиса не гарантирует отсутствия в нем функций с существенной зависимостью менее чем от s переменных. Так как на выбор функций ϕ_i, $\varphi_i^{(t)}$ и $\psi_{ij}^{(t)}$ накладывается лишь одно ограничение – они должны быть взаимно ортогональны, а, значит, зависеть от любого числа переменных, то рамках одной и той же конструкции могут существовать спектральные функции с существенной зависимостью и менее чем от s переменных.

Например, спектральное разложение (П4.27) формально записывается в форме, у которой функции показаны зависящими от всего множества переменных. Однако она является универсальной и каждое разложение есть частный ее случай.

Таким образом, разрешение s означает наличие в спектральном базисе функций, существенно зависящих от s переменных.

Пример П4.6. Покажем синтез неполиномиальной формы (П4.26), получаемой из (П4.22) при наибольшем ее разрешении $s=0$. Осуществим разложение функции из табл. П4.3 в аддитивной алгебре R_A с поразрядными операциями неэквиваленции и конъюнкции, заданными на множестве $N_4=\{0,1,2,3\}$ матрицами:

$$+=\begin{bmatrix}0&1&2&3\\1&0&3&2\\2&3&0&1\\3&2&1&0\end{bmatrix},\quad \times=\begin{bmatrix}0&0&0&0\\0&1&0&1\\0&0&2&2\\0&1&2&3\end{bmatrix}.$$

Учитывая (П4.25), положим

$$\mathbf{\Phi}=[\tau],\quad \mathbf{\Phi}^{(0)}=\begin{bmatrix}\tau&\sigma&\sigma\\\tau&\tau&\sigma\\\tau&\tau&\tau\end{bmatrix},\quad \mathbf{\Phi}^{(1)}=\begin{bmatrix}\tau&\sigma\\\tau&\tau\end{bmatrix},$$

$$\mathbf{\Psi}^{(0)}=[\tau],\quad \mathbf{\Psi}_1^{(1)}=\begin{bmatrix}\tau&\sigma&\tau\\\tau&\tau&\sigma\\\sigma&\sigma&\tau\end{bmatrix},$$

где $\sigma=0$ и $\tau=3$. Тогда в соответствии с (П4.24) получим

$$\mathbf{Q}=\left[\begin{array}{ccc|ccc}1&0&0&0&0&0\\-1&1&0&0&0&0\\0&-1&1&0&0&0\\\hline-1&0&1&1&0&-1\\1&-1&-1&-1&1&1\\0&0&-1&0&0&1\end{array}\right],\quad \mathbf{A}=\mathbf{Q}\cdot\mathbf{F}=\begin{bmatrix}3\\3\\1\\1\\3\\0\end{bmatrix},$$

где $\cdot$ – операция циклической суммы. В итоге имеем:

$$f(x_0,x_1)=3+\varphi_1^{(0)}(x_0)\times 3+\varphi_2^{(0)}(x_0)\times 1+\varphi_1^{(1)}(x_1)\times\{\psi_{10}^{(1)}(x_0)\times 1+\psi_{11}^{(1)}(x_0)\times 3\},$$

и, учитывая определения спектральных функций и операций образующей алгебры,

$$f(x_0,x_1)=3+\begin{bmatrix}0\\3\\2\end{bmatrix}(x_0)+\begin{bmatrix}0\\3\end{bmatrix}(x_1)\times\begin{bmatrix}1\\2\\0\end{bmatrix}(x_0),$$

где $[y_i](x)=y_x$. На языке программирования Си искомая функция вычисляется выражением $3\hat{}(-x0)\hat{}\,x1?x0+1:0$. ♦

П4.2.4. Неполиномиальная форма Хаара

На практике используются два представления дискретных функций, основанные на разложениях по системам прямоугольных спектральных функций, – формы Уолша и Хаара. Ортогональные функции Уолша принимают значения на множестве $W=\{-1,1\}$ и име-

ют полиномиальное представление. Обобщенные полиномиальные формы Уолша для дискретных функций с различной значностью переменных и в произвольной образующей алгебре рассмотрены на с. 420. В свою очередь, ортогональные функции Хаара не имеют полиномиального представления. Однако благодаря низкой вычислительной сложности они получили широкое распространение.

Ортонормированные функции Хаара. Рассмотрим представления дискретных функций, получаемые из (П4.26) при одинаковых значностях переменных, равных 2:

$$f(x_0^{[n]}) = a + \sum_{t=0}^{n-1} \varphi_1^{(t)}(x_t) \times \left\{ \sum_{j=0}^{2^t-1} \psi_j^{(t)}(x_0, x_1, \ldots, x_{t-1}) \times a_j^{(t)} \right\}. \qquad (П4.28)$$

Из (П4.28) на кольце целых чисел может быть получено разложение функции по ортонормированным функциям Хаара:

$$f(x_0^{[n]}) = \frac{1}{2^n} \left(a + \sum_{t=0}^{n-1} \sqrt{2^t}\,(-1)^{x_t} \left\{ \sum_{j=0}^{2^t-1} a_j^{(t)} \prod_{i=0}^{t-1} x_i^{j_i} \right\} \right), \qquad (П4.29)$$

которое синтезируется при полиномиальном представлении спектральных функций:

$$\varphi_0^{(t)}(x_t) = 1, \quad \varphi_1^{(t)}(x_t) = \sqrt{2^t}\,(-1)^{x_t}, \quad \psi_j^{(t)}(x_0, x_1, \ldots, x_{t-1}) = \prod_{i=0}^{t-1} x_i^{j_i}, \qquad (П4.30)$$

где степенные операции для $\psi_j^{(t)}$ определены так:

$$x_i^p = \begin{cases} \overline{x}_i, & \text{при } p = 0; \\ x_i, & \text{при } p = 1, \end{cases}$$

а значения степеней определяются представлением индекса j в двоичной системе счисления, $j = (j_0\ j_1 \ldots j_{n-1})_2$.

В рассматриваемом случае $H_0 = \varphi_0^{(t)} = 1$ и

$$H_t^j(x_{n-1}, \ldots, x_1, x_0) = \varphi_1^{(t)}(x_t)\, \psi_j^{(t)}(x_0, x_1, \ldots, x_{t-1}), \qquad (П4.31)$$

где $t = \overline{0, n-1}$, $j = \overline{0, 2^t - 1}$, H_t^j – функция Хаара степени t порядка j,

$$H_t^j(x) = \begin{cases} 0, & 0 \le x < \dfrac{j}{2^t}; \\ -\sqrt{2^t}, & \dfrac{j}{2^t} \le x < \dfrac{j+0{,}5}{2^t}; \\ \sqrt{2^t}, & \dfrac{j+0{,}5}{2^t} \le x < \dfrac{j+1}{2^t}; \\ 0, & \dfrac{j+1}{2^t} \le x < 1, \end{cases} \qquad (П4.32)$$

при следующем нормированном представлении x в двоичной системе счисления,

$$x = \frac{1}{2^n}(x_{n-1}\, x_{n-2} \ldots x_0)_2, \quad x = \frac{1}{2^n} \sum_{p=0}^{n-1} x_p 2^{n-p-1}, \qquad (П4.33)$$

т.е. порядок перечисления переменных у функций Хаара противоположен порядку переменных $\varphi_1^{(t)}$ и $\psi_j^{(t)}$.

Из (П4.32) видно, что степень функции t определяет, сколько раз функция пересекает координатную ось, а порядок j задает позицию (место) изменения функции в области ее определения.

Для доказательства представимости разложения Хаара в виде (П4.29) рассмотрим выражение H_t^j, полученное из (П4.30):

$$H_t^j(x) = \sqrt{2^t}(-1)^{x_{n-t-2}} \prod_{i=n-t-1}^{n-1} x_i^{j_i}. \qquad \text{(П4.34)}$$

Пусть $0 \le x < \frac{j}{2^t}$. Тогда с учетом (П4.33) имеем

$$(0,0,\ldots,0)_2 \le (x_{n-1},\ldots,x_0)_2 < 2^{n-t} j = (0,\ldots,0,j_0,\ldots,j_{t-1})_2$$

и из (П4.34) получаем $H_t^j(x) = 0$, т.к. $x_{n-t-1} = 0$.

При $\frac{j}{2^t} \le x < \frac{j+0{,}5}{2^t}$ получим

$$(0,\ldots,0,j_0,\ldots,j_{t-1})_2 \le (x_{n-1},\ldots,x_0)_2 < 2^{n-t} j + 2^{n-t-1} = (0,\ldots,1,j_0,\ldots,j_{t-1})_2,$$

откуда находим $H_t^j(x) = -\sqrt{2^t}$, т.к. в отличие от предыдущего случай $x_{n-t-1} = 1$ и существует набор переменных, для которого выражение под знаком произведения равен 1.

В свою очередь, при $\frac{j+0{,}5}{2^t} \le x < \frac{j+1}{2^t}$ имеем

$$(0,\ldots,1,j_0,\ldots,j_{t-1})_2 \le (x_{n-1},\ldots,x_0)_2 < 2^{n-t} j + 2^{n-t} = (0,\ldots,1,1,j_0,\ldots,j_{t-1})_2,$$

и $H_t^j(x) = \sqrt{2^t}$.

И, наконец, при $\frac{j+1}{2^t} \le x < 1$

$$(0,\ldots,1,1,j_0,\ldots,j_{t-1})_2 \le (x_{n-1},\ldots,x_0)_2 < (1,\ldots,1)_2,$$

откуда получаем $H_t^j(x) = 0$.

Нами показано, что из (П4.22) может быть получено разложение дискретной функции по ортонормированным функциям Хаара, которое синтезируется при разрешении $s = 0$, одинаковых значностях переменных $k_i = 2$ и полиномиальной аналитической конструкции (П4.30) спектральных функций $\varphi_i^{(t)}$ и $\psi_{ij}^{(t)}$ ($i = \overline{0,1}$).

Дискретное преобразование Хаара. Найдем рекуррентные выражения для матриц дискретного преобразования Хаара. Из (П4.23) и (П4.24) при $s = 0$ и одинаковых значностях переменных, равных 2, находим:

$$\mathbf{D}_0 = 1, \quad \mathbf{D}_{t+1} = \left(\mathbf{T}_{2^t} \otimes \begin{bmatrix} 1 & \sqrt{2^t} \\ 1 & -\sqrt{2^t} \end{bmatrix} \right) \times \left[\begin{array}{c|c} \mathbf{D}_t & \boldsymbol{\sigma}_{2^t} \\ \hline \boldsymbol{\sigma}_{2^t} & \mathbf{T}_{2^t} \end{array} \right] \ (t = \overline{0, n-1}), \quad \mathbf{D} = \mathbf{D}_{n-1}, \qquad (\text{П4.35})$$

$$\mathbf{Q}_0 = 1, \quad \mathbf{Q}_{t+1} = \begin{bmatrix} \mathbf{Q}_t & \boldsymbol{\sigma}_{2^t} \\ \boldsymbol{\sigma}_{2^t} & \mathbf{T}_{2^t} \end{bmatrix} \times \left(\mathbf{T}_{2^t} \otimes \begin{bmatrix} 1 & \sqrt{2^t} \\ 1 & -\sqrt{2^t} \end{bmatrix} \right) \ (t = \overline{0, n-1}), \quad \mathbf{Q} = \mathbf{Q}_{n-1}, \qquad (\text{П4.36})$$

где также использованы матрицы спектральных функций $\varphi_i^{(t)}$ и $\psi_{ij}^{(t)}$, задаваемые (П4.30).

Из (П4.31) следует, что порядок переменных у ортонормированных функций Хаара H_t^j является обратным по отношению к порядку переменных $\varphi_1^{(t)}$ и $\psi_j^{(t)}$. Следовательно, для получения классической матрицы дискретного преобразования Хаара необходима перестановка строк $\mathbf{D}$ (столбцов $\mathbf{Q}$) в порядке, при котором строка (столбец) i занимает место строки (столбца) j.

$$\begin{cases} i = (i_0, i_1, \ldots, i_{n-1})_2; \\ j = (i_{n-1}, i_{n-2}, \ldots, i_0)_2. \end{cases} \qquad (\text{П4.37})$$

В результате перестановки получаем матрицу ортонормированных функций Хаара, упорядоченных по своим частотам. Аналогичным образом, изменяя порядок столбцов $\mathbf{D}$ и строк $\mathbf{Q}$, получаем функции Хаара с естественной упорядоченностью.

Пример П4.7. Используя (П4.35) и (П4.36), синтезируем матрицы прямого и обратного преобразования Хаара при $n = 3$.

$$\mathbf{D}_1 = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}, \quad \mathbf{D}_2 = \begin{bmatrix} 1 & 1 & \sqrt{2} & 0 \\ 1 & -1 & 0 & \sqrt{2} \\ 1 & 1 & -\sqrt{2} & 0 \\ 1 & -1 & 0 & -\sqrt{2} \end{bmatrix},$$

$$\mathbf{D}_3 = \left[\begin{array}{cccc|cccc} 1 & 1 & \sqrt{2} & 0 & 2 & 0 & 0 & 0 \\ 1 & -1 & 0 & \sqrt{2} & 0 & 2 & 0 & 0 \\ 1 & 1 & -\sqrt{2} & 0 & 0 & 0 & 2 & 0 \\ 1 & -1 & 0 & -\sqrt{2} & 0 & 0 & 0 & 2 \\ \hline 1 & 1 & \sqrt{2} & 0 & -2 & 0 & 0 & 0 \\ 1 & -1 & 0 & \sqrt{2} & 0 & -2 & 0 & 0 \\ 1 & 1 & -\sqrt{2} & 0 & 0 & 0 & -2 & 0 \\ 1 & -1 & 0 & -\sqrt{2} & 0 & 0 & 0 & -2 \end{array} \right];$$

$$\mathbf{Q}_1 = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}, \quad \mathbf{Q}_2 = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ \sqrt{2} & 0 & -\sqrt{2} & 0 \\ 0 & \sqrt{2} & 0 & -\sqrt{2} \end{bmatrix},$$

$$\mathbf{Q}_3 = \left[\begin{array}{cccc:cccc} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\ \sqrt{2} & 0 & -\sqrt{2} & 0 & \sqrt{2} & 0 & -\sqrt{2} & 0 \\ 0 & \sqrt{2} & 0 & -\sqrt{2} & 0 & \sqrt{2} & 0 & -\sqrt{2} \\ \hdashline 2 & 0 & 0 & 0 & -2 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 & -2 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 & 0 & -2 & 0 \\ 0 & 0 & 0 & 2 & 0 & 0 & 0 & -2 \end{array}\right].$$

После перестановки строк $\mathbf{D}_3$ и столбцов $\mathbf{Q}_3$ в соответствии с табл. П4.4, полученной из (П4.37), находим:

$$\mathbf{D} = \begin{array}{ccc|c} x_0 & x_1 & x_2 & \\ \hline 0 & 0 & 0 & \\ 1 & 0 & 0 & \\ 0 & 1 & 0 & \\ 1 & 1 & 0 & \\ 0 & 0 & 1 & \\ 1 & 0 & 1 & \\ 0 & 1 & 1 & \\ 1 & 1 & 1 & \end{array} \begin{array}{c} H_0 \quad H_0^0 \quad H_1^0 \quad H_1^1 \quad H_2^0 \quad H_2^1 \quad H_2^2 \quad H_2^3 \\ \left[\begin{array}{cccc:cccc} 1 & 1 & \sqrt{2} & 0 & 2 & 0 & 0 & 0 \\ 1 & 1 & \sqrt{2} & 0 & -2 & 0 & 0 & 0 \\ 1 & 1 & -\sqrt{2} & 0 & 0 & 0 & 2 & 0 \\ 1 & 1 & -\sqrt{2} & 0 & 0 & 0 & -2 & 0 \\ \hdashline 1 & -1 & 0 & \sqrt{2} & 0 & 2 & 0 & 0 \\ 1 & -1 & 0 & \sqrt{2} & 0 & -2 & 0 & 0 \\ 1 & -1 & 0 & -\sqrt{2} & 0 & 0 & 0 & 2 \\ 1 & -1 & 0 & -\sqrt{2} & 0 & 0 & 0 & -2 \end{array}\right] \end{array},$$

$$\mathbf{Q} = \begin{array}{ccc|c} x_0 & x_1 & x_2 & \\ \hline 0 & 0 & 0 & \\ 1 & 0 & 0 & \\ 0 & 1 & 0 & \\ 1 & 1 & 0 & \\ 0 & 0 & 1 & \\ 1 & 0 & 1 & \\ 0 & 1 & 1 & \\ 1 & 1 & 1 & \end{array} \begin{array}{c} \overline{H}_0 \quad \overline{H}_0^0 \quad \overline{H}_1^0 \quad \overline{H}_1^1 \quad \overline{H}_2^0 \quad \overline{H}_2^1 \quad \overline{H}_2^2 \quad \overline{H}_2^3 \\ \left[\begin{array}{cccc:cccc} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ \sqrt{2} & \sqrt{2} & -\sqrt{2} & -\sqrt{2} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \sqrt{2} & \sqrt{2} & -\sqrt{2} & -\sqrt{2} \\ \hdashline 2 & -2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 2 & -2 & 0 & 0 \\ 0 & 0 & 2 & -2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 2 & -2 \end{array}\right] \end{array},$$

где $\overline{H}_t^j$ – функции ортогональные (обратные) H_t^j.

Таблица П4.4. Упорядочивание по Хаару

i	0	1	2	3	4	5	6	7
	$(000)_2$	$(100)_2$	$(010)_2$	$(110)_2$	$(001)_2$	$(101)_2$	$(011)_2$	$(111)_2$
j	0	4	2	6	1	5	3	7
	$(000)_2$	$(001)_2$	$(010)_2$	$(011)_2$	$(100)_2$	$(101)_2$	$(110)_2$	$(111)_2$

И, наконец, для естественного порядка функций Хаара имеем:

$$\mathbf{D}=\left[\begin{array}{cccc:cccc}1 & 2 & \sqrt{2} & 0 & 1 & 0 & 0 & 0\\ 1 & -2 & \sqrt{2} & 0 & 1 & 0 & 0 & 0\\ 1 & 0 & -\sqrt{2} & 2 & 1 & 0 & 0 & 0\\ 1 & 0 & -\sqrt{2} & -2 & 1 & 0 & 0 & 0\\ \hdashline 1 & 0 & 0 & 0 & -1 & 2 & \sqrt{2} & 0\\ 1 & 0 & 0 & 0 & -1 & -2 & \sqrt{2} & 0\\ 1 & 0 & 0 & 0 & -1 & 0 & -\sqrt{2} & 2\\ 1 & 0 & 0 & 0 & -1 & 0 & -\sqrt{2} & -2\end{array}\right],$$

$$\mathbf{Q}=\left[\begin{array}{cccc:cccc}1 & 1 & 1 & 1 & 1 & 1 & 1 & 1\\ 2 & -2 & 0 & 0 & 0 & 0 & 0 & 0\\ \sqrt{2} & \sqrt{2} & -\sqrt{2} & -\sqrt{2} & 0 & 0 & 0 & 0\\ 0 & 0 & 2 & -2 & 0 & 0 & 0 & 0\\ \hdashline 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1\\ 0 & 0 & 0 & 0 & 2 & -2 & 0 & 0\\ 0 & 0 & 0 & 0 & \sqrt{2} & \sqrt{2} & -\sqrt{2} & -\sqrt{2}\\ 0 & 0 & 0 & 0 & 0 & 0 & 2 & -2\end{array}\right]. \blacklozenge$$

Хааро-подобные формы. Известны расширения спектрального базиса Хаара, которые представляются как некоторая адаптация системы ортонормированных функций Хаара к различным прикладным задачам. Перечислим только некоторые из них: конусообразные, обобщение Ватари, комплексные, действительные многозначные, знаковые, усеченные, обобщенные в p-адических группах. Как правило, расширения, сохраняя значение степени и порядка функции, изменяют форму импульсов (конусообразные, знаковые).

Синтез других форм выполняется в рамках конструкции (П4.29) при различных степенных операциях. В табл. П4.5 классифицированы возможные Хааро-подобные формы, где заданы знаковая матрица, определяемая $\varphi_1^{(t)}$ и степенные операции для $\psi_j^{(t)}$.

В ввиду того, что в табл. П4.5 перечислены все возможные знаковые матрицы, а также, с точностью до знака, все степенные операции над множествами $B=\{0,1\}$ и $T=\{-1,0,1\}$, то других Хааро-подобных форм, синтезируемых в рамках аналитической конструкции (П4.29), не существует[95]. Примеры Хааро-подобных спектральных базисов, полученных при $n=3$, приведены на рис. П4.1 и рис. П4.2.

[95] Следует сказать, что не исключено задание произвольного набора степенных операций, в том числе и для каждой спектральной функции $\psi_j^{(t)}$. В этом случае количество порождаемых форм значительно возрастает и появляется возможность учесть особенности представляемых функций или их классов, например, для минимизации числа ненулевых коэффициентов формы.

Таблица П4.5. Хааро-подобные формы

Форма		Знаковая матрица	Степенные операции
Хаара	прямая	$\begin{bmatrix}1 & 1\\ 1 & -1\end{bmatrix}$	$\begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix}\ \begin{bmatrix}0 & 1\\ 1 & 0\end{bmatrix}$
	инверсная	$\begin{bmatrix}1 & -1\\ 1 & 1\end{bmatrix}$	
Уплотненная	прямая	$\begin{bmatrix}1 & 1\\ 1 & -1\end{bmatrix}$	$\begin{bmatrix}1 & 0\\ 1 & 1\end{bmatrix}\ \begin{bmatrix}1 & 1\\ 1 & 0\end{bmatrix}\ \begin{bmatrix}0 & 1\\ 1 & 1\end{bmatrix}\ \begin{bmatrix}1 & 1\\ 0 & 1\end{bmatrix}$
	инверсная	$\begin{bmatrix}1 & -1\\ 1 & 1\end{bmatrix}$	
Усеченная	положительная	$\begin{bmatrix}1 & 1\\ 1 & 0\end{bmatrix}\ \begin{bmatrix}1 & 0\\ 1 & 1\end{bmatrix}$	$\begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix}\ \begin{bmatrix}0 & 1\\ 1 & 0\end{bmatrix}$ $\begin{bmatrix}1 & 0\\ 1 & 1\end{bmatrix}\ \begin{bmatrix}1 & 1\\ 1 & 0\end{bmatrix}\ \begin{bmatrix}0 & 1\\ 1 & 1\end{bmatrix}\ \begin{bmatrix}1 & 1\\ 0 & 1\end{bmatrix}$
	отрицательная	$\begin{bmatrix}1 & 0\\ 1 & -1\end{bmatrix}\ \begin{bmatrix}1 & -1\\ 1 & 0\end{bmatrix}$	
Апериодическая	прямая	$\begin{bmatrix}1 & 1\\ 1 & -1\end{bmatrix}$	$\begin{bmatrix}1 & 0\\ 0 & -1\end{bmatrix}\ \begin{bmatrix}0 & 1\\ -1 & 0\end{bmatrix}$ $\begin{bmatrix}-1 & 0\\ 1 & 1\end{bmatrix}\ \begin{bmatrix}1 & 0\\ -1 & 1\end{bmatrix}\ \begin{bmatrix}1 & 0\\ 1 & -1\end{bmatrix}$ $\begin{bmatrix}-1 & 1\\ 1 & 0\end{bmatrix}\ \begin{bmatrix}1 & 1\\ -1 & 0\end{bmatrix}\ \begin{bmatrix}1 & -1\\ 1 & 0\end{bmatrix}$ $\begin{bmatrix}0 & -1\\ 1 & 1\end{bmatrix}\ \begin{bmatrix}0 & 1\\ 1 & -1\end{bmatrix}\ \begin{bmatrix}0 & 1\\ -1 & 1\end{bmatrix}$ $\begin{bmatrix}1 & -1\\ 0 & 1\end{bmatrix}\ \begin{bmatrix}1 & 1\\ 0 & -1\end{bmatrix}\ \begin{bmatrix}-1 & 1\\ 0 & 1\end{bmatrix}$
	инверсная	$\begin{bmatrix}1 & -1\\ 1 & 1\end{bmatrix}$	
	положительная	$\begin{bmatrix}1 & 1\\ 1 & 0\end{bmatrix}\ \begin{bmatrix}1 & 0\\ 1 & 1\end{bmatrix}$	
	отрицательная	$\begin{bmatrix}1 & 0\\ 1 & -1\end{bmatrix}\ \begin{bmatrix}1 & -1\\ 1 & 0\end{bmatrix}$	

Заметим, что форма (П4.28) позволяет синтезировать множество представлений, различающихся $\varphi_1^{(t)}$ и $\psi_j^{(t)}$. Более того, полиномиальная реализация $\psi_j^{(t)}$ не является обязательной, могут быть использованы, в том числе, и неполиномиальные представления спектральных функций $\psi_j^{(t)}$. Очевидно, выбор той или иной Хааро-подобной формы может определяться эффективностью вычисления базиса на используемом для этого вычислительном средстве.

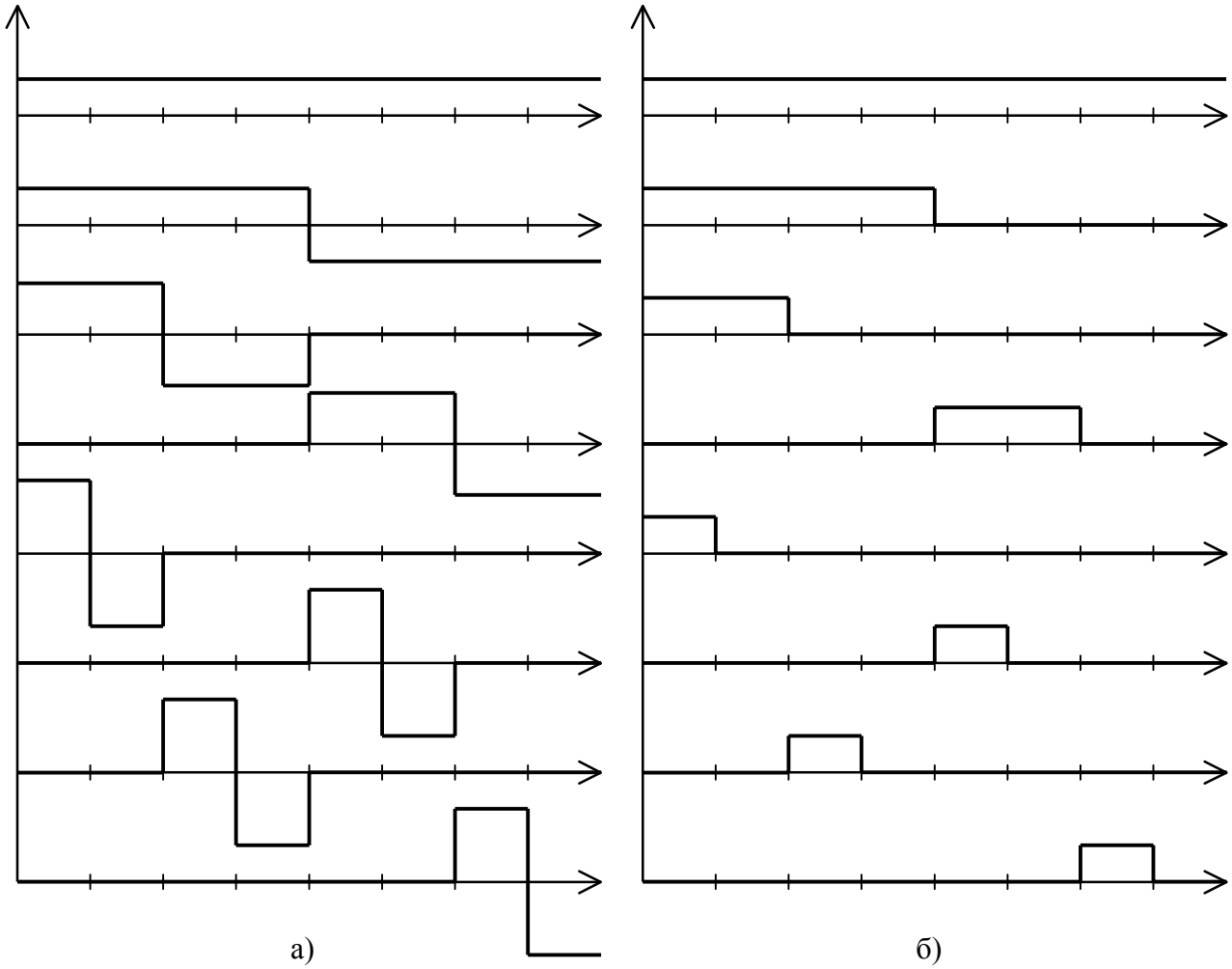

Рис. П4.1. Хааро-подобные системы спектральных функций:
а) Хаара, б) усеченная положительная.

П4.2.5. Неполиномиальные формы в аддитивной алгебре

Вычислительная сложность рассмотренных Хааро-подобных форм почти одинакова. Используемые знаковые и степенные операции не имеют прямой реализации на вычислительных средствах с арифметико-логическим устройством, выполняющем арифметические (сложение ***add***, вычитание ***sub***, изменение знака ***neg***) и поразрядные логические (конъюнкция ***and***, дизъюнкция ***or***, неэквиваленция ***xor***, отрицание ***not***) операции. Более того, вычисление дискретного преобразования в формате чисел с плавающей запятой требует дорогостоящих аппаратурных средств.

С другой стороны, выполнение дискретных преобразований, определенных над целостными кольцами и конечными полями, не изменяет принципиально сложившуюся ситуацию. Такие алгебраические системы, как правило, также не имеют прямой аппаратной реализации на существующих вычислительных средствах.

Однако существуют алгебраические системы, непосредственно реализуемые современными вычислительными средствами и позволяющие выполнять дискретные преобразования в неполиномиальных базисах.

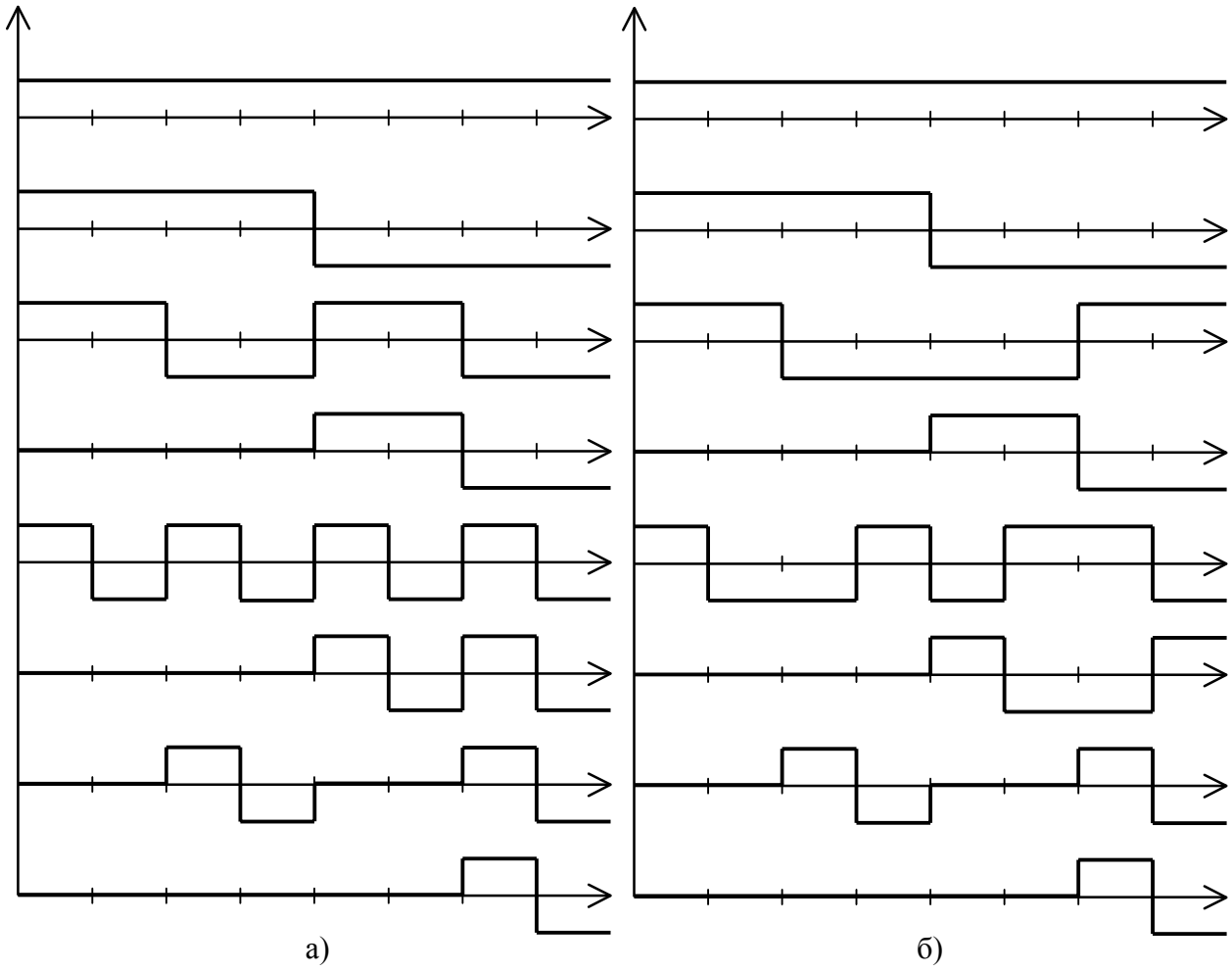

Рис. П4.2. Знаковое представление Хааро-подобных функций:
а) уплотненных, б) апериодических.

Компьютерные алгебры. Рассмотрим алгебраическую систему $R_A = \langle N_k, +, \times \rangle$, образованную на множестве из $k = 2^r$ элементов N_k операциями порязрадной неэквиваленции ***xor*** или арифметического сложения ***add***, рассматриваемыми как операции алгебраического сложения, и поразрядной конъюнкции ***and*** – как алгебраическое умножение, где r – разрядность арифметико-логического устройства.

Находим, что операции алгебраическое сложения образуют коммутативные группы $G_{\text{add}} = \langle N_k, add \rangle$ и $G_{\text{xor}} = \langle N_k, xor \rangle$ с нулем $\sigma = 0$, т.е. для любого элемента $a \in G_{\text{add}}$ или $a \in G_{\text{xor}}$ существует противоположный ему элемент $-a$, такой, что $a + (-a) = a - a = \sigma$, где знаком – обозначена обратная операция. Обратной операцией для ***add*** является операция вычитания ***sub***, а для поразрядной неэквиваленции – сама операция ***xor***. Таким образом,

$$\boldsymbol{add}(a, -b) = \boldsymbol{sub}(a, b) = \boldsymbol{add}(a, \boldsymbol{neg}(b)), \quad \boldsymbol{xor}(a, -b) = \boldsymbol{xor}(a, b),$$

то есть, противоположный элемент в G_{add} образуется операцией изменения знака ***neg***, а в G_{xor} противоположный элемент совпадает с исходным.

В свою очередь, для алгебраического умножения – поразрядной конъюнкции and – имеем

$$\sigma \times a = \sigma, \quad \tau \times a = a,$$

где $\tau = 2^r - 1$ – единица алгебры.

Следовательно, алгебраические системы $R_A^{\text{add}} = \langle N_{2^r}, \boldsymbol{add}, \boldsymbol{and} \rangle$ и $R_A^{\text{xor}} = \langle N_{2^r}, \boldsymbol{xor}, \boldsymbol{and} \rangle$ являются аддитивными образующими алгебрами, и в них может быть выполнено неполиномиальным разложение (П4.22).

В соответствии с указанной теоремой, матрица прямого преобразования в алгебре R_A^{add} должна быть биполярной, т.е. состоять из нулей, единиц и минус единиц. Однако в алгебре R_A^{xor} матрица прямого преобразования может быть только логической, т.е. состоять из нулей и единиц образующей алгебры. Последнее вызвано тем, что в этой алгебре прямой и обратный элементы совпадают.

В свою очередь, группа G_{add} является циклической и ее порядок равен $2^r - 1$, а циклический порядок группы G_{xor} равен 2, что следует из $\boldsymbol{xor}(a, a) = \sigma$ при любом a.

Таким образом имеются две аддитивные образующие алгебры, имеющие непосредственную реализацию на современных вычислительных средствах. Последнее означает, что представления дискретных функций в неполиномиальных формах компьютерных алгебр может иметь наибольшую эффективность.

Пример П4.8. Приведем таблицы операций аддитивных алгебр R_A^{add} и R_A^{xor}, реализуемых 3-разрядным арифметико-логическим устройством при беззнаковом представлении чисел, т.е. при $N_k = \{0, 1, \ldots, 7\}$:

$$\boldsymbol{add} = \begin{array}{c|cccccccc} & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ \hline 0 & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 1 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 0 \\ 2 & 2 & 3 & 4 & 5 & 6 & 7 & 0 & 1 \\ 3 & 3 & 4 & 5 & 6 & 7 & 0 & 1 & 2 \\ 4 & 4 & 5 & 6 & 7 & 0 & 1 & 2 & 3 \\ 5 & 5 & 6 & 7 & 0 & 1 & 2 & 3 & 4 \\ 6 & 6 & 7 & 0 & 1 & 2 & 3 & 4 & 5 \\ 7 & 7 & 0 & 1 & 2 & 3 & 4 & 5 & 6 \end{array},$$

$$\boldsymbol{xor} = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \end{array} \begin{array}{c} \begin{array}{cccccccc} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \end{array} \\ \begin{bmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 1 & 0 & 3 & 2 & 5 & 4 & 7 & 6 \\ 2 & 3 & 0 & 1 & 6 & 7 & 4 & 5 \\ 3 & 2 & 1 & 0 & 7 & 0 & 1 & 4 \\ 4 & 5 & 6 & 7 & 0 & 1 & 2 & 3 \\ 5 & 4 & 7 & 0 & 1 & 0 & 3 & 2 \\ 6 & 7 & 4 & 1 & 2 & 3 & 0 & 1 \\ 7 & 6 & 5 & 4 & 3 & 2 & 1 & 0 \end{bmatrix} \end{array},$$

$$\boldsymbol{and} = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \end{array} \begin{array}{c} \begin{array}{cccccccc} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 2 & 2 & 0 & 0 & 2 & 2 \\ 0 & 1 & 2 & 3 & 0 & 1 & 2 & 3 \\ 0 & 0 & 0 & 0 & 4 & 4 & 4 & 4 \\ 0 & 1 & 0 & 1 & 4 & 5 & 4 & 5 \\ 0 & 0 & 2 & 2 & 4 & 4 & 6 & 6 \\ 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \end{bmatrix} \end{array}.$$

Из таблиц видно, что нулем алгебр является элемент $\sigma = 0$, а единицей – $\tau = 7$. В свою очередь, противоположные элементы алгебр перечислены в табл. П4.6. ♦

Таблица П4.6. Противоположные элементы при беззнаковом представлении

Элемент	0	1	2	3	4	5	6	7
R_A^{add}	0	7	6	5	4	3	2	1
R_A^{xor}	0	1	2	3	4	5	6	7

Знаковое представление чисел. В виду того, что в современных вычислительных средствах используется как беззнаковое, так и знаковое представление чисел, исследуем определенные ранее аддитивные алгебры в этом случае.

Известно, что для минимизации аппаратных затрат при кодирования чисел со знаком используется дополнительный код, позволяющий вместо вычитания ***sub*** использовать операцию сложения ***add***, а изменение знака числа выполнить через отрицание ***not*** и инкремент ***inc*** (увеличение числа на единицу):

$$\boldsymbol{sub}(a,b) = \boldsymbol{add}(a, \boldsymbol{neg}(b)) = \boldsymbol{add}(a, \boldsymbol{inc}(\boldsymbol{not}(b))) = \boldsymbol{add}(a, \boldsymbol{add}(1, \boldsymbol{not}(b))).$$

В итоге, операция ***sub*** реализуется тем же арифметико-логическим устройством, один из входов которого выполняет дополнительную микрооперацию инверсии разрядов, а на вход переноса в нулевой разряд подается 1.

Пример П4.9. Приведем таблицы операций для R_A^{add} и R_A^{xor}, реализуемых 3-разрядным арифметико-логическим устройством при кодировании чисел в дополнительном коде, $N_k = \{-4, -3, \ldots, 0, \ldots 3, 4\}$:

$$\boldsymbol{add} = \begin{array}{c|cccccccc} & -4 & -3 & -2 & -1 & 0 & 1 & 2 & 3 \\ -4 & 0 & 1 & 2 & 3 & -4 & -3 & -2 & -1 \\ -3 & 1 & 2 & 3 & -4 & -3 & -2 & -1 & 0 \\ -2 & 2 & 3 & -4 & -3 & -2 & -1 & 0 & 1 \\ -1 & 3 & -4 & -3 & -2 & -1 & 0 & 1 & 2 \\ 0 & -4 & -3 & -2 & -1 & 0 & 1 & 2 & 3 \\ 1 & -3 & -2 & -1 & 0 & 1 & 2 & 3 & -4 \\ 2 & -2 & -1 & 0 & 1 & 2 & 3 & -4 & -3 \\ 3 & -1 & 0 & 1 & 2 & 3 & -4 & -3 & -2 \end{array},$$

$$\boldsymbol{xor} = \begin{array}{c|cccccccc} & -4 & -3 & -2 & -1 & 0 & 1 & 2 & 3 \\ -4 & 0 & 1 & 2 & 3 & -4 & -3 & -2 & -1 \\ -3 & 1 & 0 & 3 & 2 & -3 & -4 & -1 & 0 \\ -2 & 2 & 3 & 0 & 1 & -2 & -1 & 0 & 1 \\ -1 & 3 & 2 & 1 & 0 & -1 & 0 & 1 & 2 \\ 0 & -4 & -3 & -2 & -1 & 0 & 1 & 2 & 3 \\ 1 & -3 & -4 & -1 & 0 & 1 & 0 & 3 & 2 \\ 2 & -2 & -1 & -4 & 1 & 2 & 3 & 0 & 1 \\ 3 & -1 & 0 & 1 & 2 & 3 & 2 & 1 & 0 \end{array},$$

$$\boldsymbol{and} = \begin{array}{c|cccccccc} & -4 & -3 & -2 & -1 & 0 & 1 & 2 & 3 \\ -4 & -4 & -4 & -4 & -4 & 0 & 0 & 0 & 0 \\ -3 & -4 & -3 & -4 & -3 & 0 & 1 & 0 & 1 \\ -2 & -4 & -4 & -2 & -2 & 0 & 0 & 2 & 2 \\ -1 & -4 & -3 & -2 & -1 & 0 & 1 & 2 & 3 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 2 & 0 & 0 & 2 & 2 & 0 & 0 & 2 & 2 \\ 3 & 0 & 1 & 2 & 3 & 0 & 1 & 2 & 3 \end{array}.$$

Находим, что при знаковом представлении нулем алгебр является элемент $\sigma = 0$, а единицей – $\tau = -1$. Противоположные элементы алгебр перечислены в табл. П4.7. ♦

Таблица П4.7. Противоположные элементы при знаковом представлении

Элемент	–4	–3	–2	–1	0	1	2	3
R_A^{add}	–4	3	2	1	0	–1	–2	–3
R_A^{xor}	–4	–3	–3	–1	0	1	2	3

В частности, из вышеизложенного следует, что спектральное представление Хаара существует только алгебре R_A^{add} при знаковом и беззнаковом кодировании чисел. При этом справедлива следующая рекуррентная формула:

$$\mathbf{D}_0 = \tau, \quad \mathbf{D}_{t+1} = \left(\mathbf{T}_{2^t} \otimes \begin{bmatrix} \tau & \tau \\ \tau & -\tau \end{bmatrix} \right) \times \left[\begin{array}{c|c} \mathbf{D}_t & \boldsymbol{\sigma}_{2^t} \\ \hline \boldsymbol{\sigma}_{2^t} & \mathbf{T}_{2^t} \end{array} \right] \ (t = \overline{0, n-1}), \quad \mathbf{D} = \mathbf{D}_{n-1}.$$

Приложение 5.
Факторизация спектральных базисов

Обычно под факторизацией понимается представление системы спектральных функций, заданных матрицей прямого (обратного) преобразования в виде произведения слабо заполненных матриц меньшего порядка. Факторизация необходима для выполнения быстрых спектральных преобразований, заключающихся в умножении матрицы на вектор.

В результате факторизации получают процедуру быстрых преобразований, основанную на приведении подобных слагаемых на основе дистрибутивности умножения относительно сложения и исключении операций в тривиальных случаях, когда не выполняется умножение на ноль и единицу, сложение с нулем.

Уменьшение количества операций является первым и необходимым условием факторизации. Второе условие связано с представлением алгоритма преобразования, когда дополнительно требуется получение такого представления матрицы, при котором операции имеют регулярное расположение в структуре алгоритма. Последнее позволяет не только сократить время вычислений, но и упростить сам алгоритм, повысить его эффективность.

Таким образом, при традиционном подходе к факторизации ставится задача получения эффективных процедур спектральных преобразований. При этом эффективность вычисления самих функций (матриц) не принимается во внимание и предполагается, что функции (матрицы) уже заданы.

П5.1. Постановка задачи

Рассмотрим задачу полиномиальной факторизации, т.е. осуществим поиск представления матрицы дискретного преобразования в виде кронекеровского произведения матриц меньшей размерности. Суть подхода заключается в вычислении матриц и операций, относительно которых выполняются кронекеровские произведения, таких, что в результате факторизации может быть получена формула для системы спектральных функций и вычислена исходная матрица дискретного преобразования.

Пусть задана целочисленная матрица $\mathbf{D}$ размерности $m \times m$. Для факторизации $\mathbf{D}$ найдем такие матрицы $\mathbf{P}_j$ и $\mathbf{O}_t$ ($j = \overline{0, n-1}$, $t = \overline{0, n-2}$), для которых с точностью до расстановки скобок выполняется равенство

$$\mathbf{D} = \mathbf{P}_0 \otimes_0 \mathbf{P}_1 \otimes_1 \ldots \otimes_{n-2} \mathbf{P}_{n-1} = \bigotimes_{i=0}^{n-1} \mathbf{P}_i, \qquad (\text{П5.1})$$

где элементы и размерность $\mathbf{P}_j$ не превосходят некоторого числа k_p, а кронекеровские произведения $\otimes_t$ выполняются относительно бинарных операций, заданных матрицами $\mathbf{O}_t$, которые также имеют элементы и размерность, не превосходящие k_p.

Матрица $\mathbf{D}$ представляется в виде (П5.1), если спектральные функции, составляющие ее столбцы, являются полиномиальными. Аналитическая конструкция полиномиальных функций, задаваемая с точностью до нумерации переменных и расстановки скобок, имеет вид:

$$\theta_i(X) = x_0^{i_0} \circ_0 x_1^{i_1} \circ_1 \ldots \circ_{n-2} x_{n-1}^{i_{n-1}} \qquad (\text{П5.2})$$

или, после подстановки в (П5.2) выражений $x_j^{i_j} = x_j \rhd_j i_j$,

$$\theta_i(X) = (x_0 \bullet_0 i_0) \circ_0 (x_1 \bullet_1 i_1) \circ_1 \ldots \circ_{n-2} (x_{n-1} \bullet_{n-1} i_{n-1}) \qquad (\text{П5.3})$$

где $X = \{x_0, x_1, \ldots, x_{n-1}\}$ – множество переменных со значностями k_0, k_1, …, k_{n-1}, соответственно, $m = k_0 k_1 \ldots k_{n-1}$ – длина вектора полиномиальной функции, i_j – цифры индекса функции i в позиционной системе счисления с основаниями, задаваемыми значностями переменных, $\bullet_j$ – бинарные степенные операции (определяются матрицами $\mathbf{P}_j$), $\circ_j$ – бинарные соединительные операции (определяются матрицами $\mathbf{O}_t$).

Таким образом, представление (П5.1) позволяет также получить и единую формулу для системы спектральных функций, которую легко отразить в структуру алгоритма вычисления матрицы $\mathbf{D}$ или отдельных ее элементов.

Поставим задачу поиска эффективных формульных представлений для упорядоченной системы функций, заданной в виде целочисленной матрицы. Если найдены операции полиномиальной аналитической конструкции (П5.3), которые имеют значность не более чем k_p, то факторизацию будем называть существующей или успешной, в противном случае – несуществующей. Число k_p назовем ***значностью полиномиальной факторизации***.

П5.2. Полиномиальная факторизация

Из формулы (П5.1) непосредственно следует рекуррентное правило для полиномиальной факторизации,

$$\mathbf{T}_0^* = \mathbf{D}, \quad \mathbf{T}_{t+1}^* = \mathbf{T}_t^L \otimes_t \mathbf{T}_t^R \ (t = 0, 1, \ldots), \quad \mathbf{P}_j = \mathbf{T}_k^*, \qquad (\text{П5.4})$$

где $\mathbf{T}^{*}_{t+1}$ – левая $\mathbf{T}^{L}_{t+1}$ или правая $\mathbf{T}^{R}_{t+1}$ промежуточная матрица, полученная на предыдущем шаге. Если размерность $\mathbf{T}^{*}_{k}$ не превосходит k_p, то матрица отождествляется с некоторой матрицей степенной операции $\mathbf{P}_j$.

Вычисления завершаются, когда отождествляются все промежуточные матрицы. Если $\mathbf{T}^{L}_{t+1}$ или $\mathbf{T}^{R}_{t+1}$ не отождествлена, то результат последующей факторизации этих матриц заключается в скобки. В свою очередь индексы степеней переменных i_j последовательно возрастают в порядке и в направлении вычислений.

Заметим, что без потери общности в рекуррентном правиле (П5.4) под $\otimes_t$ достаточно подразумевать только правое кронекеровское произведение.

П5.3. Полиномиальная декомпозиция

Базовой операцией в (П5.4) является представление матрицы $\mathbf{T}$ размерности $m' \times m''$ в виде кронекеровского произведения матриц $\mathbf{L}$ и $\mathbf{R}$ относительно неизвестной операции $\circ$,

$$\mathbf{T} = \mathbf{L} \otimes_{\circ} \mathbf{R}. \qquad (\text{П5.5})$$

Найдем для заданной матрицы $\mathbf{T}$ такие матрицы $\mathbf{L}$ и $\mathbf{R}$, а также операцию $\circ$, определяемую матрицей $\mathbf{O}$, которые обращают выражение (П5.5) в равенство. Для начала определим $\mathbf{L}$, $\mathbf{R}$ и $\mathbf{O}$ следующим образом:

$$\begin{cases} \mathbf{L} = [l_{uv}], & l_{uv} = v + uk''_l \quad (u = \overline{0, k'_l - 1}, \quad v = \overline{0, k''_l - 1}), \\ \mathbf{R} = [r_{sw}], & r_{sw} = w + sk''_r \quad (s = \overline{0, k'_r - 1}, \quad w = \overline{0, k''_r - 1}), \\ \mathbf{O} = [o_{pq}], & o_{pq} = p \circ q \quad (p = \overline{0, k'_l k''_l - 1}, \quad q = \overline{0, k'_r k''_r - 1}), \end{cases} \qquad (\text{П5.6})$$

где k'_l и k'_r – делители m', а k''_l и k''_r – делители m'', такие, что $m' = k'_l k'_r$ и $m'' = k''_l k''_r$.

Так как матрицы $\mathbf{L}$ и $\mathbf{R}$ заданы, нам остается найти только матрицу $\mathbf{O}$. Из выражений (П5.6) следует

$$t_{ij} = l_{i_l j_l} \circ r_{i_r j_r} = (j_l + i_l k''_l) \circ (j_r + i_r k''_r) = o_{pq}, \qquad (\text{П5.7})$$

где t_{ij} – элемент $\mathbf{T}$, $i = (i_l, i_r)_{k'_l k'_r}$ и $j = (j_l, j_r)_{k''_l k''_r}$ – представления индексов i и j в позиционной системе счисления с основаниями, задаваемыми делителями размерностей матрицы $\mathbf{T}$. Из (П5.7) получаем выражение для вычисления соединительной операции $\mathbf{O}$,

$$p \circ q = \mathbf{T}[(p'', q'')_{k''_l k''_r} \ (p', q')_{k'_l k'_r}], \qquad (\text{П5.8})$$

где $p = (p'', p')_{k''_l k'_l}$, $q = (q'', q')_{k''_r k'_r}$, $\mathbf{T}[j \ i]$ – элемент t_{ij} матрицы $\mathbf{T}$.

Формула (П5.8) позволяет вычислить матрицу $\mathbf{O}$. Для этого операнды p и q искомой операции $\circ$ представляются в позиционной системе счисления с основаниями, яв-

ляющимися делителями чисел m' и m''. Затем цифры p', p'', q' и q'' используются для вычисления индексов элемента матрицы $\mathbf{T}$ – результата операции $\circ$.

В итоге для произвольной матрицы $\mathbf{T}$ получаем полностью определенные матрицы $\mathbf{L}$, $\mathbf{R}$ и $\mathbf{O}$, связанные уравнением (П5.5). Очевидно, сделать это можно различными способами. Число решений уравнения (П5.5) равно количеству нетривиальных разбиений текущего множества переменных на два непересекающихся подмножества, произведение значностей k_l' и k_r' которых равно m', умноженное на количество представлений числа m'' в виде двух нетривиальных сомножителей k_l'' и k_r'', таких, что $k_l''k_r'' = m''$. Выбираем то решение, у которого матрица соединительной операции $\mathbf{O}$ может быть тождественно преобразована (редуцирована) до размеров, не превосходящих значность факторизации k_p.

П5.4. Полиномиальная редукция

Следующим шагом полиномиальной факторизации является редукция $\mathbf{O}$ и соответствующая модификация $\mathbf{L}$ и $\mathbf{R}$. Для этого строки и столбцы матрицы $\mathbf{O}$ разбиваются на классы эквивалентности. Сама матрица $\mathbf{O}$ преобразуется к виду, содержащему по одной строке (столбцу) их каждого класса.

Пусть $\{E_0, E_1, \ldots, E_{n_{r-1}}\}$ и $\{C_0, C_1, \ldots, C_{n_{c-1}}\}$ – классы эквивалентности строк и столбцов $\mathbf{O}$, состоящие из индексов одинаковых строк и столбцов (или сопоставимых строк и столбцов – в случае не полностью определенных матриц). Редуцированную матрицу строим следующим образом: в качестве i-ой строки новой матрицы используем строку из класса E_i, а вместо j-го столбца – столбец из класса C_j.

Для сохранения равенства (П5.5) все элементы $e \in E_i$ левой матрицы $\mathbf{L}$ должны быть заменены i – индексом класса эквивалентности, которому принадлежит элемент e. Аналогичным образом заменяются элементы правой матрицы. Все элементы $c \in C_j$ матрицы $\mathbf{R}$ заменяются j – индексом класса эквивалентности, в который входит элемент c.

В итоге получена редуцированная матрица $\mathbf{O}$ и модифицированные матрицы $\mathbf{L}$ и $\mathbf{R}$. Если размерность $\mathbf{O}$ не превосходит k_p по любому из измерений, то редукция считается успешной. В противном случае, выбираем другое решение уравнения (П5.5) и повторяем декомпозицию и редукцию.

П5.5. Демонстрационный пример

Пусть упорядоченная система функций задана матрицей

$$\mathbf{T}=\begin{bmatrix}0&1&1&2&0&1\\2&1&2&2&0&1\\2&0&2&1&0&0\\1&2&0&1&1&2\\2&2&2&1&2&2\\2&1&2&0&2&1\end{bmatrix}$$

размерности 6×6. Выберем значность факторизации $k_p=3$. Пусть $k_l'=3$, $k_r'=2$, $k_l''=2$ и $k_r''=3$. Тогда из (П5.6) и (П5.8) получим:

$$\mathbf{L}=\begin{bmatrix}0&1\\2&3\\4&5\end{bmatrix},\ \mathbf{O}=\begin{bmatrix}0&1&0&1&0&1\\1&2&1&2&1&2\\2&2&0&2&2&2\\1&2&1&2&1&2\\2&2&0&2&2&2\\0&1&0&1&0&1\end{bmatrix},\ \mathbf{R}=\begin{bmatrix}0&1&2\\3&4&5\end{bmatrix}.$$

В частности для элемента o_{53} имеем

$$o_{53}=5\circ 3=1\,2_{23}\circ 0\,1_{32}=\mathbf{T}[1\,0_{23}\,2\,1_{32}]=\mathbf{T}[1\ \ 5]=t_{51}=1.$$

Далее находим классы эквивалентности строк и столбцов:

$$E_0=\{0,5\},\ E_1=\{1,3\},\ E_2=\{2,4\};$$

$$C_0=\{0,4\},\ C_1=\{1,3,5\},\ C_2=\{2\}.$$

После построения редуцированной матрицы для соединительной операции и замены элементов левой и правой матриц на индексы классов, в которые они входят, получим

$$\mathbf{L}=\begin{bmatrix}0&1\\2&1\\2&0\end{bmatrix},\ \mathbf{O}=\begin{bmatrix}0&1&0\\1&2&1\\2&2&0\end{bmatrix},\ \mathbf{R}=\begin{bmatrix}0&1&2\\1&0&1\end{bmatrix}.$$

В итоге имеем $\mathbf{T}=\mathbf{L}\otimes_{\circ}\mathbf{R}$, где размерности матриц $\mathbf{L}$, $\mathbf{R}$ и $\mathbf{O}$ не больше чем k_p. Следовательно, система функций, заданная матрицей $\mathbf{T}$, может быть вычислена по формуле

$$\theta_i(x_0,x_1)=x_0^{i_0}\circ x_1^{i_1}=(x_0\bullet_L i_0)\circ(x_1\bullet_R i_1),$$

где значность переменной x_0 равна 3, x_1 – 2, индексы функций представляются в системе счисления с основаниями 2 и 3, а степенные и соединительная операции задаются редуцированными матрицами $\mathbf{L}$, $\mathbf{R}$ и $\mathbf{O}$.

Выполним проверку формулы путем вычисления θ_5 (последний столбец $\mathbf{T}$). Для этого представим индекс функции в системе счисления с основаниями 2 и 3, $5=12_{23}$. От-

куда находим степени переменных $i_0 = 1,\ i_1 = 2$. Конструируем итоговую формулу $\theta_5(x_0, x_1) = x_0^1 \circ x_1^2$ и выполняем ее минимизацию

$$\theta_5(x_0, x_1) = x_0^{\begin{bmatrix}1\\1\\0\end{bmatrix}} \circ x_1^{\begin{bmatrix}2\\1\end{bmatrix}} = x_0 \begin{bmatrix}1 & 2\\ 1 & 2\\ 0 & 1\end{bmatrix} x_1 = x_0 \triangleright x_1, \qquad (\text{П5.9})$$

где степенные операции преобразованы в унарные и заданы в виде векторов. В (П5.9) также выполнено преобразование бинарной операции $\circ$ и унарных операций $\neg_0$ и $\neg_1$ в эквивалентную им бинарную операцию $\triangleright$, благодаря чему имеем минимальную формулу функции. Вычисление этой функции по любой из полученных формул дает вектор $[1\,1\,0\,2\,2\,1]^T$, совпадающий с пятым столбцом матрицы **T**. ♦

Из примера видно, что полиномиальная факторизация может быть использована для получения формульных представлений системы спектральных функций при различной их упорядоченности, для минимизации или модификации операций, изменения значностей переменных, их порядка в выражении, и т.д. В этом случае по известной формуле для полиномиальных функций вычисляется, а затем факторизуется матрица дискретного преобразования **D**. В результате может быть получено другое формульное представление для той же системы функций, если, конечно, оно существует.

П5.6. Алгоритм факторизации

На рис. П5.1 приведен алгоритм факторизации системы спектральных функций, зависящих от переменных X и заданных квадратной матрицей **T**. Алгоритм реализует частный случай факторизации, когда степенные операции ищутся в виде квадратных матриц, т.е. когда значности переменных совпадают со значностью их степеней.

Функция Formula (строка 3) является рекурсивной и возвращает формулу для системы функций. Декомпозиция матрицы **T** осуществляется функцией Middle (строка 15), которая возвращает нередуцированную матрицу **O**. Функция Rows (строка 16) и Columns (строка 17) формируют вектора эквивалентности строк и столбцов следующим образом: i-ый элемент вектора описывает i-ую строку (столбец) и равен индексу класса эквивалентности, в который эта строка (столбец) входит. Такое представление позволяет по получаемым векторам **E** и **C** легко построить матрицы **L** и **R**, для чего используется функция Left (строка 19) и функция Right (строка 20). Если количество классов эквивалентности, возвращаемое функцией Classes (строка 18), не превосходит значности факторизации k_p, то производится редукция матрицы **O**, осуществляемая функцией Operation (строка 21).

```
1    scalar k_P
2    stack powers, connects
3    string function Formula(matrix T, set X)
4        scalar u, v
5        string F_l, F_r
6        matrix L, O, R
7        vector E, C
8        if Count(X) = 1
9            u := X(1)
10           v := Push(powers, T)
11           return "(" + "x_u" + "▷_v" + "i_v"+ ")"
12       else
13           First(X)
14           do
15               O := Middle(T, X)
16               E := Rows(O)
17               C := Columns(O)
18               if Classes(E) <= k_P and Classes(C) <= k_P
19                   L := Left(E)
20                   R := Right(C)
21                   O := Operation(O, E, C)
22                   F_l := Formula(L, Implement(X))
23                   F_r := Formula(R, Complement(X))
24                   if F_l <> empty and F_r <> empty
25                       u := Push(connects, O)
26                       return "(" + F_l + "∘_u" + F_r + ")"
27                   else
28                       return empty
29                   end if
30               end if
31           while Next(X)
32           return empty
33       end if
34   end function
```

Рис. П5.1. Алгоритм факторизации матриц

Для работы с наборами переменных используются функции First (строка 11) и Next (строка 31), которые формируют начальное и следующее подмножества переменных, причем последняя функция возвращает логический ноль, если все подмножества просмотрены, в противном случае – логическую единицу. Функции Implement (строка 22) и Complement (строка 23) формируют соответственно текущий набор переменных и набор переменных, не вошедших в текущий; а функция Count (строка 8) служит для получения количества переменных в наборе.

Результатом работы алгоритма является формула, представляющая собой строку, состоящую из имен переменных, знаков операции и скобок, задающих порядок вычисления (строки 11, 26). Сами операции получаются в виде матриц и сохраняются в стеке операций: степенные – в стеке *powers* (строка 10), соединительные – в стеке *connects* (строка 25). Если формулы не существует, то результатом работы алгоритма будет пустая строка и незаполненные стеки операций.

Представленный алгоритм основан на единственности полиномиальной декомпозиции матрицы. Оператор return (строки 26, 28) выполняет выход из цикла по подмножествам переменных X и из рекурсии по глубине формулы, если матрица соединительной операции **O** имеет успешную редукцию, причем независимо от того, факторизовались ли матрицы **L** и **R** или нет.

П5.7. Неполиномиальная факторизация

Представленный выше алгоритм может быть использован для неполиномиальной факторизации матриц, эквивалентной бесповторной декомпозиции функции. Бесповторной называется такая формула, в которую переменные входят не более одного раза.

Так как система спектральных функций в рассматриваемом случае состоит из одной функции, то матрица дискретного преобразования **D** вырождается в вектор-столбец и, как следствие этого, промежуточные матрицы **L** и **R** ищутся в виде векторов. Очевидно, что вычисляемые при этом степенные операции будут унарными.

Декомпозиция завершается, когда длина текущего вектора **T** совпадает со значностью переменной, оставшейся свободной. Если факторизация успешная, то может быть получена минимальная формула функции

$$f(X) = x_{t_1} \triangleright_1 x_{t_2} \triangleright_2 \ldots \triangleright_{s-1} x_{t_s},$$

представленная с точностью до расстановки скобок, и из которой исключены унарные степенные операции и несущественные переменные. Если факторизация не удалась, то декомпозируемая функция не имеет неполиномиального (бесповторного) представления при заданной значности факторизации.

П5.8. Вычислительный эксперимент

На рис. П5.2 представлены результаты экспериментального исследования алгоритма полиномиальной факторизации, где показана зависимость времени факторизации t от размерности квадратной матрицы m при неуспешной (A) и успешной (B) факторизации. Эксперимент проводился на множестве трехзначных функций при значности факторизации, равной 3. Из рисунка следует, что время факторизации оценивается как $t \sim m^{2,2\ldots3,6}$.

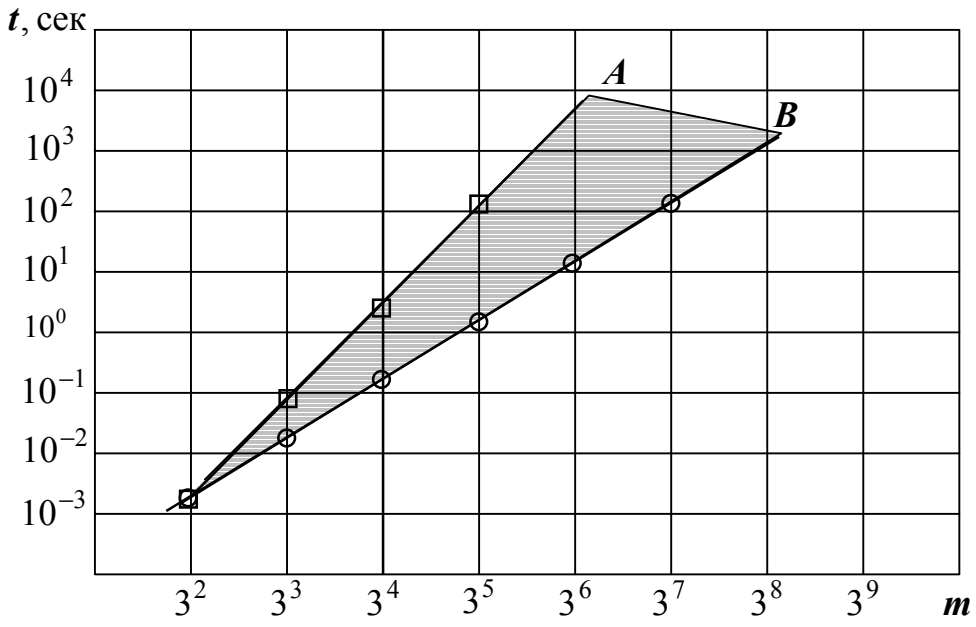

Рис. П5.2. Зависимость времени факторизации от размерности матрицы при неуспешной (A) и успешной (B) факторизации

П5.9. Сжатие данных

Приведем еще одно применение полиномиальной факторизации в области сжатия изображений. Известно, что при цифровой обработке изображений имеют дело с массивами (матрицами) данных, составные части которых сильно коррелированны между собой, а сами данные кодируются целыми числами. Традиционно ставится задача сжатия изображения с минимальными потерями качества при восстановлении.

Как показано ранее, при полиномиальной факторизации на каждом шаге производится разбиение исходной матрицы на части, причем число делений по горизонтали и по вертикали является изменяемым и определяется возможностью представления матрицы в виде совокупности одинаковых или близких друг к другу с заданной точностью областей. Если количество классов эквивалентности областей не превосходит некоторого числа, определяющего степень сжатия, то формируется соединительная операция (матрица небольшой размерности), а также правая и левые матрицы, которые подлежат дальнейшей факторизации. Если достигнута небольшая или приемлемая по другим соображениям размерность этих матриц, то процесс полиномиальной факторизации завершается.

В итоге, исходное изображение $m \times m$ точек заменяется на $2n-1$ матриц размерности $k_p \times k_p$, где $k_p \approx \sqrt[n]{m}$ – степень сжатия изображения; а объем памяти, необходимый для хранения результата преобразования, сокращается примерно в $2n$ раз.

Понятно, что при жестком задании степени сжатия факторизация может оказаться невыполнимой. В этом случае возможна модификация алгоритма, при которой на каждом шаге выбирается своя, присущая только этому шагу, степень сжатия. Тем самым обеспе-

чивается безусловность получения положительного результата и адаптивность алгоритма сжатия изображения.

Особенностью описанного алгоритма является возможность изменения масштаба сопоставляемых областей, вплоть до нескольких точек. Другая особенность алгоритма – понижение значности элементов изображения с каждым шагом декомпозиции. На первом шаге соединительная матрица содержит все отличные друг от друга отсчеты сигнала. После редукции значность элементов левой и правой матрицы понижается до соответствующих размерностей редуцированной матрицы, т.е. с каждым шагом факторизации динамический диапазон сигнала уменьшается примерно в $\sqrt{d}$ раз, где d – начальное его значение.