Глава 3.
Контекстная технология

Настоящая глава посвящена контекстной технологии обработки данных, которая появилась как объединение объектно-ориентированной и фортоподобной технологий [11, 248], осуществленное на основе понятийного анализа предметной области, контекстной интерпретации текстов программ [48] и определения семантики создаваемого в процессе решения задачи специализированного предметного языка, предназначенного для описания этого решения [64].

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 преодоление 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) специализированный предметный язык.

Для описания семантики специализированного 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:

– выделение проблемной области и некоторой активной проблематики;

– понятийный анализ предметной области и определение ее понятийной структуры;

– описание специализированного предметного языка путем создания понятийной модели предметной области;

– ситуационное описание предметной 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 специализированного предметного языка. Выявленные в процессе понятийного анализа множество понятий и декомпозиционные схемы кладутся в основу создаваемого языка: понятия предметной 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 [218], 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 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).

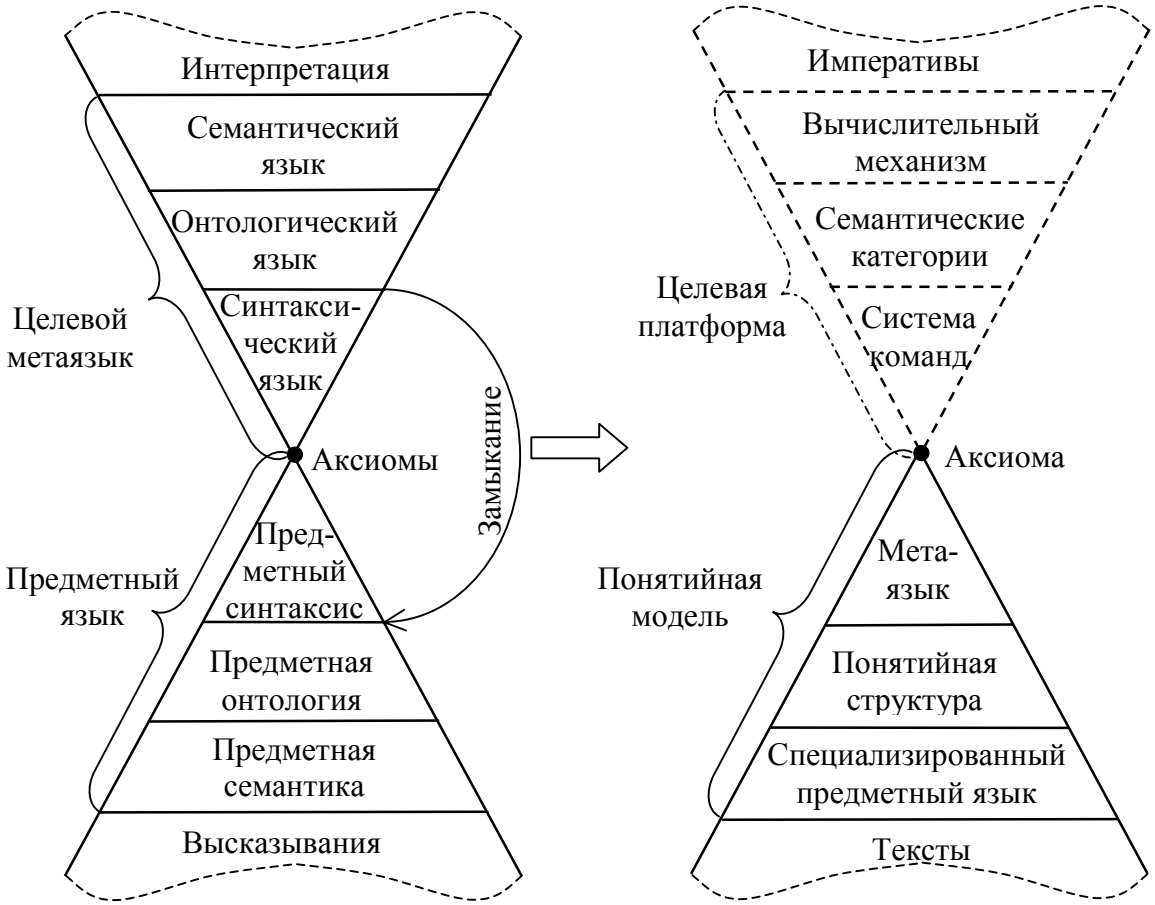

Рис. 3.12. Семантическое замыкание

Для описания предметного языка будем использовать 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, выполняет роль своего метаязыка. Так как внутри замкнутой семиотической системы семантику определить нельзя, воспользуемся внешним семантическим интерпретатором – целевой вычислительной платформой, состоящей, как и 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, метаязыка, понятийной структуры и некоторого специализированного языка.

Таким образом, для каждого класса задач и в процессе их решения будем создавать свой, присущий только этим задачам специализированный предметный язык, отражающий понятийную структуру предметной 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 выполним на метаязыке, который имеет фиксированные синтаксис и семантику, достаточную для определения синтаксиса специализированного языка и пополнения множества базовых семантических категорий. Для описания семантики специализированного языка применим метод семантической индукции, заключающийся в использовании семантических категорий, которые определяются по мере необходимости, в процессе описания синтаксиса специализированного языка и средствами этого языка. Базу индукции, или первичные семантические категории, объявим с помощью аксиомы и реализуем средствами целевой вычислительной платформы.

На содержательном уровне семантическое замыкание заключается в использовании создаваемого предметного 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*** предметной области будем понимать данные, отражающие накопленные знания об этой предметной области и обладающие синтаксической и семантической полнотой.

Для формализации знаний о предметной 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 предметной 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 [39], 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 [29], в контекстной технологии программа представляет собой определение некоторого объема знаний – в виде понятийной модели предметной области, и описания искомого решения – в рамках определенных знаний.

Контекстная технология близка технологии создания экспертных систем [91], где экспертная система представляется как база знаний, содержащая структуру и факты о предметной 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 метаязыке, а описание семантики понятий и решаемых задач выполняется на определяемом в процессе программирования специализированном 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 предметного 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 предметного языка осуществляется на языке, поддерживаемом целевой системой программирования.

3.2.3. Метаязык

Метаязыком называется 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 [21].

Ограничим роль метаязыка контекстной технологии описанием только структурных свойств предметного языка, как это делается при описании языков программирования. Среди используемых для этого метаязыков наиболее распространенными являются металингвистическая форма Бэкуса-Наура (БНФ) [78], которая была разработана для описания языка ALGOL60 и стала математическим фольклором, и синтаксические диаграммы Вирта [38].

В метаязык контекстной технологии включим средства для выражения наиболее устойчивых концепций, лежащих в основе как понятийного анализа, так и контекстной технологии. Метаязык будем использовать для описания понятийной структуры предметной области и синтаксиса правил выражения понятий в тексте программы. Семантиче-

скую роль метаязыка ограничим реализацией аксиомы, позволяющей ввести в использование первичные семантические категории.

Лексический, синтаксический и семантический анализ, а также компиляцию части модели, написанной на метаязыке, реализуем известными методами. Однако, для части модели, предназначенной для описания семантики на определяемом предметном 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, метаязык является подразумеваемой частью любой понятийной модели. Однако, в отличие от базовых примитивов, которые декларируются перед использованием с помощью аксиомы, метаязык по своей природе является предопределенной частью любой модели.

Подробное описание принципов построения и одной из возможных реализаций метаязыка 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*** [118]. 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 [174].

Семантику понятийной модели будем задавать в виде текста, который построен по правилам определяемого в модели предметного 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 система 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 специализированного 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 предметном 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 специализированного языка.

Известные методы описания семантики, например, используемые при автоматическом конструировании компиляторов [218], 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.

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. Следовательно, специализированный предметный 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 специализированном 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 специализированного 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 предметной 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 со специализированным предметным 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 метаязыке, а в ***фазе компиляции*** – на определенном языке. В ***фазе выполнения*** исполняется код, порожденный при компиляции текста, выраженного на специализированном предметном языке.

Более подробное описание одной из реализаций системы контекстной обработки данных приведено в Главе 4.

3.3. Метаязык понятийной модели

Рассмотрим метаязык контекстной технологии, предназначенный для определения специализированных предметных языков, являющихся формализованным представлением

результатов понятийного анализа предметной области и служащих для решения определенного класса прикладных задач.

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 [193, 79]. 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.

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 предложени-

ем, 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.

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

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 [349], где регулярное выражение представлено в виде терминального шаблона и, для отличия от терма, заключено в двойные кавычки. 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 метаязыка

Метаязык контекстной технологии определим как язык для выражения суждений о специализированном предметной языке, создаваемом на основе понятийного анализа предметной области и служащим для решения одной или нескольких прикладных задач. Метаязык зададим грамматикой на рис. 3.14, представленной металингвистическими формулами Бэкуса-Наура.

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 '}'

Рис. 3.14. Порождающая грамматика метаязыка

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 зна-

ков, 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.

3.3.3. Понятийная структура

Понятийная модель состоит из описаний сущностей предметной области essences (правило 2). Каждой сущности присваивается имя notion нетерминального понятия определяемого языка, а само понятие задается как находящееся в отношениях дифференциации 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 предметной 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 поня-

тием определяемого языка. Для выражения лексем могут использоваться как термы term, так и множества термов, задаваемые на языке регулярных выражений [349] 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 метаязыке 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, указание на проверку более сложных контекстных условий, чем это задается средствами метаязыка. 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,[48] 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.

---

[48] 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. 179).

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.

```
(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 [25, 189]. Синтаксические правила, которые не описываются контекстно-свободными грамматиками, называются ***контекстными условиями***.

Контекстные условия в метаязыке задаются для имен понятий notion, имен аспектов aspect и лексем lexeme. На рис. 3.14 элементы определения контекстных условий для перечисленных нетерминальных понятий метаязыка выделены курсивом: это ***notion***, ***aspect*** и ***terms***.

Задание имени понятия ***notion*** как терминального понятия метаязыка осуществляется при описании сущности essences (правило 3). После такого определения имена поня-

тий 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 метаязыке 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 метаязыка и одновременно нетерминальными понятиями определяемого языка. Имена аспектов aspect, как и имена понятий notion являются терминальными понятиями для метаязыка, 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 предметной 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 специализированном языке, для чего используются ранее определенные семантические единицы (индуктивный семантический переход).

Для обеспечения полноты и целостности понятийной модели во всех ее частях специализированный 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.

Семантики 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 [99]. 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 предметном 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 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 предметного языка

В контекстной технологии решение прикладных задач осуществляется на основе создания специализированного предметного языка, что требует использования развитых средств для описания его синтаксиса и семантики.

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 семантика 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 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 метаязыке контекстной технологии специальных выразительных средств для декларации базовых примитивов (первичных семантических категорий), непосредственно реализуемых целевой вычислительной системой. 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.

3.4.1. 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 специализированного языка и средствами самого языка, т.е. специализированный 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 метаязыковыми средствами перед использованием.

Предположением индукции служат все ранее определенные семантические категории. Семантическая категория соответствует определяемому в модели понятию и выражается 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 специализированного подязыкам.

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 языков программирования, покажем реализацию известных способов описания семантики, таких, например, как синтаксически-управляемые схемы и атрибутные грамматики.

3.4.2. Синтаксически-управляемые схемы

Синтаксически-управляемые схемы [10] и атрибутные грамматики [118] 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).

```
() 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

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. ♦

3.4.3. 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 выяснилось, 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 [15]. 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 [195].

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).

```
(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 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.

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 предметном 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 [195].

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 [48] высказана идея об определении 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 [51] показана возможность использования самоопределения языка при создании программ, описаны общие принципы контекстной обработки данных и приведен первый вариант формальной грамматики метаязыка. Вопросы, связанные с выразительными качествами метаязыка, обсуждаются в работе [59].

В работе [105] затронут вопрос о возможности универсального описания семантики формальных языков. Общим вопросам представления знаний на основе понятийных структур посвящена работа [63], а в работе [61] рассмотрены вопросы создания, хранения и использования знаний, представленных в виде откомпилированных понятийных моделей. Необходимость реализации семантической замкнутости при представлении знаний обосновывается в работе [62].

Работы [101, 102, 103] посвящены прикладным вопросам контекстной технологии, а в работе [104] рассмотрено ее применение при математическом моделировании. В работе [106] описана понятийная структура более сложной предметной области, чем это показано в примерах.

Подходы к представлению понятийных моделей различных предметных областей рассмотрены в работах [106, 62]. В работе [60] 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 метаязык 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 специализированном предметном языке и предусматривающий выразительные средства для задания необходимых семантических категорий по мере необходимости, в процессе описания конструкций определяемого 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 специализированном языке. Определение семантики специализированного 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 [57, 65].

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 ***специализированный язык***, позволяющий адекватным образом описать как множество известных фактов относительно предметной области, так и решение или свойства решения стоящих прикладных задач.

Построение специализированного 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 специализированного 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 специализированном предметном языке. Последнее есть следствие приближения специализированного 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 [32]. 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 [218].

В свою очередь синтаксический анализ предназначен для получения структуры текста, где под структурой понимается дерево грамматического разбора [193]. 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, 114]. 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 метаязыка 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.

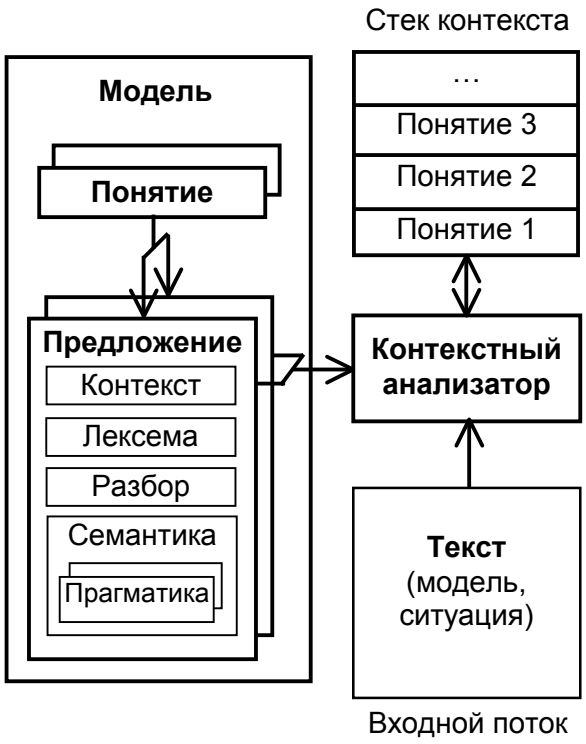

Рис. 4.2. Контекстное сопоставление

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. ♦

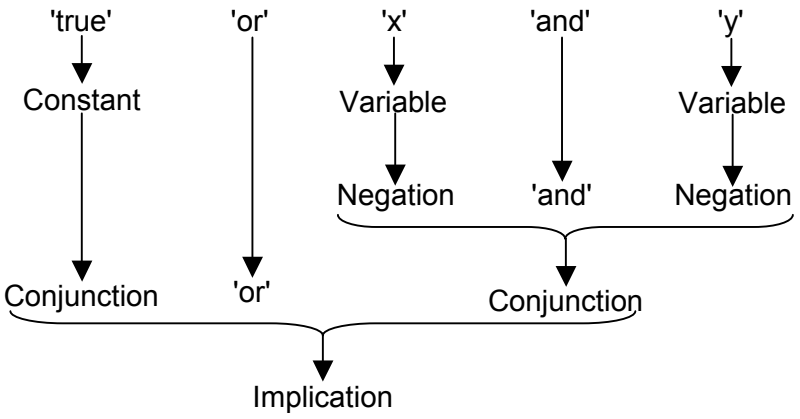

Рис. 4.5. Схема грамматического разбора

4.2.6. Конверторы

Метаязык 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 [78].

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 специализированного 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. 2.5.6 на с. 116), 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 специализированных предметных 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 специализированных 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.

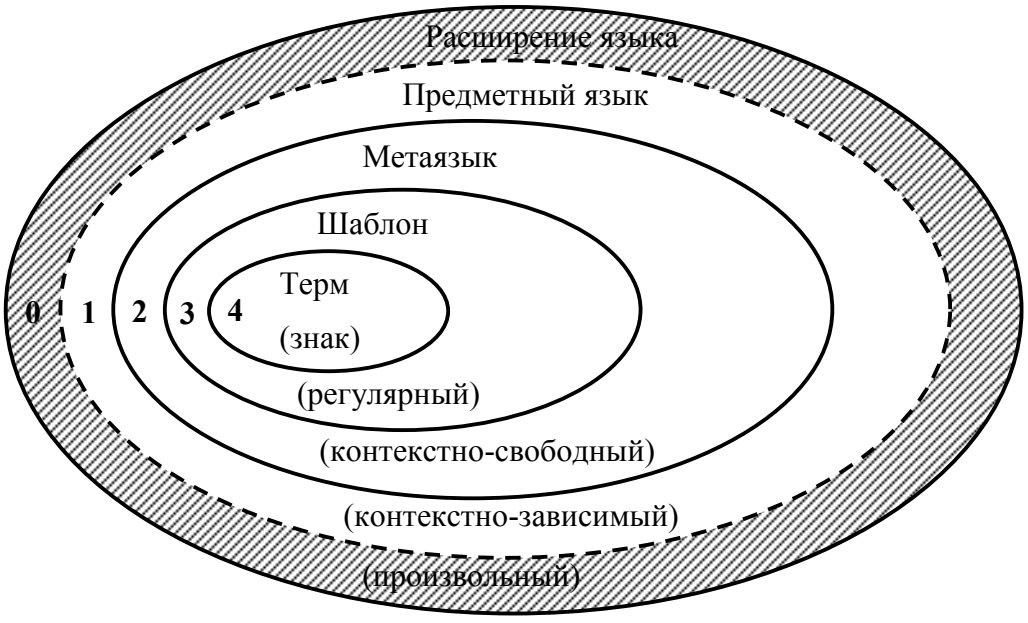

Рис. 4.6. Иерархия языков

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 [251]. На втором уровне находится метаязык, являющийся контекстно-свободным. Самые высокие выразительные способности имеет специализированный

предметный 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 [323, 148]:

– 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 [282].

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) [43].

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 [219]. В частности, система LINGOL [307] 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 метаязыка 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 [288]. Считается, что анализ текста с использованием категориальных грамматик является альтернативным порождающему подходу, предложенного Хомским [251].

В категориальных грамматиках, являющихся по своим выразительным качествам эквивалентными контекстно-свободным грамматикам [79], 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 ($B \setminus A$), то конкатенация строк α и β (β и α) будет принадлежать к категории A.

Основные законы грамматической композиции имеют форму правила отделения Modus Ponens [299]:

$$A/B, B \mapsto A, \quad B, B \setminus A \mapsto A$$

или

$$\frac{\Gamma \mapsto A/B, \Delta \mapsto B}{\Gamma, \Delta \mapsto A}, \quad \frac{\Gamma \mapsto B, \Delta \mapsto B \setminus A}{\Gamma, \Delta \mapsto A},$$

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 $B \setminus A$ 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 [231].

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 [277];

– расширения логических грамматик [243];

– расширения ATN-грамматик [347].

Например, при автоматизированного построения синтаксических анализаторов в среде ANTLR (ANother Tool for Language Recognition) [302] 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 метаязыка, 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 метаязыка 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 метаязыка вплоть до учета контекстной зависимости его правил.

4.3.5. Трансформационные грамматики

Наиболее мощными из известных являются системы, построенные на теории трансформационных грамматик [252]. 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. Хотя применение трансформаций относится к небольшому числу лингвистических фено-

менов, 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 [281], так же как и LINGOL [307], 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 [271], 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 [176], реализован в системе TULIPS [151], 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 [196], 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 [283].

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 [230].

Многие унификационные формализмы в различной степени дополнены механизмом логического программирования, основанного на ограничениях [172]. 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 [22] (1973 г.), и независимо от них А. Кольмероэ [256] (1975 г.) – разработчик языка Пролог. Однако, ближайшими предшественниками этого класса были атрибутные грамматики Д. Кнута [285] (1968 г.) и грамматики А. Ван Вейнгаардена, использованные для определения языка Алгол-68 [173] (1969 г.).

Одним из известных типов логических грамматик являются грамматики DCG (Definite Clause Grammars), разработанные Ф. Перейрой и Д. Уорреном [303] (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 [272].

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 [315]. 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) [190].

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 специализированного предметного 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 276.

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 [206], 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 метаязыка 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 [226]:

```
Boolean 'imp' Boolean
    [ asm{ pop eax; not eax; pop edx; or eax, edx; push eax } ]
```

Однако, ранее семантика этого предложения определена средствами специализированного 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 [193], 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).

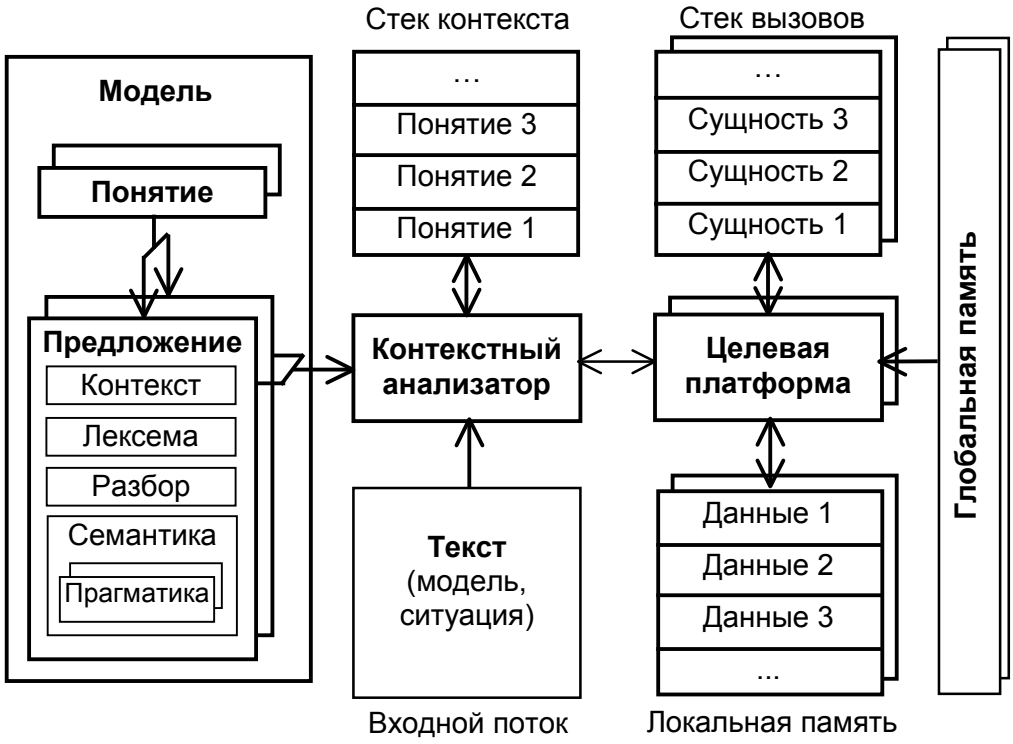

Рис. 4.9. Архитектура системы контекстного программирования

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 [202] 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.

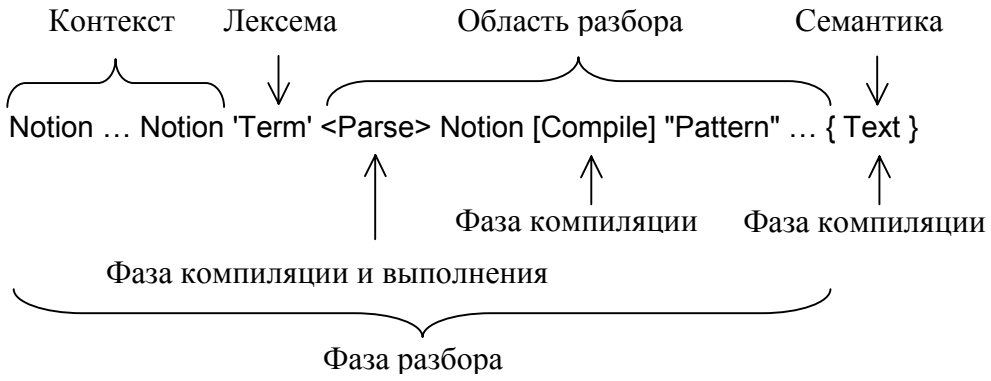

Рис. 4.10. Фазы работы системы программирования

В ***фазе разбора*** выполняется грамматический разбор текста (лексический, синтаксический и семантический анализ), заданного на метаязыке 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 метаязыка 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 специализированном 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 метаязыка, или, что тоже самое, реализуется проверка контекстных условий определяемого специализированного языка.

В итоге получаем следующий важный вывод: в процессе определения специализированного 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.

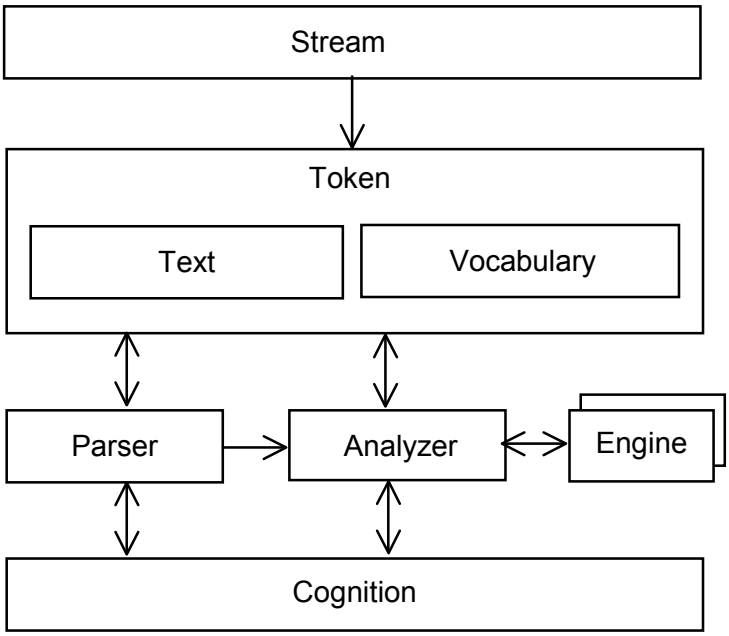

Рис. 4.11. Структура системы контекстного программирования

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 мета-

языка. 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.

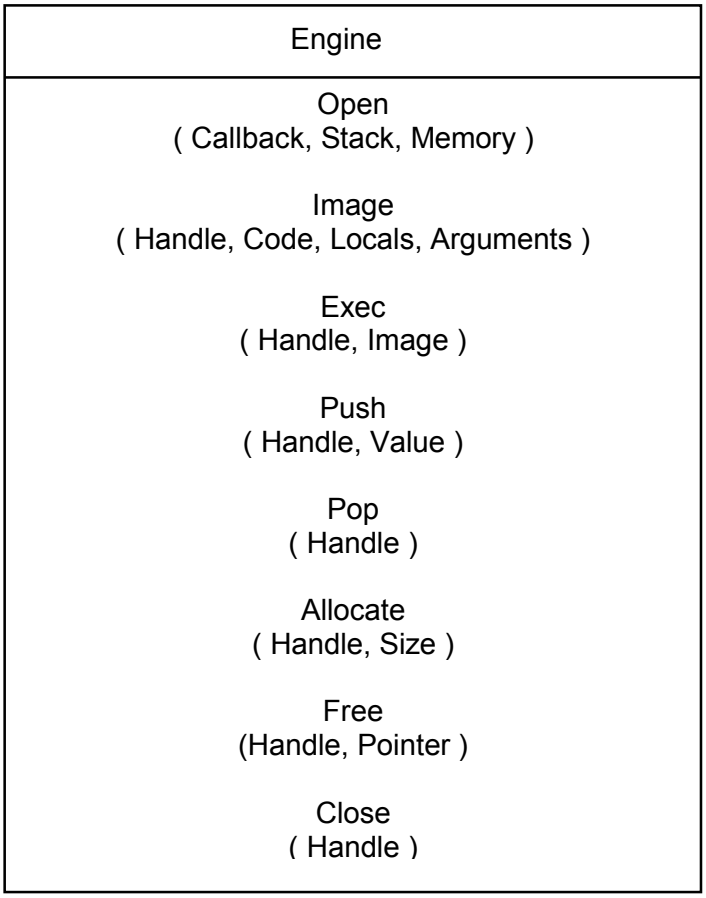

Рис. 4.12. Интерфейс виртуальной машины

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.

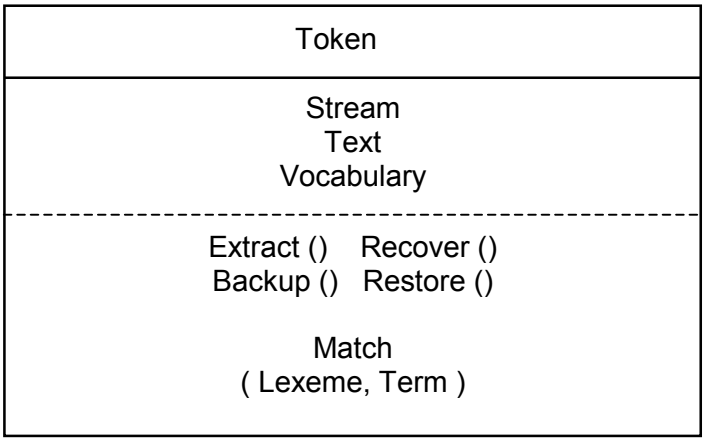

Рис. 4.13. Лексический анализатор

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 метаязыка 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.

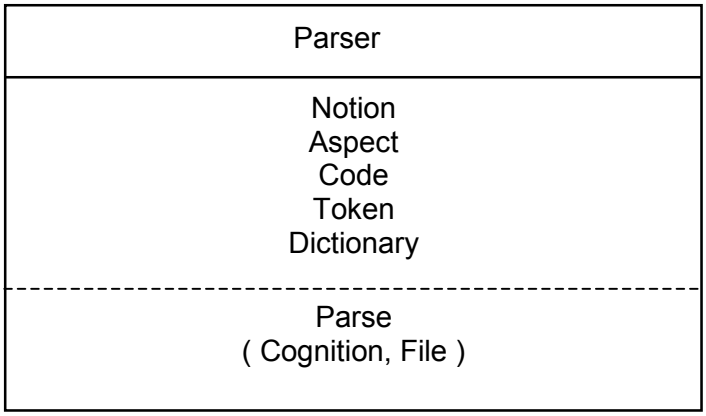

Рис. 4.14. Синтаксический анализатор

4.5.5. Семантический анализатор

Синтаксис входного текста, с одной стороны, частично определяется грамматикой метаязыка 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.

Специализированный предметный 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 метаязыка, требующие семантического анализа

Очевидно, обработку конструкции text необходимо осуществлять иначе, чем текст, описываемый метаязыком, 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.

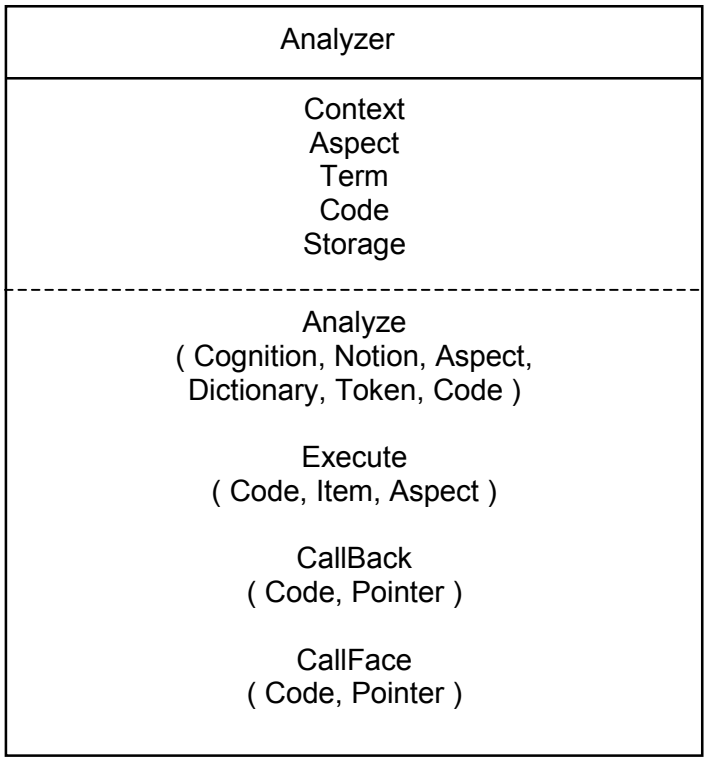

Рис. 4.16. Семантический анализатор

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[49];

– ***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.);

[49] Перенаправление входного потока, осуществляемое таким образом, что анализируемым текстом становится некоторая произвольная строка, позволяет, в частности, реализовать метод трансформационных грамматик [252], 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 метаязыка контекстной технологии (рис. 4.17). По сравнению с ранее определенными конструкциями метаязыка, 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 [125]. 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 [195].

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 [195]. Семантика, описываемая при задании правил вывода грамматики языка, называется ***статической***, а семантика получаемой в результате компиляции программы последовательности команд – ***динамической***.

Однако, разделение семантики специализированного предметного 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 специализированном предметной 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 метаязыке 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 специализированном 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 специализированного предметного языка, создаваемого в процессе декомпозиции предметной области.

В заключение укажем, что понятийная модель, представленная на метаязыке 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 специализированного предметного 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 специализированном предметном 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.

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. *Барендрегт Х.* Ламбда-исчисление, его синтаксис и семантика. М.: Мир, 1985.

14. *Башмаков А. И., Башмаков И. А.* Интеллектуальные информационные системы: Учеб. пособие. М.: Изд-во МГТУ им Н.Э. Баумана, 2005.

15. *Бездушний А. Н., Лютый В. Г., Серебряков В. А.* Разработка компиляторов в системе СУПЕР. М.: ВЦ АН СССР, 1991.

16. Бениаминов Е. М. Алгебраические методы в теории баз данных и представлении знаний. М.: Научный мир, 2003.

17. *Бердяев Н.А.* Творчество и объективация. М.: Экономпресс, 2000.

18. *Бибило П. Н.* Декомпозиция булевых функций (обзор) // В кн. Проектирование устройств логического управления. М.: Наука, 1985. С. 106-126.

19. *Блох А. Ш.* Канонический метод синтеза контактных схем // АиТ. 1961. № 6.

20. Большой психологический словарь / Сост. Мещеряков Б., Зинченко В. М.: Олма-пресс. 2004.

21. Большой энциклопедический словарь. М., 1998.

22. *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.

23. *Борщев В. Б.* Естественный язык – наивная математика для описания наивной картины мира // Московский лингвистический альманах. 1996. № 1. С. 203-225.

24. *Братко И.* Алгоритмы искусственного интеллекта на языке PROLOG. М.: Вильямс, 2004.

25. *Братчиков И. Л.* Синтаксис языков программирования. М.: Наука, 1975.

26. *Бронштейн И. Н., Семендяев К. А.* Справочник по математике для инженеров и учащихся втузов. М.: Наука, 1980.

27. *Булос Дж., Джеффри Р.* Вычислимость и логика. М.: Мир, 1994.

28. *Бурбаки Н.* Теория множеств. Структуры. М.: Мир, 1965.

29. *Буч Г.* Объектно-ориентированное проектирование с примерами применения. М.: Конкорд, 1992.

30. *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.

31. *Вагин В. Н., Головина Е. Ю., Загорянская А. А. Фомина М. В.* Достоверный и правдоподобный вывод в интеллектуальных системах. М.: Физматлит, 2004.

32. *Вайнгартен Ф.* Трансляция языков программирования. М.: Мир, 1977.

33. *Варламов О. О.* Эволюционные базы данных и знаний для адаптивного синтеза интеллектуальных систем. Миварное информационное пространство. М.: Радио и связь, 2002.

34. *Васильев С. Н., Жерлов А. К., Федосов Е. А., Федунов Б. Е.* Интеллектное управление динамическими системами. М.: Физико-математическая литература, 2000.

35. *Васюков В. Л.* Формальная феноменология. М.: Наука, 1999.

36. *Величковский Б. М.* Когнитивная наука: Основы психологии познания. В 2-х томах. М.: Академия, 2006.

37. *Виленкин Н.Я.* Класс полностью ортогональных систем // Известия Академии наук СССР. 1947. Сер. Математика. № 11. С. 363-400.

38. *Вирт Н.* Систематическое программирование. Введение. М.: Мир, 1977.

39. *Вирт Н.* Алгоритмы и структуры данных. М.: Мир, 1989.

40. *Виттих В. А.* Интеграция знаний при исследованиях сложных систем на основе инженерных теорий // Известия РАН. Теория и системы управления. 1998. № 5.

41. *Вольфенгаген В .Э.* Комбинаторная логика в программировании. Вычисления с объектами в примерах и задачах. М.: Центр ЮрИнфоР, 2003.

42. *Вольфенгаген В. Э.* Методы и средства вычислений с объектами. Аппликативные вычислительные системы. М.: Центр ЮрИнфоР, 2004.

43. *Вудс В. А.* Сетевые грамматики для анализа естественных языков // Кибернетический сборник. Вып. 13. М.: Мир, 1978. С. 120-158.

44. *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.

45. *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.

46. *Выхованец В. С.* Обобщенные полиномиальные формы // Радиоэлектроника. Информатика. Управление. 1999. № 2. С. 55-59.

47. *Выхованец В. С.* Булевы мультипликативные формы // Тезисы докладов международной научно-практической конференции «Математические методы в образовании, науке и промышленности». Тирасполь, 1999. С. 52-53.

48. *Выхованец В. С.* Контекстная технология программирования // Труды IV Международной научно-технической конференции по телекоммуникациям (Телеком-99). Одесса, 1999. С. 116-119.

49. *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.

50. *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.

51. *Выхованец В. С.* Язык контекстного программирования // Тезисы докладов 8-й Международной конференции «Проблемы управления безопасностью сложных систем». М., 2000. Т. 2. С. 89-91.

52. *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.

53. *Выхованец В.С.* Спектральные методы в логической обработке данных // АиТ. 2001. № 10. С. 28-53.

54. *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.

55. *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.

56. *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.

57. *Выхованец В. С.* Разнесенный грамматический разбор // Проблемы управления. 2006. № 1. С. 32-43.

58. *Выхованец В. С.* Алгебраическая декомпозиция дискретных функций // Автоматика и телемеханика. 2006. № 3. С. 20-56.

59. *Выхованец В. С.* Оптимальный синтез логического управления // Тезисы докладов 3-ой Международной конференции по проблемам управления. М., 2006. Т. 2. С. 105.

60. *Выхованец В. С., Гордиенко К. П.* Процессор с сокращенным набором команд // Вестник Приднестровского университета. Тирасполь, 1995. № 1. С. 124-128.

61. *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.

62. *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.

63. *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.

64. *Выхованец В. С., Иосенкин В. Я.* Понятийный анализ и контекстная технология программирования // Проблемы управления. 2004. №.4. С. 3-25.

65. *Выхованец В. С., Иосенкин В. Я.* Понятийный анализ и контекстная технология программирования // Проблемы управления. 2005. № 4. С. 2-11.

66. *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.

67. *Выхованец В. С., Малюгин В. Д.* Кратные логические вычисления // Автоматика и телемеханика. 1998. № 6. С. 163-171.

68. *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.

69. *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.

70. *Выхованец В. С., Малюгин В. Д.* Мультипликативная алгебра и ее применение в логической обработке данных // Проблемы управления. 2004. № 3. С. 67-77.

71. *Жегалкин И.И.* О технике вычисления предложений в символической логике // Математический сборник. 1927. Т. 43. С. 9-28.

72. *Замулин А. В.* Алгебраическая семантика императивного языка программирования // Программирование. 2003. № 6. С. 51-64.

73. *Замулин А. В.* Абстрактная модель компилятора как результат алгебраической семантики языка программирования. //Программирование. 2004. № 5. С. 69-80.

74. *Гаврилов М. А.* Релейно-контактные схемы с вентильными элементами // Изв. АН СССР. ОТН. 1945. № 3. С. 153-164.

75. *Гаврилов М. А.* Методы синтеза релейно-контактных схем. // Электричество. 1946. № 2. С. 54-59.

76. *Гаврилова Т. А., Хорошевский В. Ф.* Базы знаний интеллектуальных систем. СПб.: Питер, 2000.

77. *Гаскаров Д. В.* Интеллектуальные информационные системы. М.: Высш. шк., 2003.

78. *Гинзбург С.* Математическая теория контекстно-свободных языков. М.: Мир, 1970.

79. *Гладкий А.В.* Формальные грамматики и языки. М.: Наука, 1973.

80. *Голдблатт Р.* Топосы. Категорный анализ логики. М.: Мир, 1983.

81. *Горбатов В.А.* Синтез логических схем в произвольном базисе // В кн. Теория дискретных автоматов. Рига: Зинатне, 1967.

82. *Горовой В.Р.* Синтез релейных структур методом замены выходных функций // АиТ. 1967. № 1. С. 112-121.

83. *Горский Д. П.* Вопросы абстракции и образования понятий. М.: Изд-во АН СССР, 1961.

84. ГОСТ 34.320-96. Информационные технологии. Система стандартов по базам данных. Концепции и терминология для концептуальной схемы и информационной базы.

85. *Грис Д.* Конструирование компиляторов для цифровых вычислительных машин. М.: Мир, 1975.

86. *Дантеман Д., Мишел Д., Тейлор Д.* Программирование в среде Delphi. Киев: Диасофт, 1995.

87. *Данильян О. Г., Панова Н. И.* Современный словарь по общественным наукам. М.: Эксмо-Пресс, 2005.

88. *Девятков В. В.* Онтологии и проектирование систем // Приборы и системы. Управление, контроль, диагностика. 2000. № 1.

89. *Девятков В. В.* Системы искусственного интеллекта. – М.: Изд-во МГТУ им. Н.Э. Баумана, 2001.

90. *Демьянков В. З.* Доминирующие лингвистические теории в конце XX века // Язык и наука конца 20 века. М.: Институт языкознания РАН, 1995. С.239-320.

91. *Джексон П.* Введение в экспертные системы / Пер. с англ. М.: Издательский дом «Вильямс», 2001.

92. *Дубровский Д. И.* Проблема идеального. Субъективная реальность. М.: Канон, 2002.

93. *Евреинов Э. В., Косарев Ю. Г.* Однородные универсальные вычислительные системы высокой производительности. Новосибирск: Наука. 1966.

94. *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).

95. *Закревский А. Д.* Логические уравнения. Минск: Наука и техника, 1975.

96. *Закревский А. Д.* Алгоритм декомпозиции булевых функций // Труды Сибирского физико-технического института. 1964. Вып. 44. С. 5-16.

97. *Захаров В.Н.* Автоматы с распределенной памятью. М.: Энергия, 1975.

98. *Зинченко В. П.* Живое знание: психологическая педагогика. Самара, 1998.

99. *Зубков С. В.* Ассемблер для DOS, Windows и Unix. М.: ДМК, 2004.

100. *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.

101. *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.

102. *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.

103. *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.

104. *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.

105. *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.

106. *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.

107. Искусственный интеллект: В 3-х кн. Кн. 2. Модели и методы: Справ. / Под ред. Д.А. Поспелова. М.: Радио и связь, 1990.

108. Искусственный интеллект: В 3-х кн. Кн. 3. Программные и аппаратные средства: Справ. / Под ред. В.Н. Захарова, В.Ф. Хорошевского. М.: Радио и связь, 1990.

109. *Калашян А.Н., Калянов Г.Н.* Структурные модели бизнеса: DFD-технологии. М.: Финансы и статистика, 2003.

110. *Кант И.* Критика чистого разума. М.: Мысль, 1994.

111. *Карповский М.Г., Москалев Э.С.* Реализация частично-определенных функций алгебры логики с помощью разложения в ортогональные ряды // АиТ. 1970. № 8.

112. *Карри Х. Б* Основания математической логики. М.: Мир, 1969.

113. *Карнап Р.* Значение и необходимость. М.: Мир, 1959.

114. *Касьянов В. И., Поттосин И. В*. Методы построения трансляторов. Новосибирск: Наука, 1986.

115. *Катречко С. Л*. Формальная и трансцендентальная логика // Тезисы докл. межд. конф. «Современная логика». СПб, 2004.

116. *Киндлер Е.* Языки моделирования. М.: Энергия, 1985.

117. *Клини С.К*. Математическая логика. М., Мир, 1973.

118. *Кнут Д.* Семантика контекстно-свободных языков // В сб.: Семантика языков программирования. М.: Мир, 1980.

119. *Кондаков Н. И.* Логический словарь. М., 1971.

120. *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.

121. *Конноли Т., Бегг К.* Базы данных: проектирование, реализация и сопровождение. Теория и практика. М.: Вильямс, 2003.

122. *Костогрызов А. И., Липаев В. В*. Сертификация качества функционирования автоматизированных информационных систем. М.: Изд-во «Вооружение. Политика. Конверсия», 1996.

123. *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/).

124. Краткий психологический словарь / Под общ. ред. А.В Петровского, М.Г. Ярошевского. Ростов-на-Дону: Феникс, 1999.

125. *Крицкий С. П.* Аксиоматическое описание контекстных связей и условий // Программирование. 1980. № 6.

126. *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.

127. *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.

128. *Кузнецов В. И.* Двухместные и трехместные отношения между научными понятиями // Logical Studies. 2004. № 12. С. 1-24. (http://www.logic.ru/Russian/LogStud/).

129. *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.

130. *Кузнецов О. П.* О программной реализации логических функций и автоматов // Автоматика и телемеханика. 1977. № 7. С. 163-174. № 9. С. 137-139.

131. *Кузнецов О. П., Адельсон-Вельский Г. М.* Дискретная математика для инженера. М.: Энергоатомиздат, 1988.

132. *Кузнецов О. П.* Дискретная математика для инженера. М.: Лань, 2005.

133. *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.

134. *Кун Т.* Структуры научных революций. М.: Прогресс, 1977.

135. *Кухарев Г. А., Шмерко В. П., Янушкевич С. П.* Техника параллельной обработки бинарных данных на СБИС. Минск: Выш. шк., 1991.

136. *Лагута О. Н.* Логика и лингвистика. Новосибирск, 2000.

137. *Лапкин Л.Я.* О векторной программной реализация логических функций // АиТ. 1983. № 3. С. 120-128.

138. *Лейбниц Г. В.* Сочинения: В 4-х т. Т. 3. М.: Мысль, 1984.

139. *Леоненков А. В.* Самоучитель UML. СПб.: BHV, 2006.

140. *Лидл Л., Нидеррайтер Г.* Конечные поля: В 2-х т. Т. 1. М.: Мир, 1988.

141. *Лисичкина Н. В.* Проблемы современной логической теории понятий // Матер. Третьего Российского философского конгресса «Рационализм и культура на пороге III тысячелетия». Ростов-на-Дону, 2002.

142. *Липаев В. В.* Качество программных средств. М.: Эдиториал УРСС, 2002.

143. Логический подход к искусственному интеллекту / Под ред. Гаврилова Г. П. М.: Мир,1990.

144. *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.

145. *Лорьер Ж.-Л.* Системы искусственного интеллекта. М.: Мир, 1991.

146. *Лупанов О.Б.* Об одном методе синтеза схем // Известия высших учебных заведений. Радиофизика. 1958. № 1. С. 120-140.

147. *Лупанов О.Б.* О синтезе некоторых классов управляющих систем. М.: Физматгиз, 1963. Вып. 10. С. 63-97.

148. *Люгер Д. Ф.* Искусственный интеллект: стратегии и методы решения сложных проблем. М.: Вильямс, 2005.

149. Макетирование, проектирование и реализация диалоговых информационных систем / Под ред. Е. И. Ломако. М.: Финансы и статистика, 1993.

150. *Макклеллан Дж.Х., Рейдер Ч.М.* Применение теории чисел в цифровой обработке сигналов. Москва: Радио и связь, 1983.

151. *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.

152. *Малюгин В.Д.* О полиномиальной реализации кортежа булевых функций // ДАН СССР. 1982. Т. 265. № 6. С. 1338-1341.

153. *Малюгин В. Д.* Реализация кортежей булевых функций посредством линейных арифметических полиномов // АиТ. 1984. № 2. С. 114-122.

154. *Марка Д.А., Мак-Гоуэн К.* Методология структурного анализа и проектирования. М.: Мета Технология, 1993.

155. *Марков А. А*. Теория алгорифмов. М.: Изд-во АН СССР, 1954.

156. Математическая энциклопедия: В 4-х т. М.: Советская энциклопедия, 1977-1985.

157. *Мендельсон Э*. Введение в математическую логику. М.: Наука, 1976.

158. *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.

159. *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.

160. *Моррис Ч.* Основания теории знаков // В кн.: Семиотика / Под ред. Ю. С. Степанова. М.: Наука, 1983.

161. *Маторин С. И.* О новом методе системологического анализа, согласованном с процедурой объектно-ориентированного проектирования // Кибернетика и системный анализ. 2002. №1. С. 118-130.

162. *Назаренко Г. И., Осипов Г. С.* Основы теории медицинских технологических процессов. Часть1.-М.: ФИЗМАТЛИТ, 2005.

163. *Нариньяни А. С*. Недоопределенность в системе представления и обработки знаний // Техничеся кибернетика. 1986. № 5.

164. *Непейвода Н. Н.* Прикладная логика. Новосибирск, 1999.

165. *Непейвода Н. Н., Скопин И. Н.* Основания программирования. М.: РХД, 2003.

166. *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.

167. *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.

168. *Ничепорук Э. И.* О синтезе схем с помощью линейных преобразований переменных // Докл. АН СССР. 1958. Т. 123. Вып. 4. С. 610-612.

169. Новейший философский словарь: 3-е изд., исправл. Мн.: Книжный Дом. 2003.

170. *Осипов Г. С.* От ситуационного управления к прикладной семиотике // Новости искусственного интеллекта. 2002. №6. С.56-59.

171. *Пархоменко П. П.* Синтез релейных структур на различных функционально полных системах логических элементов // АиТ. 1964. № 6.

172. *Перекрестенко А.* Разработка системы автоматического синтаксического анализа на основе мягко контекстно-зависимой унификационной грамматики // Тр. конф. «Диалог-2004». 2004.

173. 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.

174. *Першиков В. И., Савинков В. М.* Толковый словарь по информатике. М.: Финансы и статистика, 1991.

175. *Петренко А.* Венский метод разработки программ (неформальное введение) // Программирование. 1991. № 6. С. 23-27.

176. *Пильщиков В. Н.* Язык ПЛЭНЕР. М.: Наука, 1983.

177. *Пирс Ч. С.* Избранные философские произведения. М.: Логос, 2000.

178. *Пирс Ч. С.* Логические основания теории знаков. СПб.: Лаборатория метафизических исследований философского факультета СпбГУ; Алетейя, 2000.

179. *Плесневич Г. С.* Силлогистики для семантических сетей // Новости искусственного интеллекта. 2004. № 3.

180. *Поваров Г. Н.* О функциональной разделимости булевых функций // Докл. АН СССР. 1954. Т. 94. С. 801-803.

181. *Поваров Г. Н.* Математическая теория синтеза контактных (1,k)-полюсников // Докл. АН СССР. 1955. Т. 5.

182. *Поспелов Д. А.* Логико-лингвистические модели в системах управления. М.: Энергоиздат, 1981.

183. *Поспелов Д. А.* Ситуационное управление: теория и практика. М.: Наука, 1986.

184. *Поспелов Д. А.* Моделирование рассуждений. Опыт анализа мыслительных актов. М.: Радио и связь, 1989.

185. *Поспелов Д. А., Осипов Г. С.* Прикладная семиотика // Новости искусственного интеллекта. 1999. № 1. С. 9-35.

186. *Поспелов Д. А.* Семиотические модели: успехи и перспективы // Кибернетика. 1976. № 6. С.114-123.

187. *Поспелов Д. А., Пушкин В. Н.* Мышление и автоматы. М.: Сов. радио, 1972.

188. Поиск подходов к решению проблем / Прангишвили И. В., Абрамова Н. А., Спиридонов В. Ф. и др. М.: Синтег, 1999.

189. *Пратт Т., Зелковиц М.* Языки программирования: разработка и реализация. СПб.: Питер, 2002.

190. Программирование в ограничениях и недоопределенные модели / Нариньяни А. С., Телерман В. В., Ушаков Д. М., Швецов И. Е. // Информационные технологии, 1998. №7.

191. *Пушкин В. Н.* Эвристика – наука о творческом мышлении. М., 1967.

192. *Рамбо Дж., Буч Г., Якобсон А.* UML. Специальный справочник. СПб.: Питер, 2002.

193. *Рейуорд-Смит В.* Теория формальных языков. Вводный курс. М.: Радио и связь, 1988.

194. *Себеста Р. У.* Основные концепции языков программирования. М.: Вильямс, 2001.

195. *Серебряков В. А., Галочкин М. П.* Основы конструирования компиляторов. М.: Едиториал УРСС, 2003.

196. 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.

197. *Смирнов С. В.* Онтологический анализ предметных областей // Известия Самарского научного центра РАН. 2001. Т. 3. № 1. С. 62-98.

198. *Смирнова Е. Д.* Формализованные языки и проблемы логической семантики. М., 1982.

199. *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.

200. Социология: Энциклопедия / Сост. А.А. Грицанов, В.Л. Абушенко, Г.М. Евелькин, Г.Н. Соколова, О.В. Терещенко. Мн.: Книжный Дом, 2003. 1312 с.

201. Стандарт онтологического исследования IDEF5 (www.idef.com/idef5.html).

202. *Столлингс В*. Операционные системы. М.: Издательский дом «Вильямс», 2002.

203. *Тамм Б. Г., Пуусепп М. Э., Таваст Р. Р.* Анализ и моделирование производственных систем. М.: Финансы и статистика, 1987.

204. *Тарский А*. Введение в логику и методологию дедуктивных наук. М.: Изд-во иностр. лит., 1948.

205. Тестирование на основе моделей / Петренко А., Бритвина Е., Грошев С. И и др. // Открытые системы. 2003. № 9.

206. *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.

207. *Турчин В. Ф.* Метаалгоритмический язык // Кибернетика. 1968. № 4. С. 45-54.

208. *Турчин В. Ф*. Базисный РЕФАЛ. Описание языка и основные приемы программирования. М.: ЦНИПИАСС, 1974.

209. *Тыугу Э. Х.* Концептуальное программирование. М.: Наука, 1984.

210. *Успенский В.А*. Машина Поста. М.: Наука, 1988.

211. *Фараджиев Р. Г., Ципкин Я. З.* Преобразования Лапласа-Галуа в теории последовательных машин // Доклады Академии наук СССР. 1966. Т. 166 № 36.

212. *Филд А., Харрисон П.* Функциональное программирование. М.: Мир, 1993.

213. *Финько О. А*. Модулярная арифметика параллельных логических вычислений. Краснодар, Краснодарский военный институт, 2003.

214. 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.

215. *Фреге Г.* Избранные работы: Пер. с нем. А.Л. Никифорова. М.: ДИК, Русское феноменологическое общество, 1997.

216. *Френкель А., Бар-Хиллел И.* Основания теории множеств. М., 1966.

217. *Фролов С. С.* Социология: Учебник. М.: Наука, 1994.

218. *Хантер Р.* Основные концепции компиляторов. М.: Вильямс, 2002.

219. *Хорошевский В. Ф.* ATNL-машина – вопросы программной и аппаратной реализации // Сборник трудов I-го симпозиума ИФАК по искусственному интелекту. Ленинград, 1983.

220. *Цейтин Г. С.* О соотношении естественного языка и формальной модели // Вопросы кибернетики. М., 1982. С. 20-34.

221. *Цейтин Г. С.* Программирование на ассоциативных сетях // ЭВМ в проектировании и производстве. Л.: Машиностроение, 1985. Вып. 2. С. 16-48.

222. *Чебурахин И. Ф.* Синтез дискретных систем и математическое моделирование. М.: Физматлит, 2004.

223. *Чен П.* Модель «сущность-связь» – шаг к единому представлению данных // СУБД. 1995. № 3. С. 137-158.

224. *Черемных С. В., Семенов И. О., Ручкин В. С.* Структурный анализ систем: IDEF-технологии. М.: Финансы и статистика, 2001.

225. *Черч А.* Введение в математическую логику. М.: Из-во иностр. лит-ры, 1959.

226. *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.

227. *Шалыто А. А.* SWITCH-технология. Алгоритмизация и программирование задач логического управления. СПб.: Наука, 1998.

228. *Шалыто А. А.* Логическое управление. Методы аппаратной и программной реализации алгоритмов. СПб.: Наука, 2000.

229. *Шамис А. Л.* Поведение, восприятие, мышление: проблемы создания искусственного интеллекта. М.: Едиториал УРСС, 2005.

230. *Шаров С. А.* Средства компьютерного представления лингвистической информации. М.: Российский научно-исследовательский институт искусственного интеллекта, 1996.

231. *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.

232. *Шеннон К. Э.* Работы по теории информации и кибернетике. М.: Изд-во иностр. лит., 1963.

233. *Эббинхауз Г.Д., Якобс К., Ман Ф.-К., Хермес Г.* Машины Тьюринга и рекурсивные функции. М., 1972.

234. *Эдельман С. Л.* Математическая логика. М., Наука, 1975.

235. *Эйнштейн А.* Физика и реальность. М., 1965.

236. *Яблонский С. В.* Введение в дискретную математику. М., Наука, 1988.

237. *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.

238. *Яблонский С. В.* Об алгоритмических трудностях синтеза минимальных контактных схем // Проблемы кибернетики. Вып. 2. М.: Физматгиз, 1959. С. 75-121.

239. *Янов Ю. И.* О логических схемах алгоритмов // Проблемы кибернетики. Вып. 1. 1958.

240. *Янов Ю. И.* Математика, метаматематика и истина // М.: Институт прикладной математики им. М.В. Келдыша, 2006.

241. *Akers S. B.* On a Theory of Boolean Functions // Journal of Society for Industrial and Applied Mathematics. 1959. No. 7, 4.

242. *Akers S. B.* Binary decision diagrams // IEEE Trans. Computers. 1978. Vol. C-27, No. 6. P. 509-516.

243. *Allen J.* Natural Language Understanding. Melno Park, CA: Benjamin/Cummings, 1987.

244. *Ashenhurst R. L.* The Decomposition of Switching Functions // Bell Laboratory Report, 1952, No. BL-1(11). P. 541-642.

245. *Bach E., Jelinek E., Kratzer A., Partee B*. Quantification in Natural Languages. Dordrecht: Kluwer, 1995.

246. *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.

247. *Boole G.* The Laws of Thought. London: Macmillan, 1854.

248. *Brodie L.* Thinking FORTH. A Language and Philosophy for Solving Problems. – N.J.: Englewood Cliffs, Prentice-Hall, Inc., 1984.

249. *Brodie M. L., Mylopoulos J., Schmidt J. W*. On Conceptual Modeling: Perspectives from Artificial Intelligence, Databases and Programming Languages. New York: Springer-Verlag, 1984.

250. *Bryant R. E.* Graph-based algorithms for Boolean function manipulation // IEEE Trans. on Comp. 1986. V. 35. P. 677-691.

251. *Chomsky N*. Aspects of the Theory of Syntax. Cambridge, MA: MIT Press, 1965.

252. *Chomsky N*. Knowledge of Language: Its Origin. Nature and Use. New York: Praeger Publishers, 1986.

253. *Chrestenson H. E.* A class of generalized Walsh functions // Pacific J. Math. 1955. Vol. 5. P. 17-31.

254. *Cohn M.* Switching Functions Canonical Form over Integer Fields (Ph.D. Thesis). Cambridge: Harvard Univ., 1960.

255. *Collins A., Quillian M. R.* Retrieval time from semantic memory // Journal of Verbal Learning and Verbal Behavior. 1969. No 8. P. 240-247.

256. *Colmerauer A.* Les Grammaires de Metamorphose. Universite Aix-Marseille, 1975.

257. Coloring Large Graph // Proc. of the 1981 S.E. Conf. on Graph Theory, Combinatorics.

258. CCortex: A scalable virtual brain (http://www.ad.com).

259. 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.

260. *Cousineau G., Curien P.-L., Mauny M.* The categorical abstract machine // Science of Computer Programming. 1987. Vol. 8, No 2. PP. 173-202.

261. *Curtis H.A.* Non-Disjunctive Decomposition // Bell Laboratory Report, 1958, No. 19, P. 49.

262. *Davies J., Fensel D., Bussler C., Studer R.* The Semantic Web Research And Applications: First European Semantic Web Symposium. Heraklion, 2004.

263. *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.

264. *Farber D. J., Griswold R. E., Polonsky I. P.* A String Manipulation Language // JACM. 1964. No 11. PP. 21-30.

265. *Floyd R.* Assigning meaning to programs // Mathematical Aspects Computer Science. Amer. Math. Soc. 1967. Vol. XIX. PP. 19-32.

266. *Forgy C. L.* Rete: a fast algorithm for the many pattern/many object pattern match problem // Artificial Intelligence. 1982. No 19. P. 17-37.

267. Formal Methods for Distributed Processing / Ed. Bowman H., Derreck J. Cambridge, Cambridge University Press, 2002.

268. *Francez N.* Verification of programs. Addison-Wesley. 1992.

269. *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).

270. *Ganter B., Wille R.* Formal Concept Analysis: Mathematical Foundations, Springer, 1999.

271. *Gazdar G., Klein E., Pullum G., Sag I.* Generalized Phrase Structure Grammar, Oxford: Basil Blackwell, 1985.

272. *Gazdar G., Mellish C.* Natural Language Processing in Prolog: An Introduction to Computational Linguistics. Massachusetts: Addison-Wesley, 1989.

273. *Green D.H.* Reed-Muller Expansions of Incompletely Specified Functions // Proc. IEE. 1987. Part E-134. P. 228-236.

274. *Gruber T.* A. Translation Approach to Portable Ontology Specifications // Knowledge Acquisition Journal. 1993. Vol. 5. PP. 199-220.

275. *Haar A.* Zur Theorie der Orthogonalen Funktionensysteme // Math. Ann. 1910. Vol. 69. P. 331-371.

276. *Halliday M. A.* An Introduction to Functional Grammar. London: Edward Arnold, 1985.

277. *Heidorn G. E.* Augmented phrase structure grammar / In Schank and Nash-Webber. 1975.

278. *Herbrand J.* Une methode de demonstration. Thesis, 1931.

279. *Hoare C. A.* An axiomatic basis for computer programming // Communications ACM. 1969. Vol. 12, No 10. PP. 576-583.

280. *Hopfield J. J* Neural networks and physical systems with emergent collective computational abilities // Proc. National Academy of Science. No 79. 1982.

281. *Hendrix G. G.* LIFER: A natural language interface utility // SIGART Newsletter. 1977. Vol. 61. P. 25-26.

282. *Johnson C. D.* Formal Aspects of Phonological Descriptions. Mouton: Hague, 1972.

283. *Kasper R. T., Rounds W. C.* A logical semantics for feature structures. // Proc. 24th Annual Conference of ACL. New York. 1986. P. 235-242.

284. *Kay A.* The Early History of Smalltalk // ACM SIGPLAN Notices. 1993. No 28(3). PP. 69-75.

285. *Knuth D. E.* Semantics of context-free languages // Math. Systems Theory. 1968. Vol. 2, No 2, P. 127-145.

286. *Kohonen T.* Self-Organization and Associative Memory. Berlin, Springer-Verlag, 1984.

287. *Kowalski R. A.* Algorithm = Logic + Control // CACM. 1979. Vol. 22, No 7.

288. *Lambek J.* The mathematics of sentence structure // American Mathematical Monthly. 1958. Vol. 65.

289. *Landin P. J.* The mechanical evaluation of expressions // Computer Journal. 1964. Vol. 6. PP. 308-320.

290. *Lechner R.J.* Harmonic Analysis of Switching Functions. // Recent Developments in Switching Theory. Academic Press, 1971. P. 121-228.

291. *Lee C.Y.* Representation of switching circuits by binary decision programs // Bell Systym Technical Journal. 1959. Vol. 38, No. 4. P. 985.

292. *Lenat D., Miller G., Yokoi T.* CYC, WordNet and EDR: critiques and responses // Communications of the ACM. 1995. Vol. 38 (11). P. 45-48.

293. *Linger R., Mills H., Witt B.* Structured Programming: Theory and Practice. Reading. MA: Addison-Wesley, 1979.

294. *Wegner P.* Vienna definition language // Computer Surveys. 1972. Vol. 4. No 1. PP. 5-63.

295. *Maitra K. K.* Cascaded Switching Networks of Two-Input Flexible Cells // IRE Trans. Electr. Comput. 1962. Vol. TC-11. P. 136-143.

296. *McCluskey E. J.* Logic Design of Multivalued IIL Logic Circuits // IEEE Trans. Comput. 1979. Vol. C28. No. 8. P. 564-569.

297. *Meyer B.* Object Technology: The Conceptual Perspective // Computer. 1996. No 1. P. 86-88.

298. *Morris Ch. W.* Foundations of the theory of signs // International Encyclopedia of Unified Scence I. Chicago, 1938. PP. 17-31.

299. *Moortgat M.* Categorical Type Logics: Handbook of Logic and Language. Elsevier, 1997.

300. *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.

301. *Newell A.* The knowledge level // Artificial Intelligence. 1982. No 18. PP. 87-127.

302. *Parr T.* The Definitive ANTLR Reference: Building Domain-Specific Languages. Dallas: Pragmatic Bookshelf, 2007.

303. *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.

304. *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.

305. *Post T. L.* Introduction to a General Theory of Elementary Proposition // Amer. J. Math. 1921. Vol. 43. P. 163-185.

306. *Pradhan D. K.* A Multi-Valued Algebra Based on Finite Fields // Proc. Int. Symp. On MVL. 1974. P. 95-112.

307. *Pratt V. R.* LINGOL: A Progress Report // Proc. 4th IJCAI. 1975. P. 422-428.

308. *Rademacher H.* Einige Sätze von allgemeinen Orthogonalfunktionen // Math. Annalen. 1922. Vol. 87. P. 122-138.

309. *Rader C. M.* Discrete convolution via Mersenne transform // IEEE Trans. Comp. 1972. Vol. C-21.

310. *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.

311. *Roth J. P.* Minimization over Boolean Trees // IBM Journal. 1960. Vol. 4, 5. P. 543-555.

312. *Roth J.P., Karp R.M.* Minimization Over Boolean Graphs // IBM Journal Res. and Develop. 1962. No. P. 227-238.

313. *Rosenblatt F.* Principles of Neurodynamics. New York: Spartan, 1962.

314. *Rudeanu S.* Boolean Functions and Equations. Amsterdam; London: North-Holland Publ. Co., 1974.

315. *Saraswat V. A.* Concurrent Constraint Programming. Cambridge: MIT Press, 1993.

316. *Savinov A.* Logical Navigation in the Concept-Oriented Data Model // Journal of Conceptual Modeling. 2005. Issue 36.

317. *Schank R. C., Rieger C. J.* Inference and the computer understanding of natural languages. Artifical Intelligence. 1974. Vol. 5, No 4. P. 373-412.

318. *Scott D. S.* Lectures on a mathematical theory of computations. Oxford University Computing, 1981.

319. *Selz O.* Zur Psychologie des Productiven Denkens und des Irrtums // Bonn: Friedrich Cohem, 1922.

320. *Semon W. L.* Characteristic Numbers and Their Use in the Decomposition of Switching Functions // Proc. ACM. 1952. Vol. 17. P. 273-280.

321. *Shannon C.E.* The a Symbolic Analysis of Relay and Switching Circuits // Trans. of American Inst. of Electrical Eengineers. 1938. Vol. 57. P. 713.

322. *Shannon C. E.* The Synthesis of Two-Terminal Switching Circuits // Bell Syst. Techn. J. 1949. Vol. 28. No. 1. P. 59-98.

323. *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.

324. *Smith B., Mulligan K.* Framework for Formal Ontology // Topoi. 1983. V.2. P. 73-85.

325. *Sowa J. F.* Conceptual Structures: information processing in mind and machine. Cambridge, MA: Addison Wesley, 1984.

326. *Sowa J. F.* Towards the expressive power of natural language // In Principles of Semantic Networks. Morgan Kaufman, 1991. P. 157–189.

327. *Sowa J. F.* Conceptual Graphs as a universal knowledge representation. // Computers and Mathematics with Applications. 1992. Vol. 23, No 2-5. P. 75-93.

328. 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.

329. *Strazdins I.* The Polynomial Algebra of Multiple-Valued Logic // Algebra, Combinatorics and Logic in Computer Science. 1983. Vol. 42. P. 777-785.

330. *Stoy J. E.* Denotational semantics: the Scott-Strachey approach to programming language theory. MIT Press. 1977.

331. *Su Y.H., Cheung P.T.* Computer Minimization of Multiple-Valued Switching Function // IEEE Trans. Comput. 1972. Vol. C21. P. 995-1003.

332. *Tarski A.* Logic, Semantics, Metamathematics. Oxford, 1956.

333. *Tarski A.* Das Wahrheitsbegriff in den formalisierten Sprachen // Studia Philosophica. 1935. № 1. S. 261-405.

334. *Thompson S.* Type Theory and Functional Programming. Addison-Wesley, 1991.

335. *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.

336. *Tosic Z.* Analytical Representation of an m-Valued Logical Function over the Ring of Integers Modulo m (Ph.D. Thesis). Beograd, 1972.

337. *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.

338. *Vykhovanets V. S.* The generalized multiplicative forms // Международная конф. по пробл. упр. М., 1999. Т. 3. С. 319-321.

339. *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.

340. *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.

341. *Wadler P.* Why no one uses functional languages. ACM SIGPLAN Notices. 1998.

342. *Walliuzzaman K. M., Vranesic Z. G.* On Decomposition of Multiple-Valued Switching Functions // Computer Journal. 1970. Vol. 13. P. 359-362.

343. *Walsh J. L.* A closed set of orthogonal functions // Amer. J. Math. 1923. Vol. 55. P. 5-24.

344. *Webb D. L.* Generation of any N-valued Logic by One Binary Operator // Proc. Nat. Acad. Sci. 1935. Vol. 21. P. 252-254.

345. *Wijngaarden V.A., Mailloux B.J., Peck J.E., Koster C.H.* Report on the algorithmic language Algol-68. Amsterdam: Mathematisch Centrum, 1969.

346. *Wille R., Ganter D.* Formal Concept Analysis. Springer. Berlin: Verlag, 1999.

347. *Woods W. A.* Cascaded ATN grammars // American Journal of Comp. Linguistics. 1980. Vol. 6, No 1.

348. *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.

349. 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 специализированном языке, а также определение эффективности полученного описания. В качестве примера рассматривается задача управления лифтом[59], ставшая уже традиционной для демонстрации различных подходов к программированию[60].

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).

---

[59] *Крут Д.* Искусство программирования. В 3-х томах. Т. 1: Основные алгоритмы. М.: Вильямс, 2001.

[60] *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 специализированный предметный 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.

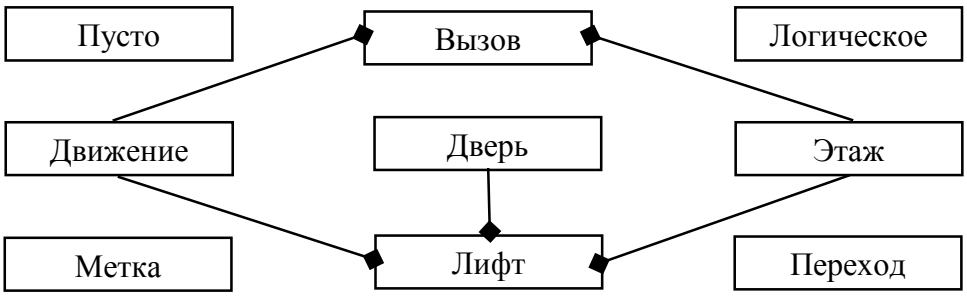

Рис. П1.1. Понятийная структура

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 специализированном 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.