ДИАЛОГОВЫЕ СИСТЕМЫ В АСУ

ЭНЕРГОАТОМИЗДАТ

Применение вычислительных машин
в исследованиях
и управлении производством

ДИАЛОГОВЫЕ СИСТЕМЫ В АСУ

Под редакцией
Д. А. Поспелова

МОСКВА
ЭНЕРГОАТОМИЗДАТ
1983

ББК 32.81
Д44
УДК 658.012.011.56:68/.3

Авторы: В. М. Брябрин, Ю. Я. Любарский, Л. Н. Микулич, Е. Я. Найденова, Д. А. Поспелов, А. Б. Преображенский, В. Ф. Хорошевский, А. Я. Червоненкис

Брябрин В. М. и др.

Д44 Диалоговые системы в АСУ/ В. М. Брябрин, Ю. Я. Любарский, Л. Н. Микулич и др.; Под ред. Д. А. Поспелова. — М.: Энергоатомиздат, 1983. — 208 с., ил.— (Применение вычислительных машин в исследованиях и управлении производством).

65 к.

Рассматриваются общие принципы построения диалоговых систем, использующих профессионально ориентированные подмножества естественного языка. Описываются существующие отечественные системы такого типа, предназначенные для работы в АСУ энергосистемами, транспортными узлами, в информационно-поисковых системах и роботах. Системы, описываемые в книге, ориентированы на отечественные ЭВМ единой серии и БЭСМ-6.

Для специалистов в области АСУ и программного обеспечения ЭВМ, а также для студентов старших курсов вузов и аспирантов соответствующих специальностей.

Д $\frac{2404000000\text{-}458}{051(01)\text{-}83}$ 210-83

ББК 32.81
6Ф0.1

ПРЕДИСЛОВИЕ

За последнее десятилетие накоплен большой опыт в проектировании и создании автоматизированных систем управления различного уровня. Этот опыт позволяет пересмотреть многие положения, казавшиеся бесспорными полтора десятилетия назад, высказать новые соображения о принципах построения АСУ и поставить новые задачи перед их создателями. В частности, стало очевидным, что для всех АСУ организационного типа (АСУ отраслями народного хозяйства, территориальные АСУ, АСУ объединений и т. п.) их эффективное функционирование невозможно без организации большого по объему информационного банка. В этом банке должна храниться не только информация, представленная в виде формализованных документов, содержащих числовую и словесную информацию, но и большое количество неформализованных данных, представляющих собой различные текстовые сообщения, написанные на обычном естественном языке. Кроме того, в памяти АСУ должны храниться разнообразные сведения о самом объекте управления, связанные с его структурой, функционированием и принятием решений по управлению. Как правило, эта информация также не может быть представлена в формальном виде, а представляется в виде высказываний об объекте, сделанных на естественном языке. Это положение сделало необходимым проведение работ по созданию специальных внутренних машинных представлений для отображения таких данных, по организации и ведению банков подобного типа.

Кроме того, общение пользователей с АСУ, как правило, не может происходить на формализованном языке, удобном для ЭВМ, являющейся непременной составляющей любой современной АСУ. Это происходит из-за профессиональной неподготовленности большинства пользователей, которым эта подготовка по служебному положению не представляется возможной. Это приводит к необходимости использовать в диалоге человек—АСУ—человек обычный естественный язык.

Высказанные положения привели к тому, что современные АСУ уже не мыслятся без наличия в них специальных средств организации диалога с человеком пусть на ограниченном, но естественном языке и средств внутреннего представления текстовой информации.

Цель этой книги состоит в попытке показать общие принципы построения диалоговых систем в АСУ. Внутренние средства представления текстовой информации будут в ней затрагиваться только в той мере, в которой это интересно и важно для диалоговых систем. В приведенной в конце книги библиографии есть специальный раздел, в котором указана основная литература и по этому важному вопросу. Знакомство с ней позволит получить полное представление о всем круге проблем, связанных с использованием текстов на естественных языках в АСУ и интеллектуальных системах различного типа.

В первых двух главах книги описываются общие принципы построения диалоговых систем высокого уровня, а в последующих главах на примере анализа нескольких отечественных диалоговых систем показано воплощение этих принципов в действующих системах. Проблемы, связанные с технологией создания и функционирования диалоговых систем (например, с программным обеспечением для них), в данной книге по существу отсутствуют. Эти вопросы, интересные для более узкого круга специалистов в области диалоговых систем, более подробно освещены в работах, указанных в библиографии, приведенной в конце книги для каждой из описанных в ней (и ряда других) систем.

В написании книги принимал участие большой коллектив авторов, работающих в различных организациях. Однако мы надеемся, что между авторами возникло такое взаимопонимание, которое позволило им подразумевать под одними и теми же словами одни и те же понятия, что не так просто делается в бурно развивающихся новых отраслях науки и техники.

Главы 1 и 2, а также комментарий к библиографии написаны Д. А. Поспеловым, гл. 3 — Ю. Я. Любарским, авторами гл. 4 являются Л. И. Микулич, Е. Я. Найденова и А. Я. Червоненкис, гл. 5 написана А. Б. Преображенским и В. Ф. Хорошевским, гл. 6 В. М. Брябриным. Общее редактирование рукописи и согласование материала отдельных глав проведено Д. А. Поспеловым.

Авторы

ГЛАВА ПЕРВАЯ
ОБЩАЯ СТРУКТУРА ДИАЛОГОВЫХ СИСТЕМ

1.1. Общие принципы

Проблемы формализации естественного языка, ввода текста в ЭВМ и понимания этого текста машиной возникли в 50-х годах в связи с бурным развитием машинного перевода. Энтузиасты считали, что существующие ЭВМ, казавшиеся мощными по производительности и памяти, смогут осуществить автоматизацию процесса перевода с одного языка на другой. Причем перевод этот мыслился как прямое преобразование конструкций одного языка в конструкции другого языка. На рис. 1.1 приведена соответствующая схема. Роль ЭВМ сводилась лишь к хранению введенных в нее (пусть даже очень больших) словарей $Я_1$—$Я_2$, некоторых средств для организации поиска в этих словарях и соединению найденных конструкций между собой. Процесс перевода предполагался, таким образом, чисто механическим. Если необходимо перевести на английский язык фразу: «Я студент», то в словаре конструкций должны содержаться, например, такие пары: *я=I am*, *студент=student*. И простое соотнесение данных пар даст на выходе системы английский эквивалент русской фразы: *I am student*. Конечно, специалисты, занимавшиеся в это время машинным переводом, не были столь наивными, чтобы считать, что только словари могут решить все проблемы. Если идти по такому пути, то объем их, как легко подсчитать, был бы столь велик, а время поиска столь большим, что даже ЭВМ ничего не смогли бы сделать за приемлемое для человека время. Аналоги всех возможных конструкций составили бы словарь огромных размеров. На его составление потребовался бы многолетний труд большого коллектива специалистов. Выход из этого кажущегося тупика дает сам язык. Любой естественный язык представляет собой синтаксически организованную систему. В нем существует конечный набор правил, которые для любой фразы языка однозначно отвечают на

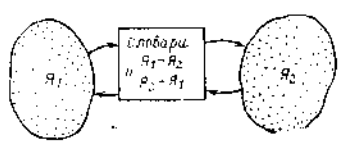


Рис. 1.1

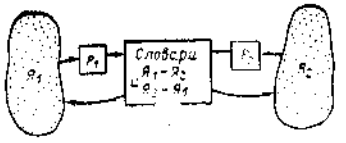


Рис. 1.2

вопрос, является ли она синтаксически правильной. (В математической логике этому соответствуют исчисления, в которых заданы правила построения правильных формул). Есть такой набор синтаксических правил и в русском языке. Именно они позволяют нам считать фразу: «Иванов является рабочим цеха № 2» правильной, а фразу: «Ивановым является рабочему цехом № 2» — неправильной. Конечность набора синтаксических правил определяет конечность допустимых синтаксических конструкций в естественном языке.

Это положение приводит к схеме, в которой вместо словарей языков $Я_1$ и $Я_2$ для преобразования используется словарь соответствия синтаксических конструкций этих языков и обычный пословный словарь (рис. 1.2). Процедуры P_1 и P_2 нужны для проведения *синтаксического анализа* фраз на естественном языке, цель которого состоит в определении синтаксической конструкции данной фразы. Например, все фразы типа: «Я студент», «Я рабочий», «Я архитектор» смогут рассматриваться как некоторая общая конструкция Я, α, где α есть имя существительное единственного числа, именительного падежа. Правда, такая же конструкция будет у фраз типа: «Я письмо» или «Я крик», но эти фразы с точки зрения норм естественного русского языка вполне допустимы синтаксически, а только о такой правильности мы и говорим. В английском языке синтаксической конструкции *Я* α будет соответствовать конструкция *I am* β. При этом β есть также имя существительное единственного числа. Необходимо только, чтобы конкретное слово в английской фразе, которое будет находиться на месте β, соответствовало по пословному словарю слову, занимающему место α в исходной фразе. Из сказанного вытекает, что при таком подходе к переводу необходимо иметь специальные процедуры, целью которых являлось бы определение таких характеристик слов, как число, падеж, род и т. д. Эти процедуры носят название процедур

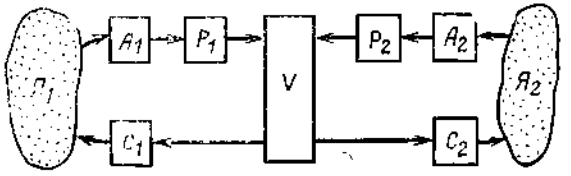


Рис. 1.3

морфологического анализа. На рис. 1.3 показана эта усложненная схема. На этом рисунке *V* обозначает словари $Я_1 — Я_2$ и $Я_2 — Я_1$. Блоки A_1 и A_2 соответствуют процедурам морфологического анализа, а C_1 и C_2 — процедурам морфологического синтеза, цель которых состоит в согласовании слов, входящих в окончательную фразу. Итак, уже на первом этапе развития работ в области машинного перевода выяснилась необходимость в двух шагах такого перевода: морфологическом и синтаксическом.

Однако попытки строить системы машинного перевода, опирающиеся на структуру, приведенную на рис. 1.3, очень скоро закончились неудачей. Уровень синтаксического анализа оказался слишком поверхностным. С его помощью, например, никак нельзя перевести на русский язык английскую фразу *It rains cats and dogs* (дослойный перевод: «Дождь идет кошками и собаками»). Эквивалент этой фразы по-русски:
«Дождь льет как из ведра» также непереводим с помощью схемы, показанной на рис. 1.3, на английский. Дело не только в подобных идиоматических оборотах. Простое несоответствие морфологических и синтаксических признаков может привести к неправильному переводу. Примером может служить перевод басни Лафонтена «Кузнечик и муравей» с французского, осуществленный И. А. Крыловым. По-французски слово кузнечик *la sigale* женского рода, а по-русски — мужского. Для сохранения женской беспечности и ветрености героя Крылов в русском варианте басни заменил кузнечика стрекозой. Но это повлекло за собой массу нелепостей. Во французском варианте муравей упрекает кузнечика за то, что он все лето пропел и пропрыгал, что вполне уместно. По-русски же подобные упреки к стрекозе (которая, конечно, ни петь, ни прыгать не умеет) выглядят весьма странно.

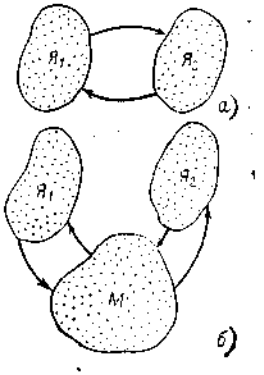


Рис. 1.4

Новый этап продуктивных исследований в области машинного перевода наступил только в начале 70-х годов после длительного периода разочарования в его возможности, вызванного как недостаточностью наших знаний о языке и неэффективностью схемы, показанной на рис. 1.3, так и недостаточной мощностью имевшихся в то время вычислительных средств. На этом новом этапе возникла принципиально новая идея перевода. Если

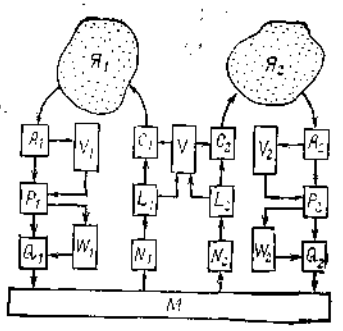


Рис. 1.5

раньше обмен информацией между индивидуумами, владеющими языками $Я_1$ и $Я_2$ соответственно, мыслился как прямой обмен между языковыми конструкциями (рис. 1.4,*а*), то теперь стало ясно, что этот обмен идет через некоторую общую для индивидуумов *модель мира* M (рис. 1.4,*б*). Эта модель мира содержит в себе всю информацию о внешнем мире, как языковую, так и внеязыковую. При условии, что модели мира двух индивидуумов совпадают, можно предполагать, что их взаимопонимание определяется совпадением некоторых конструкций в этих моделях. При несовпадении моделей взаимопонимание невозможно.

Появление модели мира усложняет структуру схемы, приведенной на рис. 1.3. Вместо нее возникает новая схема, показанная на рис. 1.5. Проследим по этой схеме этапы перевода. Фраза, написанная на языке $Я_1$, поступает на вход блока морфологического анализа. Словарь V_1 помогает этому блоку произвести необходимый анализ. В этом словаре хранятся списки основ слов, а в самом блоке A_1 хранится набор правил морфологического анализа и процедур их использования. После этого в блоке P_1 производится синтаксический анализ, с помощью которого определяется синтаксический тип фразы, а затем с помощью процедур, хранящихся в блоке Q_1, и информации, хранящейся в словаре W_1 (там хранятся так называемые *семантические* модели, которые подробно будут рассматриваться далее), происходит перевод исходной фразы в модельное представление. Иначе, фраза, написанная на естественном языке $Я_1$, после этой цепочки преобразований оказывается представленной на языке представления знаний, используемом при описании действительности в модели мира.

Сформулируем некоторые общие требования, которым должен удовлетворять язык представления знаний, который мы будем в дальнейшем обозначать как $Я_с$. Этих требований три: 1. Если некоторая фраза F естественного языка, по мнению людей-экспертов, являющихся носителями этого языка, является осмысленной, то должна существовать фраза в $Я_с$, соответствующая F. Если F в $Я_с$ соответствует несколько фраз, то с помощью некоторой конструктивной и эффективной процедуры эти фразы должны в языке $Я_с$ переводиться друг в друга. 2. Если некоторая фраза F естественного языка, по мнению людей-экспертов, являющихся носителями этого языка, не является осмысленной, то в $Я_с$ не должно существовать фраз, соответствующих F. 3. Если, по мнению людей-экспертов, являющихся носителями естественного языка, две фразы этого языка F_1 и F_2 эквивалентны по смыслу, то в $Я_с$ им должна соответствовать либо одна и та же фраза, либо различные фразы, но такие, что они формально преобразуются в $Я_с$ друг в друга.

Прокомментируем эти требования. Во-первых, отметим тот факт, что понятие осмысленности фраз естественного языка никак не формализуется и определяется только через экспертизу. Это приводит к неоднозначному толкованию осмысленности. В качестве примера можно указать на фразу: «Идея яростно спит», которая долгое время трактовалась как пример неосмысленной фразы. Однако удалось построить пример такого текста, в контексте которого эта фраза становится вполне осмысленной. Вот этот текст: «Идея яростно спит, ворочается во сне. Идея в висках стучит, нашептывая мне ...». Этот пример показывает, что этап перевода из естественного языка в представления в $Я_с$ не может быть простым. Процедуры подобного перевода должны учитывать контекст, в котором находится переводимая фраза. Правила анализа и учета этого контекста и составляют семантическую модель, о которой мы уже упоминали. Для одной фразы и даже для одного слова таких моделей может быть несколько. Создание подобных моделей дело нелегкое даже для ограниченных подъязыков естественного языка. Одной из трудных проблем, возникающих при построении семантических моделей, является, в частности, проблема определения границ контекста, который необходимо проанализировать для однозначного перехода к представлению на языке $Я_с$. В определении $Я_с$ содержится требование о наличии эффективных процедур трех типов: процедуры перевода фраз естественного языка, имеющих смысл, на $Я_с$, процедуры обратного перевода на естественный язык и процедуры эквивалентных преобразований в множестве фраз $Я_с$. Проблема построения этих процедур сложна и весьма далека от полного решения.

Вернемся к схеме, показанной на рис. 1.5. Мы рассмотрели переход от $Я_1$ к $Я_с$. Переход от $Я_2$ к $Я_с$ происходит аналогичным образом. Обратный перевод с языка $Я_с$ на естественные языки $Я_1$ и $Я_2$ также происходит однотипно. Сначала по модельному представлению строится обобщен-

ная синтаксическая структура фразы (блоки N_1 и N_2), затем эта структура согласуется с синтаксической структурой соответствующего естественного языка (блоки L_1 и L_2) и, наконец, происходит переход к основам слов и морфологической обработке этих основ.

Сформулируем *принцип обратимости*. Суть этого принципа состоит в том, что все процедуры, обеспечивающие переход от фраз естественного языка к фразам языка $Я_с$, устроены так, что они являются обратимыми. Другими словами, если выходы этих процедур считать их входами и все операции заменить на обратные, то те же самые процедуры, которые использовались при переходе от естественного языка к $Я_с$, можно использовать при переходе от $Я_с$ к фразам естественного языка.

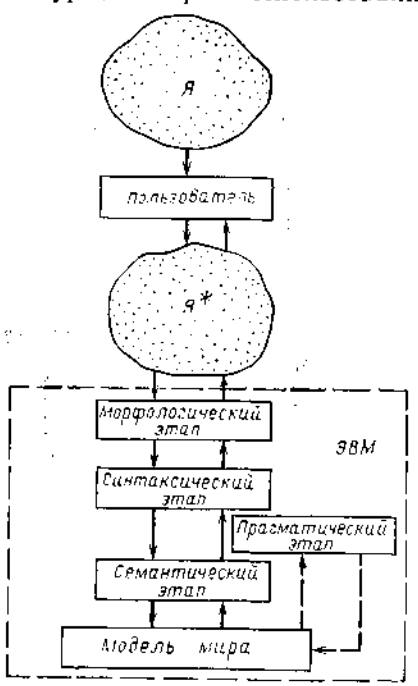


Рис. 1.6

Реализация принципа обратимости позволяет существенно упростить схему на рис. 1.5. Прежде чем делать это, заметим, что в этой книге мы интересуемся не переводом с одного языка на другой, а диалоговыми системами, реализующими коммуникацию между человеком, говорящим на некотором естественном языке $Я$, и вычислительной машиной. При этом можно предполагать, что сама машина в качестве своего «естественного» языка использует язык $Я_с$. В этих предположениях схема диалоговой системы примет вид, показанный на рис. 1.6.

Рассмотрим эту схему более подробно. Пользователь, являющийся носителем естественного языка $Я$, при общении с системой во всех практически интересных случаях использует не все средства, имеющиеся в языке $Я$, а лишь те из них, которые помогают ему в решении задач, связанных с обращением к системе. В самом деле, если система предназначена для выдачи информации о состоянии погоды на трассах движения самолетов, то обращаться к ней с вопросами, касающимися кадрового состава летчиков, бессмысленно. Именно поэтому для всех диалоговых систем в АСУ пользователь обращается с системой не на языке $Я$, а на значительно более бедном по своим выразительным возможностям (по крайней мере по лексике) подъязыке $Я^*$. Этот вариант естественного языка и воспринимается системой. Этап морфологической обработки включает в себя процедуры A и C, синтаксический этап — процедуры P и L, а семантический этап — процедуры Q и N. Прагматический этап отражает внутренние действия системы, связанные с необходимостью проведения некоторых дополнительных процедур (например, некоторых вычислений), нужных для формирования ответа пользователю. Схема, показанная на рис. 1.6, лежит в основе всех диалоговых систем, используемых в АСУ различного типа, хотя объем, в котором может быть реализован каждый из этапов, может изменяться в весьма широких пределах вплоть до исключения некоторых этапов из них. В литературе процедуры, образующие морфологический, синтаксический и семантический этапы, часто объединяют под названием *лингвистического процессора*.

Уточним смысл еще одного термина, который будет употребляться в книге. Термин этот «понимание». Его употребление в отношении системы, реализованной на ЭВМ, имеет троякий смысл. Во-первых, понимание есть процесс соотношения некоторого текста с соответствующим фрагментом описания, хранящимся в модели мира M. В этом смысле пониманне есть процесс реализации процедур семантического этапа. Во-вторых, пониманне интерпретируется как пополнение ситуации, описанной в тексте, за счет информации о внешнем мире, хранящейся в M, т. е. включение описанной в тексте ситуации в более широкую картину действительности. Эта интерпретация понимания сводится к осуществлению процедур, реализуемых в самой модели M (в частности, процедур эквивалентных преобразований, о которых уже шла речь). Эти процедуры основываются на известных системе законах внешнего мира и логиках проведения рассуждения в этом мире. Наконец, в-третьих, термин «понимание» используется в том смысле, что система понимает введенный в нее текст, если она подобно человеку сможет правильно ответить на все вопросы, связанные с информацией, введенной этим текстом. Другими словами, понимание сводится к утверждению о наличии процедур, позволяющих формировать правильные ответы на вопросы, связанные с информацией, хранящейся

в M. Чтобы не впадать в многозначность истолкования термина «понимание», будем считать, что система обладает способностью понимать вводимые в нее тексты, если она понимает их во всех трех указанных выше смыслах.

Проиллюстрируем сказанное следующим примером. Пусть на вход системы поступает фраза: «Иванов пришел на работу в 8 ч утра и всю смену проработал на конвейере». Если эту фразу сообщили человеку, а затем спросили его: «Где был Иванов в 9 ч утра?», то естественным ответом было бы: «На работе». Однако если в систему введена указанная фраза, то для ответа на поставленный вопрос системе не хватает дополнительной информации, связанной с тем, что смена продолжается заведомо больше одного часа, а конвейер не позволяет работающему на нем отлучаться со своего рабочего места. Эта дополнительная информация, прямо не содержащаяся во введенной фразе, должна пополнить вводимую информацию за счет знаний о действительности, которые хранятся в модели мира системы.

До сих пор говорили об отдельных фразах, поступающих на вход системы. Однако такое пофразовое понимание вводимого текста не всегда возможно. При вводе же многофразового текста возникают дополнительные трудности, связанные с его пониманием. Примерами этих трудностей могут служить анафорические связи, характерные для текстов на естественном языке. Пусть, например, в систему вводится текст: «Станок № 125 сегодня вышел из строя. Починку производила бригада под руководством Иванова. К вечеру он начал работать». Между словами «станок № 125» из первой фразы и местоимением «он» из третьей фразы имеется анафорическая связь. Однако установление этого факта формальным методом весьма затруднительно, так как для ЭВМ местоимение «он» может ассоциироваться не только со станком, но и с Ивановым.

При анализе текстов в схему, приведенную на рис. 1.6, следовало бы добавить еще один этап (после семантического), который содержал бы процедуры, позволяющие анализировать семантическую структуру связного текста. Однако на сегодняшний день не существует практически реализуемых процедур, которые могли бы решить эту задачу. О некоторых частных процедурах такого типа будет сказано при описании примеров диалоговых систем.

1.2. Функционирование лингвистического процессора

Рассмотрим сначала морфологический этап. На этом этапе к каждому слову предложения добавляется его полная грамматическая характеристика, взятая из словаря. Для этого каждое слово расчленяется на составляющие. Например, для русского языка этими составляющими могут быть: префикс, словообразовательная основа, суффикс, окончание и постфикс. Так, для глагола *разрешаться* это расчленение будет выглядеть следующим образом: *раз* — префикс, *реш* — основа, *а* — суффикс, *ть* — суффикс, *ся* — постфикс. Возможность такого расчленения в лингвистическом процессоре обеспечивается наличием словаря основ для всех слов, используемых в языке $Я^*$, а также списков допустимых элементов для всех остальных позиций. Списки последнего типа включают в себя правила сочетаемости тех или иных элементов, а также полные грамматические характеристики, определяемые списками окончаний и постфиксов. Кроме того, в лингвистическом процессоре должны храниться правила морфологического анализа для нестандартных словоформ и так называемых аналитических форм (связанных между собой цепочек словоформ, например *буду писать*).

Результаты морфологического этапа позволяют на синтаксическом этапе построить дерево грамматического разбора фразы, поступившей на вход лингвистического процессора. Простейшая форма такого разбора, при которой вычленяются подлежащее, сказуемое, дополнения, обстоятельства различного типа и т. п., хорошо известна всем из школьной практики грамматического разбора. На синтаксическом этапе строится дерево зависимостей, аналогичное школьному дереву разбора, но обладающее гораздо более богатым набором отношений между словами, стоящими в узлах дерева. Число различных по смыслу связей, учитываемых в развитых диалоговых системах, достигает 30—50 (при ограничении предметной области, описываемой языком $Я^*$, только областью научно-технических текстов). Поясним сказанное на примере. Пусть на вход диалоговой системы подана фраза: «Высокое давление вызывает ухудшение механических свойств манипулятора». На рис. 1.7,*а* показан школьный синтаксический разбор этого предложения, а на рис. 1.7,*б* тот разбор, который характерен для наиболее развитых диалоговых систем и систем машинного перевода. Заметим, что на дереве, показанном на рис. 1.7,*б*, вершины взвешены словами в том виде, в котором они хранятся в словаре, а рядом с ними выписана часть информации, полученная на морфологическом этапе. Остальная часть морфологической информации (например, падеж существительного или род и число прилагательного) отражена в самих отношениях. Например, определительные

отношения автоматически согласуют подчиненное слово в числе с управляющим словом и т. д. Выбор списка отношений, которые используются при построении синтаксических деревьев, происходит с учетом дальнейшего перехода к семантическому этапу. Дело в том, что для русского (и многих других) языка синтаксис в значительной мере определяет семантику фразы. На этой идее выросла теория глубинных семантических падежей Филмора, о которой будет идти речь в гл. 2 книги. На этом же основано «понимание» абстрактных и абсурдных фраз. Примером фраз такого типа могут служить: «Глокая куздра штеко будланула бокра и курдячит бокренк» и Круглый квадрат сладко напевает». Первая из этих фраз составлена из слов, которых нет в русском языке, но записана с сохранением всех правил русского синтаксиса. Это приводит к тому, что большинство людей осмысливают эту абстрактную фразу и даже представляют себе некоторую внеязыковую ситуацию, «соответствующую» ей (как правило, нечто вроде злой коровы, боднувшей быка и треплющей теленка). Вторая фраза, несмотря на свою абсурдность, также воспринимается, как обладающая семантикой. Во всяком случае, человеку, услышавшему или прочитавшему эту фразу, ясно, кто и как напевает. Таким образом, на этапе синтаксического анализа необходимо сохранить всю ту синтаксическую информацию, которая позволяет правильно понять текст на семантическом этапе. Например, разница в семантике выражений *три дня* и *дня три* определяется лишь порядком слов, т. е. чисто синтаксической характеристикой. Для сохранения этой информации на этапе синтаксического анализа в первом случае между вершинами вводится количественное отношение, а во втором — аппроксимативно-количественное.

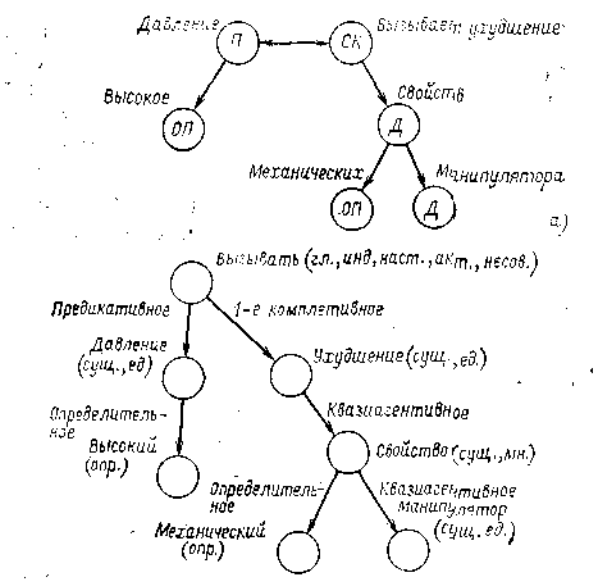


Рис. 1.7

Для перехода от морфологической информации к дереву синтаксических зависимостей используются специальные правила вывода, которые имеют следующий вид:

$$H;\ S_1(s_1)m_1;\ S_2(s_2)m_2;\ D \Rightarrow S_1 R^{12} S_2.$$

Здесь S_1 и S_2 — две словоформы, между которыми мы хотим установить связь R^{12}; s_i и m_i — соответственно синтаксическая и морфологическая информация, приписанная этим словоформам; D — информация о расположении этих *словоформ в анализируемой фразе (в частности, порядок* следования, контактность и т. д.), а H — некоторое условие применимости этого правила, зависящее от других словоформ, входящих или не входящих в данное предложение, условий на связь s_1 с s_2 и m_1 с m_2 и т. п. Кроме правил вывода, написанных в приведенной выше форме, можно использовать и различные обобщенные схемы правил. Частично эти обобщенные схемы будут рассмотрены в гл. 5 и 6. Заметим, что в продвинутых системах число различных схем правил вывода синтаксических отношений колеблется от 250 до 400.

Наконец, на семантическом этапе происходит преобразование синтаксического дерева в модельное представление. Модельные представления будут рассматриваться в гл. 2. Отметим лишь, что язык $Я_c$ не представляет собой чего-то изоморфного синтаксически естественному языку. Это видно, в частности, из того, что у человека информация о внешнем мире и событиях, протекающих в нем, не хранится в виде текстов, подобных тексту на языке, которым этот человек пользуется. Например, два очевидца одного и того же события, рассказывая о нем, синтезируют различные тексты. Даже при повторных рассказах одного индивидуума текст меняется. Сохраняется лишь некоторый общий смысл запечатленного в его модели мира.

На сегодняшний день уже созданы и успешно действуют системы, реализующие морфологический и синтаксический этапы в лингвистических процессорах. Поэтому в дальнейшем мы будем касаться реализации этих этапов

лишь в тех случаях, когда эти реализации идейно отличаются от описанного в этом разделе подхода. Основное же внимание сосредоточим на реализации семантического этапа и построении языка $Я_c$. Кроме того, рассмотрим организацию прагматического этапа в той ее части, которая связана с планированием вычислений с помощью модели мира.

1.3. Типы диалоговых систем

На пути развития АСУ различного типа диалоговые системы претерпели эволюцию. Системы, воплотившие в себя структуру, показанную на рис. 1.6, появились далеко не сразу. Да и в настоящее время не во всех АСУ требуется строить диалог с привлечением всех этапов, указанных на этой схеме. Ниже опишем несколько типов диалоговых систем, встречающихся в практике проектировщиков АСУ различного назначения.

Отметим прежде всего системы, в которых вообще нет лингвистического процессора. В таких системах общение пользователя с системой происходит непосредственно на языке модели мира. Ответы также выдаются на этом языке. Чтобы организовать подобный диалог, необходимо иметь стандартизированное представление информации во вводимых фразах, согласованное с представлением знаний в модели мира. Наиболее известным способом такой стандартизации являются *спецификационные списки*. Их организацию рассмотрим на следующем простом примере. Пусть в M хранятся данные о кадрах. Кадровая информация содержит сведения о фамилии и инициалах сотрудников, годе их рождения, поле и специальности. Эта информация организована в виде следующих единиц:

$$(<n_1><v_1>;\ <n_2><v_2>;\ \ldots;\ <n_k><v_k>).$$

В этом наборе n_i есть имена позиций или признаков, а v_i — значения в этих позициях или значения признаков. В нашем случае $k=4$; n_1 — ФИО; n_2 — год рождения; n_3 — пол и n_4 — специальность. Запись вида

(<ФИО> <Иванов>; <год рождения> <1934>; <пол> <м>; <специальность> <слесарь>)

отражает некоторый конкретный спецификационный список. На любое место v_i может быть поставлен специальный символ ?, который означает вопрос о значениях соответствующего признака. Если на вход системы поступает спецификационный список вида (<ФИО><?>; <год рождения><λ>; <пол><λ>; <специальность> <слесарь>),

то это соответствует тому, что системе задан вопрос: «Назови все фамилии слесарей». Знак λ есть знак неопределенности или безразличности. Отвечая на поставленный вопрос, система не будет сообщать пользователю возраст и пол слесарей, имеющихся в списке, хранимом в M.

Существует много диалоговых систем, в которых общение пользователя с системой основано на более или менее сложных спецификационных списках. Иногда в этих системах имеется и прагматический этап. В этих случаях вместо v_i разрешается в вопросах подставлять некоторые предикаты. Например, запрос может выглядеть в виде следующего спецификационного списка:

(<ФИО><?>; <год рождения><>1930>; <пол><м>; <специальность><токарь>).

Смысл этого запроса состоит в выдаче пользователю списка всех токарей мужчин, год рождения которых превышает 1930. Ясно, что если в первом случае поиск ответа происходил путем последовательного сравнения последней позиции спецификационного списка запроса с последними позициями спецификационных списков, хранящихся в M и выдачи значений первой позиции пользователю при совпадении содержимого четвертых позиций, то в рассматриваемом сейчас случае дело обстоит сложнее. Для каждого спецификационного списка, хранящегося в M, для которого содержимое третьей и четвертой позиций совпадает с содержимым этих позиций в спецификационном списке запроса, необходимо еще проверить условие, сформулированное во второй позиции запроса. Эта проверка требует производства арифметической операции вычитания с последующей оценкой знака результата, т. е. представляет собой некоторую вычислительную процедуру. Она и реализуется на прагматическом этапе работы системы.

В еще более сложных случаях в запросе могут быть сформулированы некоторые логические условия относительно интересующих пользователя специалистов. Примером такого запроса может служить: «Найти всех специалистов, которые либо являются токарями и моложе 25 лет, либо являются фрезеровщиками мужчинами». Запросы такого типа потребуют от системы наличия специальных процедур обработки сложных запросов, которые необходимо реализовать в модели мира. Усложнение систем подобного типа приводит к появлению *реляционных баз* информации и реляционных языков представления для них. В гл. 2 рассмотрим их более подробно.

Однако диалоговые системы рассмотренного типа оказались слишком жестко устроенными. Более гибкими ока-

запись системы, в которых в лингвистическом процессоре используется идея *дескрипторов*. В таких системах реализован усеченный этап морфологического анализа, а синтаксический анализ полностью отсутствует. Роль дескрипторов выполняют основы словоформ, список которых изначально задан. Роль морфологического анализа сводится к поиску вхождений этих основ во вводимом текстом. После нахождения дескрипторов происходит переход к модельным представлениям.

Проиллюстрируем этот принцип на том же примере, который использован для спецификационных списков. Предположим, что в словаре дескрипторов содержатся следующие единицы: ПЕРЕЧ, ФАМ, ГОД, ПОЛ, ЖЕН, МУЖ, СЛЕС, ТОК, ФРЕЗ. Кроме того, в этом списке содержатся все числа от 1900 до 1970. Пусть на вход системы поступил запрос: «Перечислите всех слесарей 1950 года рождения». В блоке морфологического анализа происходит пословная обработка запроса, в результате которой в словах находятся основы-дескрипторы, имеющиеся в списке. В результате этой работы будет получен следующий текст: ПЕРЕЧ СЛЕС 1950 ГОД. Эта запись легко преобразуется в спецификационный список:

(<ФИО><?>; <год рождения><1950>; <пол><λ>; <специальность><слесарь>)

По этому спецификационному списку и формируется ответ. Заметим, что использование дескрипторов предполагает наличие некоторых процедур, связанных с учетом местоположения дескрипторов в фразе, а также учетом наличия или отсутствия некоторых дескрипторов. И, конечно, качество функционирования диалоговых систем подобного типа почти целиком определяется имеющимся в распоряжении системы списком дескрипторов.

Следующим шагом на пути усложнения диалоговых систем является введение отношений между дескрипторами. Такой путь приводит к *тезаурусным системам*, в которых кроме морфологического этапа, характерного для дескрипторных систем, появляется и синтаксический этап. Имеющиеся сейчас тезаурусные системы, как правило, допускают отношения между дескрипторами лишь определенного типа, чаще всего родо-видовые.

В тезаурусных системах априорно задается граф связи дескрипторов. При наличии только родо-видовой связи граф этот имеет вид классификационного дерева. На самом верху его находится вершина, отмеченная дескриптором, являющимся наиболее общим для данной предметной области, а ниже по ярусам располагаются дескрипторы,

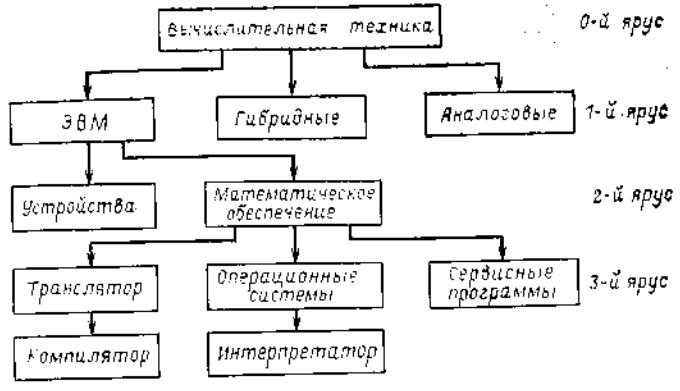


Рис. 1.8

относящиеся к все более и более узким подобластям. На рис. 1.8 показан фрагмент такого дерева для проблемной области «Вычислительная техника».

С каждым уровнем тезауруса связаны некоторые реляторы этого уровня. С нулевым уровнем нашего дерева может быть связан, например, релятор *в общем плане обсуждаются вопросы* или релятор *использование в народном хозяйстве*. К первому уровню относятся реляторы типа *инженерные решения, программное обеспечение, комплекс программ*. Кроме таких реляторов, которые являются указателями ярусов, могут быть и другие реляторы, которые связывают между собой отдельные дескрипторы или группы дескрипторов (поддеревья). Примерами реляторов второго типа могут служить: *влияет на сложность реализации, вызывает необходимость в изменении* и т. п. Анализируя словоформы в поступающем на вход тексте, морфологический блок выделяет среди них дескрипторы и реляторы, а синтаксический блок на основании выделенных единиц строит граф, на основе которого в модели мира вычленяется подобласть, о которой идет речь. Тезаурусные системы широко применяются в различных информационно-поисковых системах. Но для развитых диалоговых систем АСУ они обладают все еще слишком ограниченными возможностями. Эти возможности дают лишь лингвистические процессоры с хорошо развитым этапом семантического анализа. Именно поэтому основное внимание будет уделяться семантическому этапу и принципам построения языка представления знаний.

Всякая диалоговая система, базирующаяся на естественном языке, предназначается для решения каких-то

определенных прикладных задач. Поэтому ее можно представить в виде композиции двух основных программных модулей: языкового и прикладного. Задачей первого является перевод исходного текста на язык, доступный прикладному модулю, и обратный перевод, а задачей второго — выполнение тех процедур, которые необходимы пользователю системы. Наличие двух таких модулей приводит к существованию двух подходов к построению диалоговых систем:

разработка языкового модуля ведется под определенный прикладной модуль, решающий ограниченный класс задач, например поиск информации в базе данных заранее определенного типа или управление роботом-манипулятором и т. д.;

разработка универсального языкового модуля, использование которого возможно с достаточно широким классом прикладных модулей.

При первом подходе естественный язык, обрабатываемый языковым модулем, весьма жестко ограничивается (с содержательной точки зрения) кругом задач, решаемых прикладным модулем, что приводит к возможности задания однозначных семантических представлений для каждой лексической единицы входного языка и позволяет получать простые алгоритмы анализа, основывающиеся на этих представлениях. Грубо говоря, в этом случае языковый модуль должен выделять из входных текстов те и только те сведения, которые релевантны ограниченному набору входных информационных структур прикладного модуля. Такой подход характерен для узко специализированных систем, примером которых может являться система ДВОЭ, которая будет описана в гл. 3, или система ДИСПУТ, описанная в гл. 4. Достоинством такого подхода помимо прозрачности алгоритмов лингвистического модуля является малый объем знаний, необходимых для анализа языка, и, в частности, бедность семантического этапа обработки языковой информации. Прагматический аспект в таких системах настолько силен, что в языковом модуле по существу почти не используются этапы морфологического и семантического анализа. Но за это приходится расплачиваться специализацией языковых модулей. Обычно они тесно связаны с прикладными модулями и использовать их для каких-либо иных прикладных модулей просто невозможно.

Примерами систем, разработанных в соответствии со вторым подходом, могут служить система ДИЛОС, рассматриваемая в гл. 6, и система МИВОС, которой посвящена гл. 5. В системе ДИЛОС, например, языковый модуль содержит лингвистический, информационный, логический и вычислительный процессоры. Характерной особенностью системы ДИЛОС является наличие развитого языка семантического представления, который предназначен не только для отображения семантики и прагматики текстов, но также для организации поиска информации, проведения операций обобщения и классификации данных.

В отличие от этого в системе МИВОС язык семантического представления, а также внутренняя структура отдельных компонентов языкового модуля заранее не определяются. Основная особенность системы МИВОС, как будет видно из дальнейшего, заключается в создании универсальной метаязыковой среды — системы программирования, содержащих средства построения, поддержки (сопровождения) и реализации языковых модулей для произвольных прикладных модулей.

ГЛАВА ВТОРАЯ

ПРЕДСТАВЛЕНИЕ ЗНАНИЙ В ДИАЛОГОВЫХ СИСТЕМАХ

2.1. Классификация языков представления знаний

В § 1.1 были сформулированы основные требования к языкам смыслов $Я_с$. Их полная реализация, к сожалению, на сегодняшнем уровне развития диалоговых систем невозможна. Связано это, в частности, с тем, что в самих этих требованиях кроется некоторая противоречивость. С одной стороны, от $Я_с$ требуется, чтобы он содержал в себе некоторую формальную систему (в частности, систему формальных эквивалентных преобразований). С другой стороны, требуется некий гомоморфизм между естественным языком и $Я_с$, хотя естественный язык, по мнению всех специалистов, работающих с ним, есть объект гораздо более широкий, чем формальная система. Эта двойственность приводит к тому, что выбор того или иного варианта языка $Я_с$ является компромиссом между различными требованиями к его структуре. Интерес представляют три группы $Я_с$, каждая из которых обладает с точки зрения требований, рассмотренных в § 1.1, своими преимуществами и недостатками.

В первую группу входят *логические языки*. Эти языки используют для своего определения формальную систему логического типа. В качестве такой системы можно

использовать, например, исчисление высказываний или исчисление предикатов первого порядка, многозначные логики или модальные исчисления. В любом случае постулируется, что в основе логического языка лежит некоторая фиксированная формальная система. Другими словами, формализованы синтаксис и семантика языка $Я_c$. При этом синтаксис задается набором правил построения *правильных синтаксических выражений,* обладающих разрешающей процедурой. Это означает, что для любого выражения в данном языке эти правила однозначным образом и за конечное число шагов определяют, является ли это выражение синтаксически правильным или не является. Семантика языка логического типа задается набором правил преобразования выражений и разрешающей процедурой, позволяющей однозначным образом и за конечное число шагов определить, является ли данное выражение $Я_c$ *семантически правильным.* Поясним эти важные положения более подробно.

Дадим сначала общее определение формальной системы. Зададим произвольное множество элементов, которые будем называть *термами,*

$$T=\{t_1, t_2, \ldots, t_n \ldots\}.$$

Относительно множества T будем предполагать существование процедуры Θ, которая эффективно определяет принадлежность некоторого элемента t множеству T и отождествляет его (если t принадлежит T) с одним из элементов T. Кроме того, для любой пары элементов из T процедура Θ определяет, совпадают ли они между собой или не совпадают. Природа термов может быть произвольной. В качестве t_i могут выступать, например, буквы (графемы) русского алфавита, отдельные слова языка, иероглифы и т. п.

Определим правила P, с помощью которых из термов можно образовывать некоторые совокупности. Все совокупности, которые получаются из T с помощью правил P, будем называть *синтаксически правильными,* а сами правила P — *синтаксическими правилами.* Как уже говорилось, P должны быть устроены таким образом, чтобы существовала разрешающая эффективная процедура Π, которая позволяла бы для любой совокупности термов τ определять, принадлежит ли или не принадлежит τ к $\mathscr{F}$ ($\mathscr{F}$ — это множество всех совокупностей термов, получающихся из T с помощью P).

Выделим в $\mathscr{F}$ произвольным образом подмножество $\mathscr{F}' \subseteq \mathscr{F}$. Элементы $\mathscr{F}'$ назовем *априорно осмысленными*

(*аксиомами*). Зададим теперь правила Q, с помощью которых из одних элементов множества $\mathscr{F}$ можно получать другие элементы множества $\mathscr{F}$. Правила эти будем называть *семантическими правилами* или *правилами вывода.* Как и для синтаксических правил, для правил Q должно выполняться условие, что они позволяют построить эффективную разрешающую процедуру Ξ, с помощью которой для любой совокупности термов, входящей в множество $\mathscr{F}$, можно сказать, принадлежит ли эта совокупность к подмножеству $\mathscr{F}''$ или не принадлежит. Подмножество $\mathscr{F}''$ (см. рис. 2.1) образуется при всевозможных применениях правил вывода к элементам подмножества $\mathscr{F}'$.

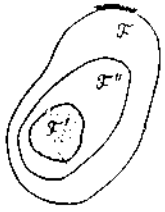


Рис. 2.1

Формальной системой назовем четверку

$$M=\langle T, P, \mathscr{F}', Q\rangle,$$

а *эффективной формальной системой* — семерку

$$M^э=\langle T, \Theta, P, \Pi, \mathscr{F}', Q, \Xi\rangle.$$

Примерами подобных формальных систем могут служить исчисление высказываний или грамматики Хомского. В случае исчисления высказываний множество термов может быть задано в виде множества прописных латинских букв с произвольными целочисленными индексами. Синтаксические правила позволяют образовывать из этих термов правильно построенные формулы, и процедура Π есть просто процедура проверки соответствия правил P записи этих формул (например, легко устанавливается, что формула $a \vee \bar{b}$ правильно построена, а $a \vee b$ не принадлежит к множеству $\mathscr{F}$). В качестве аксиом можно использовать некоторый набор правильных формул и задать конечный набор правил вывода для них. Тогда получится формальная система (может быть, и неэффективная). В случае грамматик Хомского словари терминальных и нетерминальных символов образуют множество T, правила P задаются операцией приписывания, позволяющей образовывать из термов слова различной длины. Процедура Π есть простая проверка вхождения в слово элементов только из T. Она работает совместно с процедурой Θ. В грамматиках Хомского система аксиом обычно содержит одну фиксированную аксиому S (специальный терм из нетерминального словаря), а правила вывода задаются системой подстановок.

Важным понятием, связанным с формальными системами, является *интерпретация*. Для интерпретации задается множество Z конечное или бесконечное, а также специальная процедура Ψ, позволяющая отображать элементы из множества $\mathcal{F}$ на множество Z при условии, что задано отображение множества T для термов, входящих в интересующую нас совокупность из $\mathcal{F}$, на Z. Это отображение задается некоторой процедурой χ, реализация которой происходит вне воли исследователя. В классических логических исчислениях в качестве Z используется двухэлементное множество {ИСТИНА, ЛОЖЬ}. Наличие процедуры Ψ в этих исчислениях позволяет для каждой правильно построенной формулы при заданной интерпретации входящих в нее термов получить интерпретацию всей формулы. Например, для формулы $ac \vee b\bar{c} \vee bc$, если a и l интерпретируются как ИСТИНА, а c — как ЛОЖЬ, интерпретацией всей формулы является ИСТИНА.

В классических логических исчислениях множество $\mathcal{F}'$ выбирается таким образом, что интерпретация формул из $\mathcal{F}'$ не зависит от интерпретации термов, в них входящих. Другими словами, при любой интерпретации входящих в них термов сами формулы интерпретируются однозначно. Правила вывода выбираются таким образом, что их применение не нарушает интерпретацию формул. При выполнении подобных условий становится возможным построение эффективной процедуры Ξ. Для классического исчисления высказываний, например, можно взять в качестве аксиом только тождественно истинные формулы. Тогда если правила вывода выбраны подходящим образом, то множество $\mathcal{F}''$ содержит лишь тождественно истинные формулы. Если при этом множество аксиом таково, что для этих правил вывода оно порождает все допустимые в данной системе тождественно истинные формулы, то процедура Ξ сводится лишь к проверке тождественной истинности той формулы, чья принадлежность к $\mathcal{F}''$ исследуется.

После этого отступления вернемся к классификации языков представления знаний. В известных версиях таких языков логического типа использовалось либо исчисление высказываний, либо исчисление предикатов первой степени. Выбор именно таких формальных систем обусловлен тем фактом, что для этих исчислений построены эффективные процедуры Ξ, а для более сложных формальных логических систем надежды на построение эффективных процедур Ξ малы. Во всяком случае, показано, что уже для исчисления предикатов второй степени подобной эффективной процедуры принципиально построить невозможно.

Из этого следует, что языки $Я_c$ логического типа не обладают хорошими изобразительными возможностями для передачи всех оттенков и тонкостей смысла естественных языков. Их семантика слишком «жестка» для богатых естественных языков. Но их преимуществом является способность этих языков к построению эффективной системы эквивалентных преобразований выражений. Именно поэтому подобные языки все еще находят применение в диалоговых системах, особенно в узко профессионально ориентированных, для которых входной язык представляет собой небольшое подмножество всего естественного языка. В следующем параграфе мы отметим особенности использования логических языков в диалоговых системах.

Второй тип языков можно назвать *реляционными языками*. Для них характерно введение конечного множества бинарных отношений R, с помощью которых передаются смысловые связи между элементами языка. Геометрической моделью для записей в реляционных языках служат *семантические сети*. Вершины в этих сетях отождествляются с элементами языка, а дуги — с бинарными отношениями, существующими между этими элементами. Обычно рассматривают языки, в которых априорно задаются конечные множества элементов языка X и конечное множество R. Такие (X, R)-языки, конечно, не могут отражать все возможности естественных языков, но для профессионально ориентированных систем оказываются вполне пригодными. Математической моделью реляционных языков является алгебраическое понятие модели с носителем X и сигнатурой R. Теория таких моделей может рассматриваться в качестве теории реляционных языков. Фрагмент такой теории создал Кодд, и мы чуть позже вернемся к этому вопросу.

Реляционные языки по своей структуре значительно ближе к естественным языкам, чем логические языки. Это приводит к тому, что процедуры семантического этапа при переводе текстов на язык $Я_c$ и обратного перевода получаются для этого случая более простыми, чем аналогичные процедуры для логических языков. Но с точки зрения эквивалентных преобразований в реляционных языках нет возможности построить систему процедур, относительно которой можно было бы, как это сделано для исчисления высказываний или исчисления предикатов, доказать ее полноту и эффективность.

Третья группа языков представления знаний опирается на специальные конструкции, называемые *ролевыми фреймами*. Поэтому языки этой группы в дальнейшем будем называть *ролевыми*. С точки зрения лингвистики ролевые языки представляют собой описания, в которых в явном виде используются так называемые глубинные падежи Филмора или аналогичные им средства. В ролевых языках каждая единица в предложении описывается через те семантические роли, которые она может выполнять. Примерами таких ролей могут служить: субъект действия, объект действия, препятствие, ресурс, орудие и т. п. В последнее время ролевые языки начинают активно внедряться в практику построения диалоговых систем. Объясняется это тем, что представления, используемые в ролевых языках, оказываются удобными для обработки информации в современных ЭВМ и для хранения их в списочных структурах памяти. Кроме того, ролевые языки несколько богаче по своим возможностям, чем обычные реляционные языки. Недостатком ролевых представлений является сложность введения в них эффективных процедур эквивалентных преобразований. Эта проблема до сих пор для таких языков не решена. Не развита и математическая теория языков подобного типа.

2.2. Языки предикатного типа

Среди языков логического типа рассмотрим лишь языки, в основе которых лежит исчисление предикатов первого порядка. Связано это с тем, что только такие языки нашли применение в диалоговых системах, базирующихся на естественном языке. Наиболее успешно *предикатные языки* применяются в тех случаях, когда с их помощью представляются математические тексты. В качестве примера рассмотрим предикатный язык, использованный в свое время для описания фрагмента школьной планиметрии.

Пример 2.1. Заданы два словаря: словарь предметных областей переменных и словарь предикатов. Первый словарь включает 20 элементов, примерами которых могут служить: *п* — предмет, *тчк* — точка, *пр* — прямая, *уг* — угол, *трк* — треугольник и т. п. Список предикатов содержит 100 элементов, примерами которых могут служить: *ПР* — быть прямым, *РАВН* — быть равным, *ПРИН* — принадлежать, *РАВНБ* — быть равнобедренным, *БИС* — быть биссектрисой, *ГУГ* — быть главным углом треугольника, *МЕД* — быть медианой, *ВЫС* — быть высотой. Кроме этих словарей заданы обычные логические связки: конъюнкция, дизъюнкция, импликаций, эквивалентность и отрицание. Заданы также кванторы общности $\forall$, существования $\exists$ и специальные кванторы существования: $\exists_n$ — «существует ровно n таких, что»; $\exists_{>n}$ — «существует более чем n таких, что»; $\exists_{>n}$, $\exists_{<n}$, $\exists_{\leqslant n}$ (семантика этих трех кванторов строится аналогично первым двум). Кроме того, используются при записи круглые скобки, разделитель в виде занятой и всевозможные индексы. Приписывание индекса любой предметной области переменных есть означивание ее. Таким образом, *трк* означает любой треугольник, а $трк_{АВС}$ — некоторый треугольник *АВС*. Приведем примеры записи текстов на введенном предикатном языке:

1. *Все прямые углы равны между собой.* Запись этого текста имеет следующий вид:

$$\forall уг_A \, \forall уг_B \, ((уг_A ПР) \& (уг_B ПР) \rightarrow (уг_A РАВН \, уг_B)).$$

2. *Через всякие две не совпадающие точки можно провести прямую и притом только одну.* Соответствующая запись:

$$\forall тчк_A \, \forall тчк_B \, (\Gamma (тчк_A РАВН \, тчк_B) \rightarrow \exists_1 пр_C \, ((тчк_A ПРИН \, пр_C) \& (тчк_B ПРИН \, пр_C))).$$

3. *В равнобедренном треугольнике биссектриса угла при вершине является одновременно медианой и высотой.* Запись на предикатном языке имеет следующий вид:

$$\forall трк_{АВС} ((трк_{АВС} РАВНБ) \& \forall п_X \forall п_Y ((п_X БИС \, п_Y) \& (п_Y ГУГ \, трк_{АВС}) \rightarrow (п_X МЕД \, трк_{АВС}) \& (п_X ВЫС \, трк_{АВС}))).$$

Отметим одну интересную особенность записей на предикатном языке. Если произвести обратный перевод, т. е. перейти от предикатной записи к тексту на естественном языке, то получим не те тексты, которые написаны в приведенных примерах, а несколько иные тексты. Например, при обратном переводе записи второго примера получим такой текст: «Для любой точки *А* и любой точки *В*, если точка *А* не совпадает с точкой *В*, существует единственная прямая *С*, которой принадлежат и точка *А*, и точка *В*». Полученный текст, не совпадая с исходным, тождествен ему по смыслу. Это означает, что при переводе с естественного языка на предикатный язык два указанных текста должны либо превратиться в одну и ту же предикатную запись, либо в записи, эквивалентные с точки зрения исчисления предикатов. Другими словами, так как исчисление предикатов в нашем случае вы-

ступает в качестве языка смыслов, то необходимо уметь организовать процедуры отождествления различных синтаксических текстов, обладающих одинаковым смыслом (см. § 1.1).

Обозначим через H множество *трансформационных преобразований* над предложениями естественного языка. Каждое такое преобразование меняет синтаксис предложения, но не меняет его смысл. При переводе предложений из N, где N есть совокупность всех предложений, получающихся друг из друга с помощью преобразований из H, в предикатные представления будем получать различные записи. Для их преобразования друг в друга можно воспользоваться имеющейся в исчислении предикатов первого порядка системой эквивалентных преобразований E. Приведем эту систему:

1. $f * \varphi \Longleftrightarrow \varphi * f$, где $*$ есть $\&, \vee$ или $\equiv$.

2. $f * (\varphi * \psi) \Longleftrightarrow (f * \varphi) * \psi$, где $*$ есть $\&$ или $\vee$.

3. $\neg (f * \varphi) \Longleftrightarrow \neg f * \neg \varphi$, где $*$ есть $\&$ или $\vee$, а $*$ $\vee$ или $\&$.

4. $f \to (\varphi \to \psi) \Longleftrightarrow (f \& \varphi) \to \psi$.

5. $(f \& \varphi) \vee (f \& \psi) \Longleftrightarrow f \& (\varphi \vee \psi)$.

6. $(f \to \varphi) * (f \to \psi) \Longleftrightarrow f \to (\varphi * \psi)$, где $*$ есть $\&$ или $\vee$.

7. $(f \to \varphi) \& (f \to \varphi) \Longleftrightarrow f \to (\varphi \vee \psi)$.

8. $\mathfrak{K}^1_x \mathfrak{K}^2_y S(x, y) \Longleftrightarrow \mathfrak{K}^1_{xy} S(x, y)$, где $\mathfrak{K}^1_x$ и $\mathfrak{K}^2_x$ — одноименные кванторы

9. $\neg \exists_x S(x) \Longleftrightarrow \forall_x \neg S(x)$.

10. $\mathfrak{K}_x (S * U(x)) \Longleftrightarrow S * \mathfrak{K}_x U(x)$,

где $\mathfrak{K}$ — произвольной квантор, S — формула, не зависящая от x, а $*$ есть $\&, \vee$ или $\to$.

11. $\forall_x (S(x) \to U) \Longleftrightarrow \exists_x S(x) \to U$, где U есть формула, не зависящая от x.

12. $\forall_x (S(x) \& U(x)) \Longleftrightarrow \forall_y S(y) \& \forall_z U(z)$.

Другими словами, x можно всюду в $S(x)$ заменить на y, а x в $U(x)$ можно всюду заменить на z.

13. Введение и элиминирование предметных областей. Это преобразование основано на том, что любые предметные области (*тчк*, *трк* и др.) могут быть заменены на предметную область «предмет». Например, высказывания: «Сумма двух нечетных чисел есть четное число» и «Предмет, являющийся суммой двух предметов, каждый из которых есть нечетное число, есть четное число» имеют одинаковый смысл.

14. Квантор $\exists_{>1}$ может быть заменен на квантор $\exists$. Это следует из того, что высказывания типа: «Существует по крайней мере один x такой, что...» и «Существует такой x, что...» по смыслу совпадают.

Указанные 14 схем преобразований образуют систему E. Пусть имеем некоторое предложение естественного языка α. Через N_α обозначим множество всех предложений, получающихся в этом естественном языке из α с помощью трансформационных преобразований из H. Предположим, что процедура перевода, имеющаяся в некоторой диалоговой системе, преобразует α в предикатную формулу S_α. Применим к S_α всеми возможными способами преобразования из E. Пусть L_α есть множество предикатных формул, полученных в результате этой процедуры. Формальная система будет эффективной, если все предложения из N_α отображаются на семантическом этапе в формулы из L_α, а предложения, не принадлежащие N_α, не отображаются в формулы из L_α. Возникает вопрос: верно ли это утверждение? Ответ на него связан с анализом множества H, т. е. с анализом полноты системы трансформационных преобразований. Для русского языка попытки построения системы H делались неоднократно. Для примера приведем ряд трансформационных правил, обычно включаемых в подобные системы.

1. Введение имени деятеля. С помощью этой трансформации предложение типа: «Всякий предмет, являющийся диагональю ромба, является его осью симметрии» преобразуется в предложение: «Всякая диагональ ромба является осью его симметрии». При этом, конечно, допустима и обратная трансформация.

2. Введение лексемы *между собой*. С помощью этой трансформации предложение типа: «Отрезки AB и CH равны» преобразуется в предложение: «Отрезки AB и CH равны между собой». Допустима и обратная трансформация.

3. Введение грамматических отрицаний. Это правило позволяет произвести преобразование предложения типа: «Неверно, что всякий ромб является квадратом» в предложение: «Не всякий ромб является квадратом». Верна и обратная трансформация.

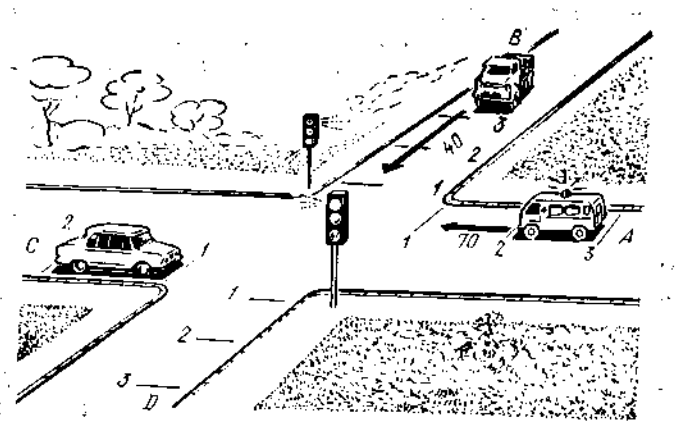


Рис. 2.2

4. Введение кванторных прилагательных. Это правило трансформации можно проиллюстрировать на паре предложений: «Существует такое число, что всякое число имеет его своим делителем» и «Всякое число имеет своим делителем некоторое одно и то же число».

Число правил, входящих в состав H, невелико. Во всяком случае, оно не превышает двух-трех десятков. Поэтому использование системы H достаточно эффективно и вполне может служить для тех целей, которые сформулированы в данном параграфе.

Пример 2.2. Для иллюстрации выразительной силы предикатных языков рассмотрим еще один пример записи текстов на языках такого типа. Он будет относиться к описанию ситуаций, складывающихся в окрестности уличного перекрестка. На рис. 2.2 показана некоторая конкретная ситуация, сложившаяся на перекрестке. Она может быть описана в виде следующего текста на естественном языке: «Красный сигнал светофора горит по направлению *C-A*. Легковой автомобиль стоит в ожидании начала движения на линии СТОП (сечение *1* на рис. 2.2) на направлении *C*. По направлению *A* к перекрестку приближается микроавтобус, снабженный сигналом «Скорая помощь». В данный момент он находится на сечении *2*, а его скорость равна 70 км/ч. В направлении *B* к перекрестку со скоростью 40 км/ч приближается грузовой автомобиль, находящийся в данный момент на сечении *3*. Направление *D* от автотранспорта свободно».

Введем следующие предикаты: $Q(x, y)$ — автомобиль x имеет тип y. Переменная x определена на множестве натуральных чисел, интерпретируемых как имена автомобилей, присваиваемые им в окрестности перекреста управляющей системой. Переменная y принимает значения из множества $M_1 = \{$*легковая, общественный транспорт, грузовая, трактор, мотоцикл, специальный автотранспорт*$\}$. Для компактности записи элементы M_1 в дальнейшем будем обозначать как *л, о, с, т, м, с* соответственно. Следующий предикат $R(z)$ — автомобиль имеет скорость z. Переменная z принимает значения из множества $M_2 = \{0, 10, 20, \ldots, 120, 130\}$. Элементы M_2 определяют значение скорости в километрах в час с некоторой фиксированной дискретностью. Следующий предикат $T(w)$ имеет смысл: «автомобиль находится на направлении w», где значения w берутся из множества $M_3 = \{A, B, C, D\}$. Предикат $U(v)$ имеет смысл: «автомобиль находится на сечении v», где v принимает значение из множества $M_4 = \{1, 2, \ldots, l\}$. Величина l определяет число сечений, выделяемых управляющей системой в окрестности перекрестка. Наконец, предикат $P(u)$ истинен тогда, когда в направлении *C-A* горит зеленый сигнал светофора. При использовании введенных обозначений предикатная запись ситуации, показанной на рис. 2.2, будет иметь следующий вид:

$$[Q(1, л) \,\&\, R(0) \,\&\, T(C) \,\&\, U(1)] \,\&\, Q(2, т) \,\&\, R(40) \,\&\, T(B) \,\&\, U(3)] \,\&$$

$$\&\, [Q(3, с) \,\&\, R(70) \,\&\, T(A) \,\&\, U(2)] \,\&\, \neg\, p(u).$$

Особенностью этой записи являются прямоугольные скобки, роль которых сводится к указанию на область действия предиката. Если эти скобки опустить, то станет невозможно определить, к какой конкретно автомашине относится тот или иной предикат. Если все же постараться избегнуть введения скобок, то в предикатах R, T и U надо ввести еще переменную x, привязав соответствующие высказывания к идентификатору автомашины.

Укажем на некоторые трудности, которые вызывает использование предикатных языков при представлении текстов, написанных на естественном языке.

1. В предикатных языках обычного типа невозможно выразить многие квантификаторы естественных языков. Такие квантификаторы, как *только, как правило, много, изредка* и т. п., невыразимы в них.

2. Необходимость всегда точно указывать область действия предиката резко сужает набор текстов на естественном языке, для которых имеется полная аналогия с предикатным представлением.

3. Количество различных предикатов, которое необходимо использовать, весьма велико. Фактически мощность этого множества совпадает по порядку с мощностью словаря языка.

2.3. Языки реляционного типа

Для языков этой группы характерно явное выделение отношений, связывающих между собой те или иные понятия. В качестве примера $Я_6$ такого вида рассмотрим три

языка: *табличные языки Кодда, RX-коды и язык ситуационного управления*. Начнем с описания табличного языка Кодда. Пусть нас интересуют некоторые данные, касающиеся кадрового состава определенной организации. Все сведения об этом введены в память ЭВМ и структурированы в виде таблицы следующего типа:

Таблица 2.1

Тип сущности	Атрибуты		
ФИО	Пол	Год рождения	Специальность
Иванов Иван Иванович	м	1945	Слесарь
Петров Петр Петрович	м	1951	Слесарь
Родина Анна Петровна	ж	1952	Токарь
Смирнов Илья Ильич	м	1922	Фрезеровщик

Сделаем некоторые замечания. В левой колонке таблицы перечислены представители некоторого однородного множества. Это множество есть множество людей, работающих в данное время на данном предприятии. Таблица определяет тип сущности под названием «Штатный список работающих в настоящее время на предприятии». Другими примерами типов сущностей могут служить «комплектность автомобиля», «список обязательных товаров» и т. д. Каждый представитель сущности может идентифицироваться двояко. Его можно выделить с помощью индивидуального имени этого представителя (например, Иванов Иван Иванович) либо перечнем значений атрибутов, соответствующим ему (например, для того же Ивана Ивановича Иванова это будет набор типа <м, 1945, слесарь>). Если на данном предприятии в настоящее время имеются представители сущности с одинаковыми именами (например, работают два Ивана Ивановича Иванова), то их различие может происходить с помощью присущих им перечней атрибутов. Каждую строку таблицы можно задать в виде некоторого фиксированного вектора следующего типа:

$$<i;\ j_1(h_1);\ j_2(h_2);\ \ldots,\ j_m(h_m)>. \qquad (2.1)$$

В соотношении (2.1) i соответствует имени типа сущности, j_k — именам атрибутов, а h_k — значениям атрибутов. Как увидим в дальнейшем, в ряде диалоговых систем (например, в ДИЛОСе, описанном в гл. 6) подобные представления в базе данных используются весьма эффективно.

Представления в виде (2.1) дают возможность задавать вопросы относительно имен, сущностей и атрибутов, а также о значениях атрибутов.

Пример 2.3. При наличии в памяти ЭВМ табл. 2.1 можно сформулировать, например, следующие четыре запроса системе:

1) <?; пол (ж); год рождения (1952); специальность (токарь)>;
2) <?; пол (м); год рождения (λ); специальность (слесарь)>;
3) <Смирнов Илья Ильич; μ_1 (?); μ_2 (?); ...; μ_k (?)>;
4) <?; пол (м); год рождения (?); специальность (токарь)>.

Первый вопрос трактуется как вопрос об имени сущности, которая является женщиной 1952 г. рождения и имеет специальность токаря. Простая процедура поиска по совпадению всех данных, имеющихся в запросе, приводит к тому, что система в качестве ответа выдает: Родина Анна Петровна. Второй запрос содержит специальную переменную λ. Смысл ее состоит в указании на незначимость значения атрибута «год рождения». Другими словами, от системы требуется перечислить всех мужчин слесарей, не обращая внимания на их возраст. В нашем случае в ответ система должна указать, что имеется две сущности Иванов Иван Иванович и Петров Петр Петрович, которые удовлетворяют запросу. В третьем запросе указано имя сущности, а на остальных местах стоят μ_i (?) и k не фиксировано. Суть этого запроса может быть передана словами: «Сообщите все, что знает система о Смирнове Илье Ильиче». Ответом служит последняя строка из табл. 2.1. Наконец, в четвертом запросе содержатся сразу два вопроса, и словесно он может быть интерпретирован так: «Укажите имена сущностей и года их рождения для всех токарей мужчин». На этот запрос в нашем конкретном случае система должна ответить, что таких сущностей она не знает.

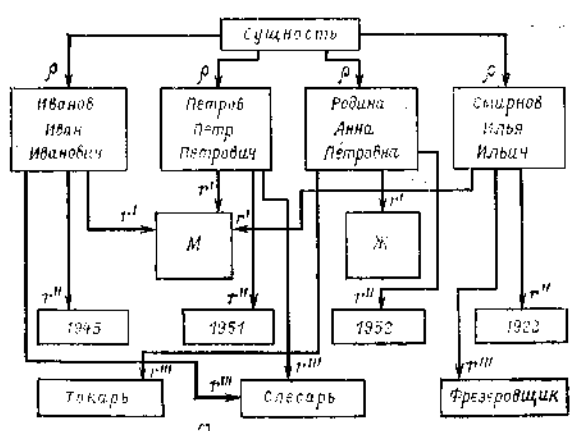


Рис. 2.3

(Таблицы, подобные табл. 2.1, формально задают на множестве $H=I\times H_1\times \ldots \times H_k$, где H_i есть множество значений атрибута j_i, а I — множество имен сущностей (l-арные ($l\leqslant k$) отношения). Они могут быть сведены к совокупности бинарных отношений, хотя при этом часть информации, непосредственно содержащейся в таблице, может в явном виде не представляться).

Пример 2.4. На рис. 2.3 показана совокупность бинарных отношений, отражающая связи, имеющиеся в табл. 2.1 между сущностями и значениями атрибутов. Сами атрибуты при этом начинают играть роль имен бинарных отношений. В рассматриваемом примере ρ — иметь имя, r' — иметь пол, r'' — родиться в, r''' — иметь специальность. Если в вершину дерева поместить не *сущность*, а, например, *специальность*, то дерево перестроится и будет иметь такой вид, как показано на рис. 2.4.

В этом случае отношение r'''' трактуется как *иметь сущность* и наблюдается неоднозначность в связях. Например, отношение r'' для вершины *слесарь* двузначно.

Для отношений можно построить алгебраическую теорию *реляционных представлений* (от английского relation — отношение). Эта теория позволяет строить единое алгебраическое представление для таблицы, а не совокупность отдельных представлений, как это было в примере 2.4. Кроме того, в рамках общей теории удается ввести над представлениями обычные алгебраические операции типа пересечения, объединения и дополнения, ставить задачу эквивалентных преобразований и т. д. Другими словами, удается создать специальные табличные реляционные языки, которые часто называют табличными языками Кодда. Можно показать, что эти языки эквивалентны языку исчисления предикатов первого порядка.

Естественным расширением возможности таких языков являются языки, в которых отношения заданы в явной форме.

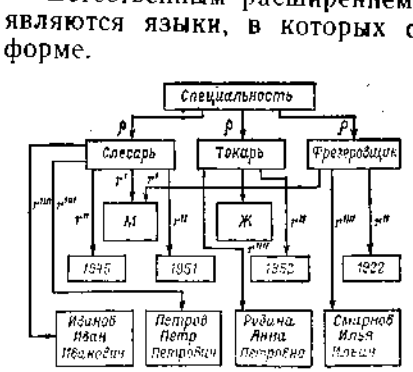


Рис. 2.4

Рассмотрим языки, в которых выделены два типа объектов: *предметы* и *отношения*. Предметы будем обозначать буквами X с различными верхними и (или) нижними индексами, а отношения — буквами R с различными нижними индексами. В качестве правильных синтаксических выражений (см. § 2.1) в языке будем использовать пары $R_iX^m{}_j$, любые конъюнкции этих пар (знак конъюнкции будем опускать) и конъюнкции, получающиеся при замене любого элемента X_j в правильном синтаксическом выражении на правильно построенное выражение. Эти правила есть синтаксические правила нашего языка.

Пример 2.5. Пусть множество предметов состоит из X_1 и X_2, а множество отношений — из R_1 и R_2. Тогда синтаксически правильными выражениями будут, например, R_2X_1; $R_2X_2R_1X_2R_2X_1$. Синтаксически правильным выражением будет и конъюнкция вида

$$R_2(R_1X_1R_1X_2)R_1(R_1X_1R_1X_2)R_2X_1,$$

полученная из конъюнкции $R_2X_2R_1X_2R_2X_1$ заменой X_2 на правильно построенное выражение $R_1X_1R_1X_2$.

Предметам и отношениям жестко приписываются некоторые семантические значения (интерпретации в терминах § 2.1). В этом случае оказывается возможным производить определение новых понятий через ранее определенные понятия, которые для этого нового понятия выступают в роли определяющих признаков.

Пример 2.6. Пусть R_1 есть отношение *быть элементом класса*, а R_2 — *включать в качестве своего элемента*. Пусть также $X^1{}_7$ — *летательный аппарат тяжелее воздуха*, $X^1{}_8$ — *двигатель, создающий тягу*, и $X^1{}_9$ — крыло. Тогда запись $X^2{}_{10}=R_1X^1{}_7R_2X^1{}_8R_3X^1{}_9$ дает определение предмета $X^2{}_{10}$, которое в словесной форме выглядит так: *Летательный аппарат тяжелее воздуха, включающий в себя двигатель, создающий тягу, и крыло*. Это определение можно рассматривать как определение понятия *самолет*. Сами понятия $X^1{}_7$, $X^1{}_8$ и $X^1{}_9$ можно также задать в виде некоторых записей, представляющих собой правильно построенные выражения. Пусть $X^0{}_1$ есть понятие *летательный аппарат*, а $X^0{}_2$ — *аэродинамическая сила*. И пусть отношение R_3 есть *использовать свойство*. Тогда $X^1{}_7$ может быть определено как $X^1{}_7=R_1X^0{}_1R_3X^0{}_2$. Если X^0 это *силовая установка*, $X^0{}_4$ — *тяга*, R_4 есть отношение *давать*, то $X^1{}_8$ можно определить как $X^1{}_8=R_1X^0{}_3R_4X^0{}_4$. Если, наконец, $X^0{}_5$ это *неподвижная относительно целого несущая поверхность*, $X^0{}_6$ — *подъемная сила*, то определение понятия *крыло* можно дать в следующей форме: $X^1{}_9=R_1X^0{}_5R_4X^0{}_6$. Подставляя в определение самолета вместо соответствующих понятий их определения, получаем определение самолета через совокупность более частных признаков:

$$X^2{}_{10}=R_1(R_1X^0{}_1R_3X^0{}_2)R_2(R_1X^0{}_3R_4X^0{}_4)\ R_2(R_1X^0{}_5R_4X^0{}_6).$$

Из приведенного примера виден смысл верхних индексов у понятий. Эти индексы показывают уровень обобщения понятия, позволяют вводить иерархию понятий и их классификацию.

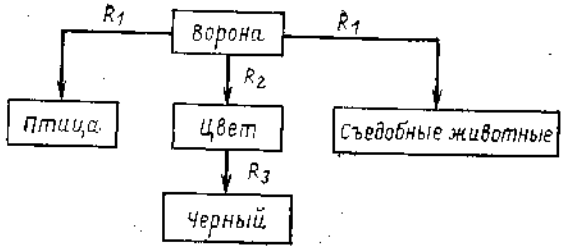


Рис. 2.5

Языки описанного типа получили название *RX*-кодов. Они с успехом используются в информационно-поисковых системах, для которых родо-видовые отношения играют весьма существенную роль. Тогда естественная иерархия, присущая языку *RX*-кодов, позволяет эффективно организовывать поисковые процедуры. Однако у *RX*-кодов есть естественные ограничения. Они хорошо описывают лишь древовидные структуры связей. Определяемое понятие всегда является корнем этого дерева, а связи между понятиями одного уровня не учитываются.

Пример 2.7. На рис. 2.5 показана структура определения понятия «ворона». Отношение R_1 есть *быть элементом класса*, отношение R_2 — *иметь цвет*, а отношение R_3 — *признак — значение*. При определении понятия «ворона» предполагается, что понятия «птица», «цвет», «съедобные животные» определены ранее. Между этими понятиями, возможно, и существуют какие-то отношения, но в структуре определения вороны они не учитываются.

Однако для текстов на естественном языке древовидные структуры отношений не столь характерны. В общем случае системы отношений образуют произвольную сеть, в которой могут встречаться циклы любого ранга, содержащие произвольное конечное число отношений. Поэтому естественным расширением возможностей языков типа *RX*-кодов явились *языки синтагматических цепей*. В качестве примера такого языка рассмотрим язык ситуационного управления.

Введем множество термов, включающее в себя три класса: класс понятий $X=\{x_1, x_2, \ldots, x_n \ldots\}$, класс имен $I=\{i_1, i_2, \ldots, i_m \ldots\}$, класс отношений $R=\{\rho, r_1, r_2, \ldots, r_k\}$. Классы понятий и имен считаются конечными или счетными, а класс отношений — конечным. Отношение ρ играет роль отношения *иметь имя*. С помощью синтаксических правил из этих термов можно образовывать правильно

построенные выражения. Элементарным выражением такого типа являются тройки $(x_i r_j x_w)$, $(x_i, \rho i_w)$, $(r_j, \rho i_w)$, где $x_i \in X$; $i_w \in I$; ρ, $r_j \in R$. Для образования более сложных выражений допускается введение операций конъюнкции и дизъюнкции между элементарными выражениями, а также операция отрицания, которая может быть использована как для элементов из R, так и для элементарных выражений.

Пример 2.8. Пусть $X=\{x_1, x_2\}$; $I=\{i_1, i_2, i_3\}$; $R=\{\rho, r_1, r_2\}$. Тогда можно образовать, например, следующие элементарные выражения: $(x_1 r_1 x_2)$, $(x_1 \rho i_3)$, $(r_2 \rho i_1)$ и такие более сложные выражения

$$((((x_1 r_2 x_2)\rho i_1) r_1 (x_2 \rho i_3));$$

$$(x_1 \bar{r}_2 x_2)\,\overline{(x_1 \rho i_3)};$$

$$(x_2 \rho i_1) \vee (x_2 \rho i_3).$$

В первом из них путем суперпозиции произведена операция объединения нескольких элементарных выражений в сложное. Во втором между двумя элементарными выражениями имеется операция конъюнкции *, а кроме того использованы отрицания над отношением r_2 и элементарным выражением $(x_1 \rho i_3)$. В третьем выражении использована операция дизъюнкции.

В отличие от *RX*-кодов язык ситуационного управления позволяет описывать не только постоянные отношения, присущие данному предмету, но и ситуативные отношения между предметами.

Пример 2.9. Рассмотрим следующий текст, описывающий вполне определенную ситуацию: «Иванов сидит в кинотеатре «Восток» в 5-м ряду, занимая место № 2. В этом же ряду на месте № 11 сидит Петров, а сзади, за ним, Сидоров». Введем следующие обозначения: x_1 — человек, x_2 — кинотеатр, x_3 — местоположение, i_1 — Иванов, i_2 — Петров, i_3 — Сидоров, i_4 — «Восток», i_5 — 5-й ряд, место № 2, i_6 — 5-й ряд, место № 11, i_7 — 6-й ряд, место № 11, ρ — иметь имя, r_1 — одновременно, r_2 — находиться в, r_3 — занимать, r_4 — быть непосредственно сзади. Тогда ситуация, описанная текстом, приведенным выше, следующим образом запишется на языке ситуационного управления:

$$(((((x_1 \rho i_1) r_2 (x_2 \rho i_4)) r_1 ((x_1 \rho i_2) r_2$$

$$(x_2 \rho i_4))) r_1 ((x_1 \rho i_3) r_2 (x_2 \rho i_4))) \&$$

$$((x_1 \rho i_1) r_3 (x_2 \rho i_5)) \& ((x_1 \rho i_2) r_3$$

$$(x_3 \rho i_6)) \& ((x_1 \rho i_3) r_4 (x_1 \rho i_2)).$$

В этой записи сознательно одновременно использованы и отношение r_1 и операция конъюнкции. Эти два средства выражения в языке ситуа-

* Знак конъюнкции опускается там, где это не приводит к неоднозначности понимания.

ционного управления практически эквивалентны. Вместо приведенной в этой записи части, относящейся к фиксации местоположения Сидорова, как сидящего непосредственно за Петровым, можно в явной форме указать его местоположение, используя имя i_7.

Особенностью языков синтагматических цепей является их способность вводить новые понятия за счет выделения типовой структуры отношений.

Пример 2.10. Рассмотрим следующую ситуацию. По улице движутся одна за другой пять однотипных грузовых автомашин. Пусть их номера есть i_1—i_5. Введем следующие обозначения: x_1 — автомашина, x_2 — тип автомашины, x_3 — улица, x_4 — направление, i_6 — грузовая, ρ — иметь имя, r_1 — обладать, r_2 — двигаться по, r_3 — находиться, r_4 — быть сзади, r_5 — быть в ε-окрестности. Рассмотрим теперь следующие записи: $(x_1\rho i_1)$, $(x_1\rho i_2)$, $(x_1\rho i_3)$, $(x_1\rho i_4)$, $(x_1\rho i_5)$. Они просто фиксируют существование автомашин с данными номерами. Для простоты дальнейшей записи будем обозначать эти элементарные синтагматические цепи как α_1, α_2, α_3, α_4, α_5. Теперь рассмотрим записи, которые обозначим как β_j: $(\alpha_j r_1(x_2\rho i_6))$, где $j=1, 2, 3, 4, 5$. Их смысл состоит в том, что автомашина с номером i_j является грузовой. Далее введем записи $(\beta_j r_3(x_3\rho\xi))$, смысл которых состоит в утверждении, что грузовая машина с номером i_j находится на улице с именем ξ. Имя ξ играет специальную роль. В исходном тексте имени улицы нет, но из текста ясно, что все пять машин движутся по одной и той же улице. Для записи этого факта и вводится нефиксированное имя улицы ξ. Повторение ξ во всех записях для β_j означает их движение по одной и той же улице. Обозначим эти последние записи как γ_j. Теперь рассмотрим записи

$$(\gamma_j r_2(x_2\rho\eta)).$$

Имя η для направления играет такую же роль, что и имя ξ для улицы. Повторение его в записях указывает на то, что все пять автомашин движутся по улице в одном и том же направлении. Обозначим полученные записи как δ_j. Рассмотрим, наконец, записи

$$(\delta_1 r_4\delta_2)\,(\delta_2 r_4\delta_3)\,(\delta_3 r_4\delta_4)\,(\delta_4 r_4\delta_5)$$

и

$$(\delta_1 r_5\delta_2)\,(\delta_2 r_5\delta_3)\,(\delta_3 r_5\delta_4)\,(\delta_4 r_5\delta_5).$$

Обозначим первую из этих записей ν, а вторую π. Тогда запись $\nu\&\pi$ есть полное описание ситуации, заданной текстом, приведенным в начале примера.

Пусть теперь значение j не фиксировано равным пяти, а задано в виде произвольного параметра q, и при этом $q\geqslant 3$. Пусть также номера автомашин i_j не фиксированы, а являются свободными переменными именами ω_j. Тогда запись $\nu\&\pi$ означает не какую-то фиксированную ситуацию, а более общее понятие, которое можно назвать «колонна грузовых автомашин».

При отображении текстов естественного языка множеств X и R явно не хватает. В естественном языке имеются еще такие средства, как *квантификаторы* (например, сильно, редко, иногда, далеко), *модальности* (желательно, необходимо и т. п.), *императивы* (стой, сделай и т. д.), *модификаторы, оценки* и многое другое. Расширение реляционных языков за счет введения дополнительных выразительных средств приближает их возможности к языкам смыслов.

Пример 2.11. Пусть имеется текст: «Иногда при повышении давления в первичном реакторе необходимо понизить в нем температуру». Введем следующую запись:

$$((m_1x_1)\,r_1(m_2x_2)) \Rightarrow \forall(x_3p_1(m_1x_1)).$$

В этой записи использованы следующие обозначения: x_1 — реактор, x_2 — давление, x_3 — температура, r_1 — обладать, m_1 — первичный, m_2 — повышенное, p_1 — понизить, $\forall$ — иногда. При этом m_1 и m_2 являются модификаторами, приписываемыми понятиям. Они задают некоторые признаковые характеристики для понятий. В ряде случаев их можно рассматривать как имена, но это приведет к усложнению записи. Императив p_1 и квантификатор $\forall$ также являются новыми элементами в описании. Модальность необходимости понижения температуры задана специальным знаком $\Rightarrow$.

Сравнивая между собой языки предикатного и реляционного типов, необходимо отметить следующее. В гл. 1 говорилось о том, что язык представления знаний $Я_c$ требует наличия трех эффективных процедур: процедуры перевода с естественного языка на $Я_c$, процедуры обратного перевода и процедуры эквивалентных преобразований внутри $Я_c$. Для языков предикатного типа процедуры последнего типа хорошо известны. Они сводятся к построению логического вывода в рамках исчисления предикатов первого порядка. Существуют довольно мощные процедуры такого типа, самой известной из которых является *метод резолюций*. Для реляционных языков таких процедур на сегодня нет. Однако реляционные языки, как правило, обладают одним качеством, не присущим предикатным языкам. Они допускают эффективные процедуры обобщения описаний и позволяют создавать иерархически организованные системы знаний. Одно из таких обобщений дано в примере 2.10. Заметим, что полученное в нем обобщенное описание для понятия «колонна грузовых автомашин» может быть обобщено и далее.

Пример 2.12. Заменим в записи ситуации из примера 2.10 переменную x_2 свободной переменной ϕ, принимающей значения из некоторого множества Ψ, элементами которого являются всевозможные

типы автомашин — легковая, грузовая, автокран и т. п. Тогда описание v&л будет соответствовать понятию «колонна автомашин». Обобщение можно продолжить. Вместо x_1 введем свободную переменную θ, принимающую значения из некоторого множества Θ, элементами которого являются автомашины, люди, лошади, муравьи и т. п. Тогда запись v&л может трактоваться как «колонна движущихся однородных объектов» или просто «колонна», если не бояться существующей в русском языке омонимии этого слова.

Такое постепенное обобщение описаний в реляционных языках приводит в конце концов к построению минимального описания с максимальной свободой в заполнении переменных конкретными понятиями и со свободными параметрами типа параметра q в примере 2.10. Это минимальное описание, сохраняющее смысл понятия (например, понятия «колонна движущихся однородных объектов»), называется *структурным фреймом* описания. В следующем параграфе познакомимся с фреймами другого типа.

2.4. Языки ролевого типа

Вслед за логическими и реляционными языками появились языки нового типа. Основной причиной их появления послужило то, что многие достаточно сложные понятия (например, «движение», «командировка», «неудача» и многие другие) не поддаются явному описанию в виде предикатных формул или совокупности отношений. Эту группу языков можно назвать ролевыми.

В таких языках понятие сложного типа определяется через совокупность *обязательных ролей*, заполнение которых необходимо для выражения сущности данного понятия. Понятие определяется строкой, называемой *ролевым фреймом*,

$$[i;\ \rho_1,\ \rho_2, \ldots, \rho_m,\ \rho_{m+1}, \ldots, \rho_n],$$

где i — имя понятия; $\rho_1, \ldots, \rho_m$ — обязательные роли, а $\rho_{m+1}, \ldots, \rho_n$ — необязательные роли.

Пример 2.13. Понятие «самодвижение» можно задать с помощью следующего ролевого фрейма:

[САМОДВИЖЕНИЕ; ОБЪЕКТ, КАК, ГДЕ, КОГДА, КУДА, ОТКУДА, ЗАЧЕМ].

Конкретной реализацией этого фрейма может служить, например, ролевый фрейм

[САМОДВИЖЕНИЕ; камень, катится, λ, λ, λ, с горы, λ].

Символ λ использован для указания тех ролей, которые остались незаполненными при переходе от фразы «Камень катится с горы» к ролевому фрейму. Отметим, что для правильного отображения информации, содержащейся в фразе, в ролевом фрейме необходимо заполнить все обязательные роли в этом фрейме.

Для понятия «движение» ролевой фрейм имеет вид

[ДВИЖЕНИЕ; ОБЪЕКТ, С ПОМОЩЬЮ ЧЕГО, КАК, ГДЕ, КОГДА, КУДА, ОТКУДА, ЗАЧЕМ].

Конкретной реализацией этого фрейма для фразы: «В восемь часов утра мальчик на велосипеде едет в молочный магазин за творогом» будет служить ролевой фрейм

[ДВИЖЕНИЕ; мальчик, на велосипеде, едет, λ, λ, в восемь часов утра, в молочный магазин, λ, за творогом].

В качестве обязательных и необязательных ролей могут выступать имена других ролевых фреймов, что позволяет устанавливать связи между отдельными фреймами и организовывать иерархические структуры из них.

По существу, структура ролевого фрейма совпадает со структурой спецификационного списка, рассмотренного нами в начале главы. Для полного перехода от структуры фрейма к спецификационному списку нужно еще кроме указания имени ролей оставить специальные места для имен конкретных объектов, замещающих эти роли. Другими словами, задать структуру ролевого фрейма в виде

$$[i;\ \rho_1\langle\omega_1\rangle;\ \rho_2\langle\omega_2\rangle;\ \ldots;\ \rho_m\langle\omega_m\rangle;\ \rho_{m+1}\langle\omega_{m+1}\rangle;\ \ldots;\ \rho_n\langle\omega_n\rangle].$$

Будем различать *символические фреймы* и *конкретные фреймы*. В символических фреймах позиции, соответствующие ω_i, остаются незаполненными. В конкретных фреймах все обязательные роли принимают некоторые конкретные значения, т. е. обязательно заполняются позиции ω_1, ω_2, ..., ω_m. Такие конкретные фреймы часто называют *экземплярами фреймов* или *примерами фреймов*. Пары ρ_i, ω_i, представляющие семантически определенную часть ролевого фрейма или, что то же, спецификационного списка, часто называют *слотами*.

Пример 2.14. Пусть мы хотим построить специализированную диалоговую систему, отвечающую на вопросы, связанные с персонализацией лиц, работающих в СССР в области искусственного интеллекта. Единицей хранения информации в системе будет символический фрейм

[ФИО; **МЕСТО РАБОТЫ** <ω_1>; **АДРЕС РАБОТЫ** <ω_2>;
ПУБЛИКАЦИИ <ω_3>; **ОБЛАСТЬ ИНТЕРЕСОВ** <ω_4>;
ВОЗРАСТ <ω_5>; ДОМАШНИЙ АДРЕС <ω_6>;
УЧАСТИЕ В ОРГАНИЗАЦИЯХ ПО ИИ <ω_7>]

Таким образом, наши фреймы будут содержать по семь слотов, из которых четыре будут обязательными. Отметим, что в позициях ω_i могут стоять не только конкретные значения, но и списки конкретных значений. Например, в позиции ω_3 в системе может храниться список публикаций какого-либо конкретного специалиста в области искусственного интеллекта.

Примером фрейма-экземпляра может служить запись

[Нариньяни А. С.; **МЕСТО РАБОТЫ** <Вычислительный центр СО АН СССР>;
АДРЕС РАБОТЫ <Новосибирск 90, Академгородок>;
ПУБЛИКАЦИИ <(большой список публикаций по ИИ)>;
ОБЛАСТЬ ИНТЕРЕСОВ <диалоговые системы, представление знаний, параллельные процессы, робототехника, кибернетика>;
ВОЗРАСТ < >;
ДОМАШНИЙ АДРЕС < >;
УЧАСТИЕ В ОРГАНИЗАЦИЯХ ПО ИИ <Научный совет по проблеме ИИ КСА при Президиуме АН СССР, секция «Искусственный интеллект» Научного совета по комплексной проблеме «Кибернетика», при Президиуме АН СССР>].

Список публикаций А. С. Нариньяни в области искусственного интеллекта мы для экономии места не поместили. Слоты ВОЗРАСТ и ДОМАШНИЙ АДРЕС в данном экземпляре фрейма остались незаполненными.

При хранении информации в памяти ЭВМ в виде ролевых фреймов часто возникает проблема перерасходования памяти из-за дублирования однотипной информации. Если, например, в системе, рассмотренной в примере 2.14, хранятся данные и о других специалистах, работающих в Вычислительном центре СО АН СССР, то для всех их слоты МЕСТО РАБОТЫ и АДРЕС РАБОТЫ будут заполнены одинаково. И чем больше в системе будет храниться информации о сотрудниках ВЦ СО АН СССР, занимающихся проблемами искусственного интеллекта, тем больше дублирующей информации будет хранить система. Чтобы избежать этой неприятности в системах, использующих фреймовое представление, часто применяют прием, получивший название *принципа наследования свойств* (ПНС). Суть его состоит в иерархическом упорядочивании фреймов с помощью специального дерева зависимостей. При этом узлам дерева соответствуют фреймы, хранящиеся в системе. Предполагается, что вся информация,

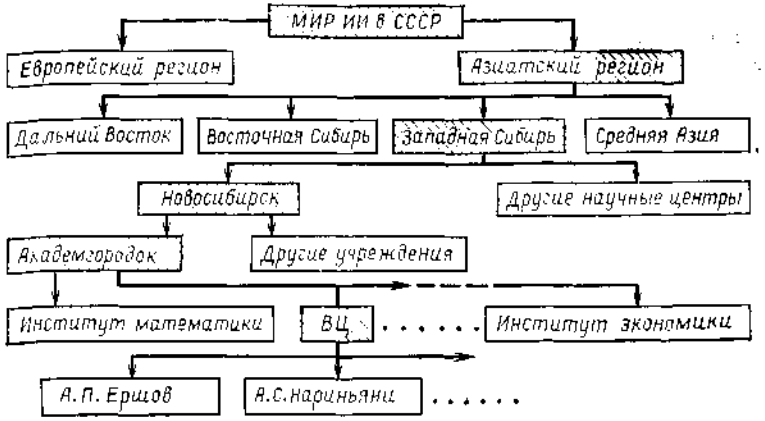


Рис. 2.6

записанная в фреймах, лежащих на пути в дереве от корневой вершины до данной, автоматически переносится и в данную вершину.

Пример 2.15. Вернемся к фреймам из примера 2.14. Введем специальное дерево, фрагмент которого показан на рис. 2.6. С помощью этого дерева все специалисты СССР, работающие в области искусственного интеллекта, упорядочиваются в соответствии со своими местами работы по географическому принципу. При этом фрейм с именем А. С. Нариньяни наследует всю информацию, содержащуюся во фреймах, соответствующих узлам дерева, заштрихованным на рис. 2.6. Это позволяет записать теперь экземпляр фрейма с именем А. С. Нариньяни в следующем виде:

[Нариньяни А. С.; **МЕСТО РАБОТЫ** <ПНС>;
АДРЕС РАБОТЫ <ПНС>;
ПУБЛИКАЦИИ <(большой список публикаций по ИИ)>;
ОБЛАСТЬ ИНТЕРЕСОВ <диалоговые системы, представление знаний, параллельные процессы, робототехника, ПНС>;
ВОЗРАСТ < >;
ДОМАШНИЙ АДРЕС < >;
УЧАСТИЕ В ОРГАНИЗАЦИЯХ ПО ИИ <Научный совет по проблеме ИИ КСА при Президиуме АН СССР, секция «Искусственный интеллект» Научного совета по комплексной проблеме «Кибернетика» при Президиуме АН СССР>].

Появление символа ПНС в слотах МЕСТО РАБОТЫ и АДРЕС РАБОТЫ означает, что необходимая информация для этих слотов содержится во фреймах более высокого уровня, в частности во фрейме с именем ВЦ. Этот фрейм имеет вид

[ВЦ; **МЕСТОПОЛОЖЕНИЕ** <ПНС>;
ПОЛНОЕ НАЗВАНИЕ <Вычислительный центр Сибирского отделения АН СССР>;
АДРЕС <ПНС>;
ЧИСЛО НАУЧНЫХ СОТРУДНИКОВ < >;
ОБЩЕЕ ЧИСЛО СОТРУДНИКОВ < >;
СТРУКТУРА <(полное описание структуры учреждения)>;
ИССЛЕДОВАНИЯ ПО ИИ <ПСС>].

Для слотов с именами МЕСТОПОЛОЖЕНИЕ и АДРЕС необходимая информация может быть получена из вышележащих фреймов (в частности, из фрейма с именем Академгородок). В последнем слоте использован *принцип сборки сведений* (ПСС). Этот принцип позволяет получать необходимую информацию из нижележащих уровней дерева и объединять ее в единое множество.

Приведенный пример достаточно полно характеризует образование иерархических структур из ролевых фреймов. Отметим только, что на самом деле деревьев, подобных показанному на рис. 2.6, может быть несколько. Можно, например, предположить, что кроме дерева, упорядочивающего фреймы по географическому признаку, имеется дерево, упорядочивающее всех специалистов, работающих в СССР в области искусственного интеллекта, по тем или иным проблемам, характерным для этой области. В этом случае указания типа ПНС или ПСС должны иметь индексацию, указывающую на то дерево, которое имеется в данном случае в виду.

Описанные нами принципы языка ролевых фреймов нашли свое первое воплощение в специальных языках для представления знаний, разработанных в последние годы в США. Это языки FRL (Frame Representation Language) и KRL (Knowledge Representation Language).

Первый опыт работы с подобными языками показывает, что основной трудностью для них является построение эффективных процедур поиска и записи информации в памяти ЭВМ, что связано со структурой ролевых фреймов.

Несколько иным вариантом ролевого языка являются системы, использующие в явной или неявной форме ролевые универсалии, отраженные в естественном языке. Рассмотрим в связи с этим язык, получивший название *универсального семантического кода (УСК)*. Остановимся

дишь на некоторых фундаментальных свойствах УСК и подобных ему языков, интересных с точки зрения использования в диалоговых системах, базирующихся на естественном языке.

Элементарной единицей в языке УСК является тройка вида (SAO), где S — *позиция субъекта*, O — *позиция объекта*, а A — *позиция акции* (действия). Такой тройкой описывается, например, предложение: «Канавокопатель роет траншею». В этой фразе слово «канавокопатель» занимает позицию S, «роет» — позицию A, а «траншею» — позицию O. Введем два способа расширения элементарной цепочки (SAO): *доминацию* и *композицию*. При доминации вместо S или (и) O подставляется последовательность субъектов или объектов.

Пример 2.16. Рассмотрим последовательность фраз: «Рабочий завинчивает гайку», «Рабочий гаечным ключом завинчивает гайку», «Склад обеспечивает рабочего деталями». Этим фразам соответствуют следующие записи в УСК: (SAO), (S_1S_2AO), (SAO_1O_2).

Можно показать, что, по крайней мере для русского языка, операция доминации имеет смысл только одношагового расширения цепочки. Другими словами, цепочкам типа $(S_1S_2S_3AO)$ или $(SAO_1O_2O_3)$ нет соответствия во фразах русского языка. При операции композиции вместо S, O или A подставляется множество равноправных элементов такого типа.

Пример 2.17. Рассмотрим последовательность фраз: «Иванов и Петров пилят дрова», «Бригады Петрова, Сидорова и Шустер собирают агрегат», «Автомобиль объезжает ямы и лужи», «Старик пилит, рубит и складывает поленья». Эти фразы имеют следующую запись в УСК: $([S_1, S_2]AO)$, $([S_1, S_2, S_3]AO)$, $(SA[O_1, O_2])$, $(S[A_1, A_2, A_3]O)$.

В отличие от доминации операция композиции может включать в себя теоретически любое число элементов в множествах, заменяющих позиции в элементарной цепочке.

Кроме доминации и композиции используется еще операция *усечения*. Смысл ее состоит в том, что в цепочке (SAO) остается незаполненной либо позиция O, либо позиции S и O одновременно.

Пример 2.18. Рассмотрим следующие фразы: «Автомобиль возвращается», «Диспетчер рукой нажимает кнопку», «Человек пальцем ударяет себя», «Ключом открывают дверь», «Смеркается». Этим фразам могут быть поставлены в соответствие следующие цепочки УСК: $(SA\bar{O})$, (S_1S_2AO), $(S_1S_2A\bar{O})$, $(\bar{S}_1S_2AO)$, $(\bar{S}A\bar{O})$.

Таким образом, в цепочках более сложных, чем исходная, вариантов использования операции усечения гораздо больше, чем в исходной. Заметим также, что случай цепочки $(SA\bar{O})$ можно при необходимости преобразовать так, чтобы позиция субъекта не отрицалась. Появление усечения во второй цепочке связано с тем, что рука есть лишь часть диспетчера и нажимает кнопку, конечно, не рука, а диспетчер.

Последней операцией в УСК является подстановка цепочек (SAO) в позиции S_2 и O_2. (Можно показать, что подстановка целой цепочки в позиции S и O элементарной цепочки (SAO) невозможна.)

Пример 2.19. Рассмотрим следующие фразы: «Вахтенный сообщает капитану о том, что лодка приближается к судну» и «Отец сообщением о том, что собака пропала, расстроил ребенка». Эти фразы имеют в УСК следующее структурное представление: $(S_1A_1O_1(S_2A_2O_2))$ и $(S_1(S_2A_2O_2)A_1O_1)$.

Очевидно, что при тех ограничениях на структуру цепочек в УСК, которые сформулированы, разнообразие допустимых цепочек становится весьма небольшим. Это показывает, что введенные средства еще очень бедны с точки зрения представления смыслов естественных языков. Поэтому в УСК вводятся дополнительные средства для передачи этих возможностей естественных языков. Примерами подобных средств служат операторы возможности и необходимости, вводимые в позиции агенса, аналогичные тем, которые вводятся в модальной логике. Они порождают цепочки $(SMAO)$ и $(SNAO)$, интерпретируемые как «Субъект может воздействовать на объект» и «Субъект должен воздействовать на объект». Кроме того, вводятся специальные средства квантификации, позволяющие формировать цепочки, соответствующие таким кванторам, как: все, существуют, некоторые, много и т. п. Наконец, вводятся модификаторы, позволяющие сужать квантифицированные множества, например, за счет указания на признаки элементов, что позволяет из множества автомобилей вычленять грузовые автомобили или из множества станков — токарные станки, а из последнего множества — токарные станки для обработки дерева. Для отображения в УСК отрицательных конструкций вводится операция отрицания на всех трех позициях элементарной цепочки (так, что $(\neg X \neg A \neg Y)$ интерпретируется как «не X не влияет на не Y»).

Пример 2.20. Для иллюстрации возможностей УСК приведем несколько примеров цепочек этого языка вместе с их интерпретацией: $(\{S\}AO)$ — «рабочие работают». В этой цепочке $\{S\}$ означает некоторое множество индивидов, называемых рабочими, которые вступают между

собой в операцию композиции. Эта фраза, вообще говоря, не слишком понятна. С одной стороны, она описывает некий вневременной и внепространственный процесс, касающийся некоторого абстрактного множества людей, называемых рабочими. С другой стороны, в рамках некоторой конкретной ситуации речь может идти о конкретной группе рабочих, которые в данном месте и в данное время выполняют некоторую определенную работу. Для различения этих возможностей в УСК вводятся специальные средства. Они реализуются в виде записей: $(\{S\}DA\bar{O})$ — «(Эти) рабочие работают (сейчас и здесь)», т. е. речь идет о конкретных рабочих, к которым относится указание «эти»; $(\{S\}KA\bar{O})$ — «Эти рабочие работают (всегда и везде вместе)», т. е. речь идет об определенной группе рабочих, например о бригаде с постоянными членами ее; $(\{\{S\}\}KA\bar{O})$ — «Все рабочие работают», т. е. некоторое утверждение вневременное и внеситуативное.

Приведем еще несколько примеров записей на УСК. $(\{X\}\ \{A\}\ \{Y\})$ — «Много X долго влияет на много Y; $(/X/\ /A/\ /Y/)$ — «Активный X активно воздействует на активного Y»; $(X_1\bar{X}_2 \neg KA\bar{O})$ — «X_1 не всегда сильный»; $(X_1\bar{X}_2KNA\bar{O})$ — «X всегда должен быть красивым».

Для ролевых языков (и в частности для УСК) основной проблемой является автоматизация перевода текстов на естественном языке в представления в виде ролевых фреймов. Эта проблема пока далека от положительного решения. Объясняется это тем, что ролевые языки значительно ближе к языку смыслов $Я_с$, чем предикатные и реляционные языки. А проблема автоматизации перевода во многом еще не решена и для этих языков.

ГЛАВА ТРЕТЬЯ

ДИАЛОГОВАЯ ВОПРОСНО-ОТВЕТНАЯ СИСТЕМА ДЛЯ ЭНЕРГЕТИКИ (ДВОЭ)

3.1. Общая характеристика системы

Система ДВОЭ предназначена для работы в автоматизированных системах диспетчерского управления энергосистемами и энергообъединениями. Ее прикладной модуль, связанный с обработкой большого количества информации, поступающей на вход системы, решением задач слежения, корректировки, оптимизации и ряда других, играет по отношению к языковому модулю и модели представления знаний определяющую роль. Система ДВОЭ является профессионально ориентированной системой. Но ряд принципов, реализованных в ее языковом модуле и особенно в модели представления знаний, являются интерес-

ными и при построении других диалоговых языковых систем, ориентированных на решение конкретных прикладных задач.

В настоящее время ДВОЭ реализована в виде комплекса взаимосвязанных программ на ЭВМ ЕС-1010. В качестве языка программирования для системы использован один из языков типа ассемблер.

Системы информационного типа, использовавшиеся в последнее время в энергообъединениях и энергосистемах, обладают жестким, неадаптивным поведением. Это определяется следующими причинами:

1) язык диалога является в сущности языком функций информационной системы, а не естественным профессиональным языком диспетчерского управления в данной отрасли;

2) форма ответа не зависит от ситуации и однозначно определяется формой запроса;

3) различные программы, входящие в комплекс математического обеспечения, выполняются или с заранее заданными периодами (обработка, обновление оперативной информации на экранах дисплеев и другие функции), или при выполнении заранее заданных условий (инициирование редактирующих программ диалоговой системы); в этих системах не принимается в зависимости от ситуации в объекте управления или в самой системе решение о необходимости инициирования тех или иных программ;

4) не производится автоматический или по крайней мере автоматизированный анализ ситуации в объекте управления с выдачей диспетчерскому персоналу совета о рекомендуемых в данной ситуации действиях.

Эти недостатки диалоговых информационных систем (ДИС) с жестким режимом функционирования можно преодолеть лишь на пути создания адаптируемых систем, необходимой принадлежностью которых является модель знаний о том мире, в котором функционирует ДИС. Наличие же такой модели и развитой системы представления знаний дает возможность существенно обогатить диалог системы с пользователем. При этом целесообразно не заменять старые ДИС с жестким поведение новыми адаптивными системами, а возводить новые части ДИС, опираясь на уже имеющиеся блоки жесткой обработки информации. Таким образом, новая ДИС представляется двухуровневой системой: уровень жесткого поведения управляет простейшими функциями, а более сложные функции находятся под управлением системы адаптивного поведения. При таком взаимодействии жесткого и адаптивного

поведения ДИС объектом внедрения для «интеллектуальных» информационных систем являются уже существующие ДИС.

С системотехнической точки зрения введение в состав ДИС подсистемы адаптивного поведения означает включение в комплекс программного обеспечения ДИС дополнительного комплекса программ в соответствии с общими системными требованиями. Функционирование программ подсистемы жесткого поведения должно осуществляться в соответствии с условиями, действовавшими до включения программ адаптивной подсистемы.

Дополнительные программы, подключаемые при этом к комплексу ДИС, можно разделить на две группы: программы собственно адаптивного поведения и программы связи подсистем жесткого и адаптивного поведения, причем программы связи по способу инициирования относятся к подсистеме жесткого поведения.

На рис. 3.1 приведена общая блочная структура ДИС, в которой сочетаются подсистемы жесткого и адаптивно-

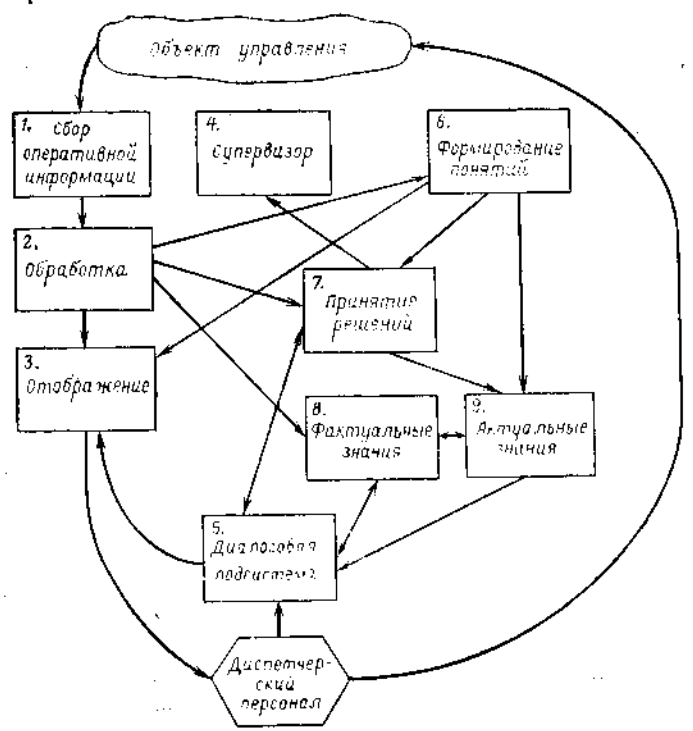


Рис. 3.1

го поведения (в дальнейшем будем сокращенно именовать эти подсистемы соответственно ПсЖ и ПсА).

К ПсЖ относятся блоки *1, 2, 3*, осуществляющие сбор оперативной информации, поступающей от объекта управления, обработку этой информации и отображение в однозначном соответствии запросам диалоговой системы.

Блоки *4* (супервизор) и *5* (диалоговая система) состоят как из частей, принадлежащих ПсЖ, так и из частей, относящихся к ПсА. При этом относящиеся к ПсЖ части этих блоков обеспечивают нижние уровни их функционирования, а верхние уровни находятся в ПсА. В частности, к жестким супервизорным функциям относится запуск отдельных программ с априорно заданными циклами и инициирование запуска программ в соответствии с априорно заданными условиями, а адаптивное распределение ресурса системы относится к ПсА. В общем случае блок *1* осуществляет сбор не только внешней информации от объекта управления, но и внутренней информации о функционировании системы технических и программных средств, на базе которой реализована ДИС. В дальнейшем для краткости под понятием «информация от объекта управления» будем понимать и внешнюю и внутреннюю информацию.

К ПсА относятся полностью блоки *6—9*. Основой ПсА является система знаний — модель мира информационной системы, состоящей из блоков *8, 9*. Блок *8* представляет собой систему знаний о фактах внешнего и внутреннего мира ДИС, а блок *9* — систему знаний о действиях, которые целесообразно выполнять в различных ситуациях, возникающих во внешнем и внутреннем мирах ДИС. Назовем блоки *8* и *9* соответственно подсистемами фактуальных и актуальных знаний.

Вопросно-ответной системой в узком смысле является подсистема фактуальных знаний — блок *8* (модель информации о мире), связанная с диалоговой подсистемой (блок *5*).

Блок *8* имеет три входа. Вход от блока *2* (блок обработки в ПсЖ) обеспечивает ввод в модель новых фактов об объекте управления. Ввод этой информации в подсистему фактуальных знаний обеспечивается специальными программами, связывающими ПсЖ с ПсА (автоматический ввод). Через диалоговую подсистему *5* диспетчерский персонал вводит в вопросно-ответную систему *8* запросы на языке, близком к профессиональному естественному языку диспетчерского управления и через диалоговую подсистему *5* и подсистему отображения *3* получает

ответы на эти запросы. При автоматическом анализе ситуаций в объекте управления ПсА осуществляет внутренний диалог подсистемы актуальных знаний *9* с подсистемой фактуальных знаний *8*. Задавая блоку *8* внутренние вопросы и анализируя полученные ответы, подсистема актуальных знаний конкретизирует содержащуюся в ней обобщенную информацию о релевантных ситуации действиях и вырабатывает совет для диспетчерского персонала (этот совет передается через блоки *5* и *3*). Таким образом, вопросно-ответная система должна отвечать на вопросы, заданные на различных языковых уровнях: внешнем (от диалоговой подсистемы) и внутреннем (от подсистемы актуальных знаний).

Подсистема актуальных знаний (блок *9*) должна содержать набор сценариев для анализа ситуаций в объекте управления, причем каждый сценарий соответствует отдельной типичной ситуации.

Блок *7* служит для принятия решения о том, какой из сценариев блока *9* целесообразно инициировать в данной конкретной ситуации. Входную информацию блок принятия решения получает через ПсЖ, от блока обработки *2*, а также от блока *6*. Блок принятия решений может использоваться не только для инициирования сценариев анализа ситуации, но и для адаптивных супервизорных функций: для адаптивного распределения ресурса ЭВМ ДИС в зависимости от текущей ситуации. Эту функцию блока *7* показывает связь, направленная к супервизору *4*.

Эффективность работы блока принятия решения во многом зависит от степени обобщения, классификации входной информации. Такое обобщение и осуществляет блок формирования понятий текущей ситуации *6* на основании информации, полученной от блока *2* ПсЖ.

Результаты обобщения и классификации ситуации в блоке *6* представляют также непосредственный интерес для диспетчерского персонала, так как сжатое описание ситуации на языке ее наиболее общих признаков помогает локализовать внимание диспетчера. Поэтому выход блока *6* связан со входом блока отображения *3*: сформированное понятие ситуации отображается на экранах дисплеев.

Решения, которые принимаются в блоке *7*, критерии, актуальные в данной ситуации, «намерения» блока принятия решений также могут быть предметом диалога с диспетчерским персоналом. Для этой цели служит связь между блоками *7* и *5*.

Таким образом, к вопросно-ответной системе в широком смысле следует также отнести блоки актуальных знаний,

принятия решений, формирования понятий. Для ДИС характерна работа в реальном масштабе времени в изменяющемся мире при отсутствии «самостоятельного» воздействия ЭВМ на объект управления: такое воздействие осуществляется человеком-диспетчером. При этом важными требованиями к ДИС являются сервисность общения с ней, скорость реакции на запросы, полнота и достоверность информации, отображаемой для диспетчерского персонала. Высокие требования к надежности функционирования ДИС определяют ее реализацию на базе двух резервирующих друг друга управляющих мини-ЭВМ.

Работа на мини-ЭВМ накладывает существенные ограничения на ресурсы ДИС (оперативную память, вычислительную мощность). Эти общие требования к ДИС определяют и требования к подсистеме адаптивного поведения, в состав которой входит вопросно-ответная система:

1. Необходимость работы в реальном масштабе времени, по отношению как к информации, поступающей от объекта управления, так и к запросам диспетчерского персонала. Процессы, с которыми имеют дело многие ДИС, быстротечны. Так, при управлении энергосистемами период поступления новой информации от объекта управления не превышает 1—3 с.

Поскольку диспетчерский персонал будет обращаться к вопросно-ответной системе в трудных (например, предаварийных) ситуациях, недопустимы большие задержки ответа. Время реакции вопросно-ответной системы не должно превышать нескольких секунд.

2. Ограниченный ресурс мини-ЭВМ по производительности и объему оперативной памяти требует выбора для реализации ПсА наиболее простых и наиболее эффективных методов. Эта ограниченность ресурса, в частности, делает нецелесообразным использование языков высокого уровня для программирования задач ПсА.

3. Подсистема адаптивного поведения входит в общий комплекс программ ДИС вместе с программами ПсЖ. При этом программы ПсА подчиняются общим системным условиям. Таким образом, программист, создающий математическое обеспечение для ПсА ДИС, не может рассчитывать на полный ресурс мини-ЭВМ, а должен выполнять системные условия, которые, естественно, ограничивают его ресурсные возможности.

Рассматривая перечисленные выше требования, следует отметить их противоречивость: требование минимального времени реакции и работы в реальном масштабе времени явно противоположно требованию считаться с ограничением ресурсом мини-ЭВМ и соблюдать системные ограничения.

Этими противоречивыми требованиями обусловливаются те ограничения и упрощения, которые введены в вопросно-ответную систему ДВОЭ, в частности ограничения и упрощения во входном языке системы.

3.2. Язык вопросно-ответной системы

Входной язык вопросно-ответной системы должен быть близок к естественному профессиональному языку управления данным производством. Пользование этим языком не должно требовать от диспетчерского персонала знания специальных формальных правил или сколько-нибудь сложного предварительного обучения.

Средства общения должны обеспечивать удобное пользование языком и защиту от неправильных употреблений. Вопросно-ответная система должна быть инвариантна к лексике входного языка, легко перестраиваться на работу с языками диспетчерского управления другими производственными процессами (при сохранении определенных ограничений, например, на объем словарного состава языка). В дальнейшем будем сокращенно именовать входной язык языком ЯДРО (Язык Диспетчерских Решений Оперативный).

Упрощения, вводимые в ЯДРО, по отношению к естественному профессиональному языку должны быть легко понятны носителям естественного языка. Упрощения не должны превращать естественный язык в формальный, в языке ЯДРО целесообразно сохранить такие важные свойства естественного языка, как неоднозначность (полисемию), синонимию, богатые выразительные возможности (в частности, метонимию), возможности умолчаний, эллипсисов, возможность использования качественных характеристик (определительных наречий).

Ниже подробнее рассматриваются основные характеристики языка ЯДРО.

1. *Лексика и морфология.* Сложный морфологический анализ требует больших затрат ресурса ЭВМ и ввиду этого нежелателен при реализации вопросно-ответной системы на мини-ЭВМ. С другой стороны, требование печати слов входного языка на универсальной алфавитной клавиатуре (а только при этом обеспечивалась бы возможность ввода во входной язык морфологически изменяемых словоформ) противоречит необходимой оперативности общения диспетчера с ДВОЭ. Диспетчер не должен

пользоваться клавиатурой, как машинистка, так как это отвлекало бы его внимание от выполнения профессиональных обязанностей. Выходом из этого положения может быть использование функциональной клавиатуры, на клавиши которой нанесены слова входного языка (на каждой клавише по слову). При этом устраняется также необходимость в морфологическом анализе, так как слово, написанное на клавише, превращается в «иероглиф». Однако такое использование функциональной клавиатуры эффективно только при небольшом словарном составе языка. Известно, что словарь профессиональных терминов диспетчера энергосистемы составляет около 400 слов, из которых наиболее часто употребляются около 200 слов. Поскольку ДИС, построенная на базе мини-ЭВМ, является главным образом фактографической системой, можно исключить ряд слов, необходимых только для поиска документов (справок и т. п.). Практически словарь языка оперативного диспетчерского управления энергосистемой может быть сокращен до 100—120 слов. Клавиатура с таким количеством клавиш достаточно удобна в пользовании. В число слов, вынесенных на клавиши, входят цифры (0—9), некоторые дополнительные слова (вопросительные слова, например КАКОЙ), знаки препинания (вопросительный знак, запятая).

2. *Синтаксис.* Ограничимся рассмотрением в вопросно-ответной системе только вопросительных предложений языка ЯДРО. Вопросы обеспечивают потребности диспетчера при общении с ДВОЭ, так как информация о фактах вводится в ПсА ДИС автоматически из ПсЖ. Относительно редко возникающая необходимость введения сведений в ЭВМ в диалоговом режиме может быть обеспечена «жесткой» частью диалоговой подсистемы (операции по заполнению выводимых на экраны бланков).

Далее сузим предмет рассмотрения только до вопросительных предложений с вопросительными словами КАКОЙ (КАКИЕ). По-видимому, некоторые другие типы вопросов могут быть сведены путем простых преобразований к вопросам с вопросительными словами КАКОЙ.

Пример 3.1. Например, запросы с вопросительным словом СКОЛЬКО могут быть преобразованы в вопрос со словом КАКОЙ:

1. СКОЛЬКО ПАРАМЕТР АВАРИЙН? (Сколько параметров находятся в аварийном состоянии?)

Этот вопрос преобразуется в вопрос

2. КАКОЙ ПАРАМЕТР АВАРИЙН? (Какие параметры находятся в аварийном состоянии?)

После получения ответа на такой преобразованный вопрос нужно выполнить операцию СЧЕТ для определения общего числа выделенных параметров. Результат счета является ответом на исходный вопрос.

Всякое вопросительное предложение может быть разделено на *рему* (о чем спрашивают) и *тему* (условия вопроса). Вопросительное условие указывает на рему, и мы будем считать, что рема непосредственно следует за вопросительным словом. Например, в вопросе 2 из примера 3.1 рема — это ПАРАМЕТР, а тема — АВАРИЙН.

Если вопросительное слово отсутствует, то примем, что рема расположена в начале предложения. Изменение порядка слов в вопросительном предложении может менять его смысл.

Пример 3.2. 3. ПАРАМЕТР АВАРИЙН? В этом вопросе роль ремы играет ПАРАМЕТР, так как вопросительное слово отсутствует.

4. КАКОЙ ОБЪЕКТ РЕГИОН СЕВЕР?

5. КАКОЙ РЕГИОН ОБЪЕКТ 2?

В вопросе 4 ремой является ОБЪЕКТ, а РЕГИОН СЕВЕР является темой. В вопросе 5 РЕГИОН есть рема, а ОБЪЕКТ 2 — тема. Однако, перефразировав вопрос 5 следующим образом: 5а. ОБЪЕКТ 2 КАКОЙ РЕГИОН? мы не изменяем его смысл, а следовательно, тему и рему.

Тема вопроса может содержать не одно условие, как в вопросах, приведенных в двух предшествующих примерах, а несколько.

Пример 3.3. Имеем вопрос:

6. КАКОЙ ПАРАМЕТР РЕГИОН 2 АВАРИЙН?

Считаем, что все условия, входящие в тему, конъюнктивны, т. е. при разделении исходного вопроса на несколько подвопросов с одним условием из темы исходного вопроса в каждом подвопросе ответ на исходный вопрос может быть найден как пересечение множеств ответов подвопросов. Вопрос 6 разделяется на подвопросы:

6а. КАКОЙ ПАРАМЕТР РЕГИОН 2?

6б. КАКОЙ ПАРАМЕТР АВАРИЙН?

Ответ получается как пересечения множества параметров, входящих в регион 2, и множества параметров, находящихся в аварийном состоянии.

В языке вопросно-ответной информационной системы ДВОЭ допускается и тот случай, когда вопрос содержит придаточную часть, отделяемую от основной части предложения запятой.

Пример 3.4. 7. КАКОЙ ПАРАМЕТР, РЕГИОН АВАРИЙН?

Придаточная часть может быть выделена в подвопрос, причем рема подвопроса входит условием в тему основного вопроса:

7а. КАКОЙ РЕГИОН АВАРИЙН? Ответ на этот подвопрос входит условиями в основной вопрос:

7б. КАКОЙ ПАРАМЕТР РЕГИОН *? Звездочкой помечено множество регионов, являющихся ответом на подвопрос 7а. Таким образом, вопрос с придаточной частью является иерархической конструкцией.

В вопросе могут допускаться анафорические связи.

Пример 3.5. На вход системы могут поступить два вопроса в следующей последовательности:

8. КАКОЙ РЕГИОН ОБЪЕКТ 2?

9. КАКОЙ УПОМ АВАРИЙН?

Система воспринимает смысл этих вопросов, расшифровывая их следующим образом: «Какой из упомянутых объектов находится в аварийном состоянии?»

Пример 3.6. Приведем ряд примеров вопросов на языке ЯДРО, иллюстрирующих выразительные возможности языка:

10. КАКОЙ РЕГИОН АВАРИЙН? — вопрос, содержащий синекдоху как разновидность метонимии: употребление общего понятия вместо частного; без метонимии вопрос 10 нужно было бы сформулировать значительно более громоздко:

10а. КАКОЙ РЕГИОН, ОБЪЕКТ, ПАРАМЕТР АВАРИЙН?

(какие [имеются] регионы, [содержащие] объекты, параметры [которых] [находятся в] аварийном [состоянии]?)

11. КАКОЙ ПЕРЕТОК ...

11а. КАКОЙ МОЩНОСТЬ ЛИНИИ ... эти фрагменты вопросов являются синонимическими конструкциями.

12. КАКОЙ ПАРАМЕТР СИЛЬНО АВАРИЙН? — вопрос с определительным наречием.

В приведенных выше примерах вопросов опущены имена отношений, т. е. эти предложения являются эллиптическими (опущенные слова в расшифровках вопросов заключены в квадратные скобки).

Ответами на вопросы, задаваемые на языке ЯДРО, должны быть перечисления названий объектов, параметров или характеристик объекта управления. Для ответа на некоторые вопросы целесообразно предъявить информацию в виде таблицы или схемы с нанесенными на нее текущими параметрами объекта. Например, на вопрос: 13. КАКОЙ СХЕМА ПОДСТАНЦИЯ РЕГИОН СЕВЕР? — какие схемы подстанций региона Север? следует вывести на экран дисплея схемы подстанций, релевантные вопросу, в ответ на вопрос 6а нужно перечислить названия параметров, принадлежащих энергообъектам, входящим в регион 2.

Для синтеза таких ответов нет необходимости использовать естественный язык. Для синтеза ответов используется язык технологических наименований, схем, таблиц. При этом указанная информация используется также и в ПсЖ (например, технологические наименования параметров входят в списки нарушений технологических пределов, формируемые в блоке *обработка* ПсЖ). Выходной язык ДВОЭ будем в дальнейшем называть ИМЯ.

3.3. Активная семантическая сеть

Семантические сети являются основной формой представления информации о фактах мира и для накопления опыта системы. В большинстве блоков ПсА используются семантические сети. Семантическая сеть представляет собой граф, вершинам которого соответствуют объекты и понятия мира системы, а дугам — отношения между этими объектами и понятиями. Каждому типу отношений соответствует свой цвет дуги. Между двумя вершинами семантической сети может существовать несколько дуг разного цвета, таким образом, сеть является раскрашенным мультиграфом. При моделировании семантической сети на ЭВМ моделируются вершины и дуги сети. Модель вершины сети будем называть *узлом*, а модель дуги — *связью*. Будут рассматриваться только бинарные отношения.

На рис. 3.2 показан фрагмент семантической сети, характерный для энергосистем. Все дуги в этой сети определяют возбуждающие связи. Кроме того, на рисунке использованы следующие обозначения:

ПК — отношение *принадлежит классу*; ВК — *включает в класс*; ПР — *принадлежит региону*; ВО — *включает объект*; ЛПО — *(линия) принадлежит объекту*; ОСЛ — *объект связан с линией*; ЯТО — *является типом объекта*; ИТО — *имеет тип объекта*; ИО — *измеряется на объекте*; ИП — *имеет параметр*; ИЛ — *измеряется на линии*; НС — *находится в состоянии*; БСП — *быть состоянием параметра*; ИВ — *имеет вид параметра*.

Рассматриваемые семантические сети активные, так как в процессе их функционирования отдельные вершины и дуги будут возбуждаться и тормозиться. В любой момент узлу соответствует уровень *возбуждения* $x(x_{min} \leqslant x \leqslant x_{max})$, а связи — значение *проходимости* $R(R_{min} \leqslant R \leqslant R_{max})$. Семантически уровень возбуждения узла означает степень важности денотата этого узла (объекта или понятия) для достижения поставленной цели (например, для ответа на заданный вопрос). Значение проходимости связи означает силу отношения, соответствующего этой

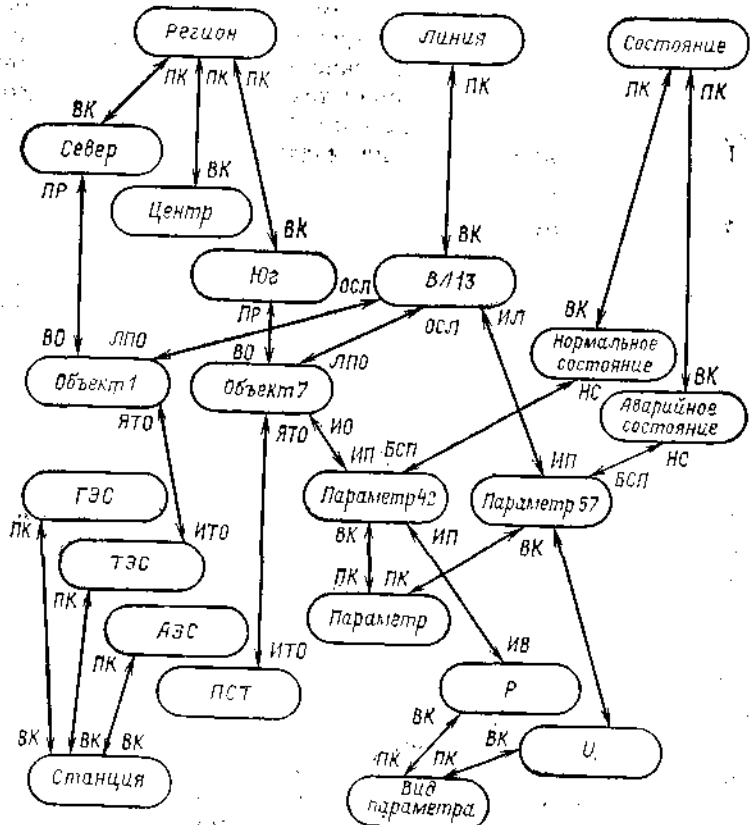


Рис. 3.2

связи. Работа модели активной семантической сети выражается в распространении возбуждений между узлами сети по связям, причем в каждом узле суммируются возбуждения связанных с ним узлов с весовыми коэффициентами, пропорциональными проходимостям связей:

$$x_j = \left[\left(\sum_{\substack{k=1 \\ k \neq j}}^{N} x_k R_{jk} \right) \wedge x_{max} \right] \vee x_{min}, \qquad (3.1)$$

где N — число узлов модели семантической сети; j — номер узла; x_j — уровень возбуждения j-го узла; R_{jk} — проходимость связи между узлами j и k; $\wedge$ — обобщенная конъюнкция (выбор минимума); $\vee$ — обобщенная дизъюнкция (выбор максимума).

Целесообразно принять $x_{min}=0$; $x_{max}=1$; $R_{min}=-1$; $R_{max}=+1$.

При этом отрицательным значениям проходимости соответствуют тормозные связи, а положительным значениям — возбуждающие. Тормозные связи, передавая возбуждение, ослабляют уровень возбуждения узла, в который они входят. Возбуждающие связи, наоборот, усиливают возбуждение этого узла. Важным элементом модели семантической сети является *активатор*. Роль активатора сводится к выявлению наиболее возбужденного узла сети и к дополнительному его возбуждению в соответствии с определенным критерием; при этом ослабляется возбуждение остальных узлов. Последняя функция может выполняться слабыми тормозными связями между узлами сети. Кроме дополнительного возбуждения активатор уменьшает инерционность наиболее возбужденного узла. С учетом работы активатора преобразуется формула (3.1):

$$x_j = \left\{ \left[\left(\sum_{\substack{k=1 \\ k \neq j}}^{N} x_k R_j \right) + \alpha\Delta \right] \wedge x_{max} \right\} \vee x_{min}, \qquad (3.2)$$

где α — функция активатора для узла с номером j;

$$\alpha_j = \left(\bigvee_{k=1}^{N} x_k \right) \left(x_j \geqslant x_k \right); \qquad (3.3)$$

Δ — критерий дополнительного возбуждения выбранного активатором узла; в качестве Δ принимается обычно разность возбуждений двух определенных узлов сети. Таким образом, роль активатора сводится к выделению в семантической сети наиболее важного в данный момент ее функционирования узла и ослаблению влияния на него узлов, которые с точки зрения решаемой в настоящее время задачи можно рассматривать как помехи и шумы. Метод сравнения возбуждения отдельных узлов, реализуемый с помощью соотношения (3.3), не единственно возможный при функционировании активатора. Для ряда задач важно, например, выделять не отдельные узлы семантической сети, а целые участки ее.

3.4. Обучение в семантической сети

Процессы обучения в активной семантической сети осуществляются путем изменения по определенному алгоритму значений проходимости связей между узлами модели. Рассмотрим три различных алгоритма такого измене-

ния связей: а) *Наблюдение*, б) *Рефлекс*, в) *Межуровневая проекция*.

1. Алгоритм *Наблюдение*. Этот алгоритм действует в том случае, когда система получает сообщение $\{i, j, r_\varphi\}$, где i, j — номера (имена) объектов или понятий, φ — тип отношения, а r — его сила.

Сообщение может быть получено автоматически от ПсЖ (от блока *обработка*) или через диалоговую систему. В первом случае может быть задана функция, связывающая проходимость связи, моделирующей отношение φ, с некоторыми количественными характеристиками объектов или понятий, соответствующих узлам i и j модели сети. Например, для сообщения

$\{i$ [*параметр*] φ [*находится в*] j [*состояние* 2]$\}$

сила отношения и, следовательно, проходимость R_{ij} связи может быть определена из формулы

$$R_{ij}=k(y-p_т)/(y_{max}-y_{min}), \qquad (3.4)$$

где y — значение параметра; $p_т$ — значение технологического предела, который нарушен этим параметром (именно это нарушение и определяет нахождение параметра в состоянии № 2 (аварийное состояние)); y_{max}, y_{min} — максимальное и минимальное из возможных значений параметра; k — коэффициент пропорциональности. Вычисления проходимостей связей для сообщений, поступающих автоматически от ПсЖ (эти сообщения для ПсА аналогичны наблюдениям), производятся программами, связывающими ПсЖ с ПсА. Если сообщение поступает через диалоговую систему, то может оказаться удобным задавать силу отношения качественными характеристиками, подобными лингвистическим переменным в теории нечетких множеств: [параметр]$_i$ [сильно]$_r$ [отклоняется от]$_\varphi$ [плановое значение]$_j$. При этом для задания проходимости связи модели сети необходимо иметь функцию соответствия $R=F(\mu)$, где μ — термы лингвистической переменной, например (*слабо*/0,1, *средне*/0,5, *сильно*/0,9).

Алгоритм *Наблюдение* целесообразно применять при непосредственном задании конкретных фактов.

2. Алгоритм *Рефлекс*. При функционировании модели сети возбуждения узлов распространяются по направлению связей. Если возбуждающая связь направлена от узла i к узлу j и задано начальное возбуждение узла i, то процесс возбуждения узла j запаздывает обратно пропорционально значению проходимости этой связи R_{ij}. На

основании этого возможен алгоритм самообучения, при котором проходимость связи на каждом такте модели сети усиливается пропорционально возбуждению узлов, между которыми включена эта связь:

$$R_{ij}^{\tau+1}=R_{ij}^{\tau}+(x_i^{\tau}+x_j^{\tau})\,\Delta, \qquad (3.5)$$

где τ — такт работы модели.

Однако такой алгоритм самообучения не обеспечивает необходимого для работы в реальном времени быстродействия, так как на каждом такте приходится пересчитывать все связи, число которых для практических задач весьма велико.

Для уменьшения времени пересчета проходимостей связей целесообразно предоставить активатору выбор тех связей, проходимость которых должна измениться в данном такте. Будем полагать, что в процессе самообучения должны изменяться проходимости только возбуждающих связей; тормозные связи считаются неизменными. При этом процессы торможения в модели сети обусловливаются отрицательной индукцией от связанных тормозными связями узлов (такие узлы соответствуют антагонистическим понятиям, например «состояние нормальное» и «состояние аварийное»).

Исходя из сказанного выше, по принципу рефлекса усиливаться (т. е. увеличивать проходимость) должна та возбуждающая связь, которая связывает узлы, выбранные активатором в последовательные временные такты. Кроме того, должны ослабляться (уменьшать проходимость) связи, исходящие из узла, выбранного активатором на каждом такте, причем это ослабление осуществляется, если проходимость соответствующей связи ненулевая и возбуждение узла, к которому направлена эта связь, не равно нулю. Процесс ослабления связей аналогичен забыванию и должен сопровождаться процессом самообучения для адаптации к изменяющемуся миру системы.

Таким образом, самообучение по принципу рефлекса может быть описано формулой

$$R_{ij}^{\tau}=[R_{ij}^{\tau}+\alpha_j^{\tau}k_1(x_i^{\tau}-x_j^{\tau})\sigma(R_{ij}^{\tau})\sigma(x_j^{\tau})+ \\ +\alpha_i^{\tau-1}\alpha_j^{\tau}k_2(x_i^{\tau}+x_j^{\tau})]\wedge R_{max}, \qquad (3.6)$$

где $\sigma(y)=[\mathrm{sign}(y)]\vee 0$; k_1 и k_2 — постоянные коэффициенты; $\alpha^{\tau}{}_j$ — функция активатора для j-го узла в такт τ; остальные обозначения те же, что и в предыдущих формулах.

Второе слагаемое описывает процесс забывания, третье слагаемое — процесс самообучения. Очевидно, что при коррекции связей по (3.6) на каждом временно́м такте пересчету подвергаются только связи одного узла сети: одна из связей, входящих в этот узел, усиливается, а все связи, выходящие из узла, ослабляются. Увеличение быстродействия алгоритма по сравнению с быстродействием предшествующего алгоритма пропорционально, таким образом, числу узлов сети.

Ввиду того что самообучение по рассматриваемому алгоритму основано на установлении соответствия между событиями, происходящими последовательно во времени, алгоритм *Рефлекс* целесообразно применять для коррекции связей, соответствующих отношениям каузативного характера.

3. Алгоритм *Межуровневая проекция*. Прежде чем рассматривать этот алгоритм, необходимо ввести понятие уровня семантической сети. Разделим отношения, представленные в сети, на два класса: *парадигматические*, соответствующие обобщению (конкретизации) объектов и понятий мира системы (например, родовидовые отношения), и *синтагматические (ситуативные)*. Соответствующие парадигматическим отношениям связи будем называть π-связями, а связи, соответствующие ситуативным, синтагматическим отношениям, — σ-связями. Для семантической сети, показанной на рис. 3.2, парадигматическими отношениями можно назвать отношения *принадлежность классу* (ПК), *включать в класс* (ВК), остальные отношения — ситуативные. Соответственно связи, помеченные на этом рисунке индексами ПК и ВК, являются π-связями, а остальные связи — σ-связями. Если π-связи изображать вертикальными линиями, а σ-связи — горизонтальными, получим уровни сети, соединенные π-связями, причем σ-связи оказываются внутриуровневыми. Таким образом, уровни отделены друг от друга π-связями. Номер уровня определяется числом π-связей, отделяющих этот уровень от нулевого, причем узлы нулевого уровня соответствуют наиболее конкретным объектам и понятиям мира системы. Таким образом, возрастание номера уровня соответствует увеличению уровня обобщения.

На рис. 3.3 показана двухуровневая семантическая сеть для примера, приведенного на рис. 3.2. При этом σ-связи показаны сплошными линиями (для упрощения не показаны типы σ-связей), а π-связи — пунктирными. На верхнем (первом) уровне отсутствуют σ-связи на рис. 3.3. Между узлами всех уровней, кроме нулевого, может существовать два типа ситуативных связей: связи общности (проходимости их обозначаются $R_{\text{в}}$) и связи существования $R_{\text{э}}$. Связь существования верхнего уровня отражает тот факт, что на нижнем уровне в данный момент существует хотя бы одна связь данного вида. Например, связь $R_{\text{э}}$ на уровне 1 (рис. 3.3) между узлами *параметр* и *линия* означала бы, что на нулевом уровне имеется связь между одним из узлов, соответствующих конкретному параметру, и узлом, соответствующим конкретной линии (на рисунке *параметр 57—ВЛ 13*). Связь типа $R_{\text{в}}$ между узлами *параметр* и *линия* уровня 1 означала бы, что на нулевом уровне узлы конкретных параметров связаны в основном с узлами конкретных линий (чем справедливей это утверждение, тем выше должно быть значение проходимости $R_{\text{в}}$). Связи $R_{\text{в}}$ верхнего уровня должны вырабатываться в процессе самообучения и отражать опыт сети. Точнее, связь на некотором уровне отражает «опыт» связей k-го уровня, узлы которых связаны с верхним уровнем π-связями. Процесс формирования связей верхнего уровня может быть описан выражениями

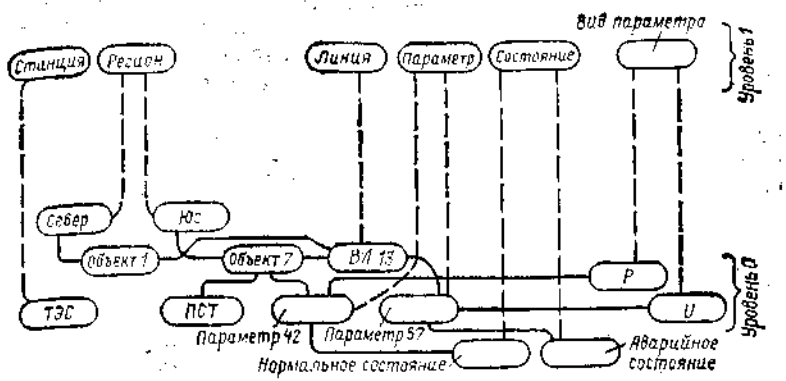


Рис. 3.3

$$R^{\tau+1}_{\text{в}\,(p,q),k+1} = R^{\tau}_{\text{в}\,(p,q),k+1}\,(1 - c_1 + R^{\tau}_{\text{э}\,(i,j)k}\,c_{2(k,k+1)});$$

$$R^{\tau+1}_{\text{э}\,(p,q),k+1} = R^{\tau}_{\text{э}\,(i,j),k} + c_{3(k,k+1)} \text{ при } (p_{k+1}, \pi, i_k)\,(q_{k+1}, \pi, j_k), \quad (3.7)$$

где k, $k+1$ — номера уровней; i, j — номера узлов уровня k; p, q — номера узлов уровня $k+1$; c_1 — коэффициент забывания связи; c_2 — коэффициент проекции между уровнями k и $k+1$ для связей общности; c_3 — то же для связей существования,

Запись (l_{k+1}, π, f_k) означает, что узлы l_{k+1} и f_k связаны π-связью. В соответствии с этими выражениями проходимость связи существования уровня k как бы проецируется вдоль межуровневых π-связей наверх — в связь общности уровня $k+1$. Поэтому такой алгоритм изменения связей может быть условно назван *проекцией вверх*. Такой процесс самообучения соответствует простейшей наивной индукции, т. е. с помощью проекции вверх осуществляется обобщение информации об отношениях, представленных в модели семантической сети.

Связи общности верхних уровней могут использоваться для корректировки связей существования нижних уровней. Этот процесс, который можно назвать *проекцией вниз*, будет рассмотрен ниже.

4. Взаимодействие алгоритмов обучения. Рассмотренные выше алгоритмы обучения и самообучения используются совместно при функционировании модели семантической сети. Конкретная информация, автоматически поступающая в ПсА (в частности, от устройств телемеханики), отражается на нижнем, нулевом уровне семантической сети с помощью алгоритма *Наблюдение* (напомним, что на нулевом уровне имеются только связи существования). Сообщения человека-диспетчера, поступающие через диалоговую подсистему, могут быть неявно адресованы, а следовательно, и отражены в разных уровнях семантической сети. При этом на нулевом уровне изменяются связи существования, на верхних уровнях могут изменяться как связи существования, так и связи общности (в зависимости от входных сообщений).

Так, человек сообщает системе о фактэх и закономерностях внешнего мира. С помощью алгоритма *Рефлекс* должны изменяться связи типа существования, имеющие семантику каузативных отношений (связи вида *причина — следствие*). В основном эти изменения должны осуществляться на нулевом уровне. В результате рефлекторного изменения каузативных связей существования модель сети приобретает способность прогнозировать временну́ю последовательность событий внешнего мира, и, следовательно, тем самым повышается скорость реакции модели на эти события. Алгоритм *Межуровневой проекции* обеспечивает накопление на верхних уровнях модели сети обобщенного опыта в виде связей общности. Этот опыт может быть использован, в частности, для обеспечения достоверности конкретной информации, вводимой в ПсА.

Ранее рассмотренные алгоритмы функционируют при априорно заданных узлах сети и π-связях между узлами разных уровней. Полное обучение может быть достигнуто при возможности образования в процессе функционирования новых узлов и новых отношений классификации. Это осуществляется блоком формирования понятий, изображенным на рис. 3.1.

3.5. Виртуальная семантическая сеть

Рассмотренный способ моделирования семантической сети на ЭВМ имеет ряд существенных недостатков при использовании его для реальных объектов управления (в частности, семантических сетей энергосистем).

В первую очередь следует назвать большой объем семантических сетей для реальных объектов: для представления диспетчерских знаний об энергосистеме (притом только для целей оперативного управления) требуется моделировать семантическую сеть с числом узлов около 10^4 и примерно с таким же числом связей. Одновременное представление всей семантической сети в оперативной памяти мини-ЭВМ практически невозможно. Хранение данных о семантической сети на внешних или промежуточных носителях не обеспечивает работы сети в реальном масштабе времени. Но даже если бы удалось поместить все массивы, необходимые для представления полной сети в ОЗУ ЭВМ, остались бы существенные трудности, связанные с поиском информации в большой сети с множественными типами отношений. При поиске необходимой информации ненужная в данном поиске информация, представленная в сети, будет создавать информационный шум, искажая результаты. В методе активных семантических сетей это проявляется как влияние возбуждения неактуальных, слабо возбужденных узлов через связи, ненужные для данного поиска. При использовании других методов придется применять переборные процедуры, что не позволяет работать в реальном масштабе времени.

Преодолеть подобные трудности можно разбиением семантической сети на отдельные части и представлением в ОЗУ ЭВМ в каждый момент работы системы только выделенной части сети. Будем называть такую выделенную часть сети *проблемной семантической сетью* (ПСС).

Для выборки ПСС из полной сети необходимо ввести разбиение семантической сети на отдельные подсети и определить механизм сборки ПСС из этих подсетей при поступлении входной информации. Массивы подсетей могут храниться во внешней памяти ЭВМ (например, на магнитных дисках) и при сборке ПСС вызываться в ОЗУ.

Небезразлично, каким образом осуществить разбиение сети. Подсеть как элемент системы знаний должна обладать определенной целостностью и иметь собственные механизмы управления для осуществления элементарных процедур логического вывода. Ранее было введено понятие уровня сети и семантическая сеть была представлена в виде иерархической конструкции.

Введем теперь понятие *семантической группы* узлов сети (сокращенно СГ). Узлы одного уровня, л-связи которых входят в один узел ближайшего верхнего уровня, принадлежат одной СГ. Именем этой СГ является имя того узла верхнего уровня, в который входят связи от узлов, принадлежащих СГ нижнего уровня. Для фрагмента сети, состоящего из узлов нулевого уровня, изображенного на рис. 3.4:

узлы *север* и *юг* принадлежат к СГ РЕГИОН;

узлы *параметр 42* и *параметр 57* — к СГ ПАРАМЕТР;

узлы *P* и *U* — к СГ ВИД ПАРАМЕТРА;

узлы *нормальное состояние* и *аварийное состояние* — к СГ СОСТОЯНИЕ и т. д.

Внутри каждой СГ вводится порядковая нумерация узлов, причем имя узла на языке СГ (ЯСГ) состоит из имени СГ и номера узла. Например, вместо имени узла *нормальное состояние* (язык ЯДРО) получим *состояние 1* (ЯСГ), соответственно вместо *аварийное состояние — состояние 2*, вместо *север — регион 1* и т. д.

Проведем разбиение сети на подсети (назовем их *проблемными сферами*, сокращенно ПС) таким образом, чтобы в каждой подсети существовал только один вид ситуативных отношений. При этом СГ узлов входит в выделяемую ПС полностью, если хотя бы один из узлов этой СГ связан σ-связью данного типа. Если некоторые узлы СГ связаны σ-связями одного типа, а другие узлы — σ-связями другого типа, то данная СГ должна быть размножена между соответствующими ПС. Связи между различными ПС задаются таблицей связи ПС (ТСПС). Поскольку связи между ПС образованы размножившимися СГ, то порядок нумерации узлов, при котором сохраняется порядок в разных ПС, обеспечивается, если в ТСПС задать номера связанных ПС и номера связанных СГ в этих сферах (отдельно хранятся сведения об СГ, в частности, количество узлов в СГ). В ПС могут входить также л-связи, существовавшие между узлами разных уровней соответствующих СГ, вошедших в эту ПС. Таким образом, в общем случае ПС представляет собой иерархическую структуру с однотипными σ-связями внутри всех уровней и с л-связями между уровнями. Как правило, в ПС на каждом уровне входят две СГ. Примеры возможных ПС в семантической сети фактуальных знаний об энергосистеме (см. рис. 3.2, 3.3) сведены в табл. 3.1.

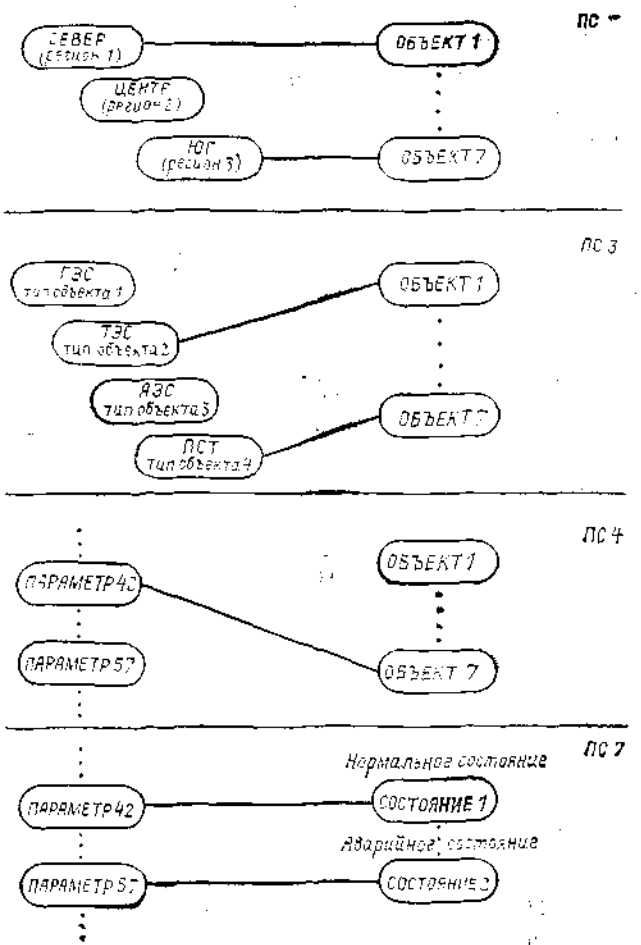


Рис. 3.4

В табл. 3.1 для упрощения приведены одноуровневые ПС. Фрагменты нескольких таких ПС (ПС 1, 3, 4, 7) приведены на рис. 3.4.

Для определения порядка соединения ПС в общей семантической сети необходимо ввести дополнительно специальную ПС *a*, в которой имеются два уровня: каждый

Таблица 3.1

№ п/п	Имя ПС	Отношение	СГ
1	Объект — регион	ПР, ВО	Объект, регион
2	Объект — линия	ЛПО, ОСЛ	Объект, линия
3	Объект — тип объекта	ЯТО, ИТО	Объект, тип объекта (Узлы СГ тип объекта: ГЭС (ЯДРО) — тип объекта 1 ТЭС (ЯДРО) — тип объекта 2)
4	Объект — параметр	ИП, ИО	Объект, параметр
5	Линия — параметр	ИП, ИЛ	Линия, параметр
6	Параметр — вид параметра	ИВ, ИП	Параметр, вид параметра
7	Параметр — состояние параметра	НС, БСП	Параметр, состояние

узел нулевого уровня ПС соответствует одной СГ, причем σ-связи между узлами СГ отражают имеющиеся в сети σ-связи между узлами этих СГ; каждый узел первого уровня соответствует одной ПС, а π-связи между узлами нулевого и первого уровней отражают связи между СГ и ПС (вхождения СГ в ПС). Пример ПС α для сети из рис. 3.2 и 3.3 приведен на рис. 3.5. Обратим внимание на горизонтальные σ-связи между узлами нулевого уровня. Эти связи обеспечивают «понимание» умолчаний во входном языке. Например, если возбужден узел *регион* (СГ1), то автоматически возбуждается узел *объект* (СГ2), по-

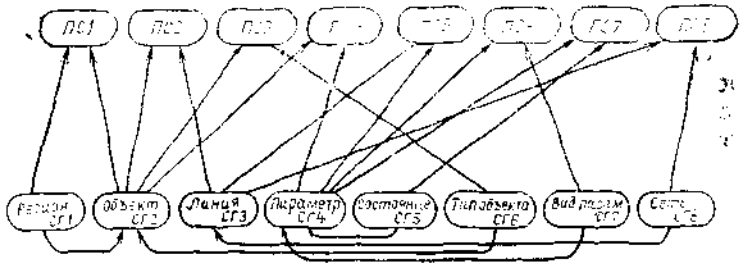


Рис. 3.5

скольку СГ1 однозначно связан в семантической сети только с СГ2. Аналогичные связи существуют между другими узлами — СГ *сеть→линия, тип объекта→объект, вид параметра→параметр, состояние→параметр* и др. Для принятия решения о том, какие ПС должны быть вызваны в ОЗУ ЭВМ для сборки ПСС, необходимо исходя из контекста (из текста запроса) возбудить узлы СГ в ПС α, определить значения возбуждений узлов ПС и сравнить эти последние с порогом p_α. Если выполняется условие

$$x_{\text{ПС}_i} - p_\alpha > 0, \qquad (3.8)$$

то ПС_i должна быть вызвана в оперативную память. Порог задается заранее так, чтобы условие (3.8) выполнялось, только если возбуждены все узлы СГ, связанные с данным узлом ПС межуровневыми связями.

Пример 3.7. Для ответа на запрос 2 из примера 3.1 в ПС α начальное возбуждение получают узлы СГ: СГ4 (*параметр*) и СГ5 (*состояние*), на уровне 1 возбудится выше порогового значения узел ПС7. В ответ на запрос (10) из примера 3.6 начальное возбуждение в ПС α получат узлы СГ1 (*регион*) и СГ5 (*состояние*). Это начальное возбуждение, распространяясь по внутриуровневым связям *регион — объект* и *состояние — параметр*, возбудит узлы СГ2 (объект) и СГ4 (*параметр*) (эти понятия не были упомянуты в запросе, они умалчивались). На уровне 1 возбудятся выше порогового значения узлы ПС1, ПС4, ПС7. Из ПС с этими номерами и осуществляется СБОРКА ПСС в ОЗУ ЭВМ.

Таким образом, полная семантическая сеть нигде в ЭВМ не хранится. На внешних носителях хранятся только ПС и ТСПС. В соответствии с запросом (или с контекстом ситуации) в ОЗУ собирается релевантная проблемная сеть. Для пользователя, задающего запрос, или для программы, обратившейся к ДВОЭ, система ведет себя так, как будто полная сеть представлена в ОЗУ ЭВМ. Такой принцип хранения и сборки семантической сети назовем *виртуальной семантической сетью* (точнее, речь идет не о сети, а о ее модели). Количество ПС, из которых собирается ПСС, значительно меньше, чем общее количество ПС (в реальной системе до 64 ПС, а ПСС собирается максимум из 4 ПС). Важно отметить, что сборка ПСС только из ПС, релевантных запросу, кроме обеспечения работы системы знаний в реальном масштабе времени способствует уменьшению информационного шума.

Информация, вводимая в ПС от ПсЖ, адресуется непосредственно в нужную проблемную сферу: $\{\varphi, i, j, r\}$, где Φ — номер ПС; i и j — номера узлов в проблемной сфере Φ; r — проходимость связи между узлами i и j, соответ-

ствующая силе отношения типа φ между денотатами узлов i и j.

Сообщение, поступающее через диалоговую систему на входном языке ЯДРО, после синтаксического анализа представляется в форме $\{\Phi, I, J, R\}$, где Φ — имя ситуативного отношения; I и J — имена понятий, между которыми изменяется отношение; R — сила устанавливаемого отношения. Задачей поверхностного семантического анализа входных сообщений является преобразование

$$\{\Phi, I, J, R\} \rightarrow \{\varphi, i, j, r\}.$$

Это преобразование осуществляется с помощью каталогов отношений и понятий по алгоритму, представленному на рис. 3.6. Дальнейшее семантическое представление аналогично представлению информации, поступающей от ПсЖ. Такой ввод информации в ПС соответствует алгоритму *Наблюдение*. На верхних уровнях ПС проходимости связей отражают опыт наблюдений за отношениями понятий на нижнем уровне. Этот опыт формируется по алгоритму *Межуровневая проекция вверх*. Для контроля достоверности вводимой в ПС информации целесообразно использовать индуктивно накопленный опыт связей общности R верхних уровней проблемных сфер. Для этого осуществляется *Межуровневая проекция вниз*, соответствующая простейшему дедуктивному выводу

$$R^{*}_{(i,j),k} = R_{\forall (p,j),k+1}\, c_{4,(k+1),k}, \qquad (3.9)$$

где $R^{*}_{(i,j),k}$ — спроецированное на k-й уровень значение проходимости связи; $R_{\forall (p,j),k+1}$ — проходимость связи типа общности на уровне $k+1$; $c_{4,(k+1),k}$ — коэффициент проекции вниз; в частности, можно принять $c_4 = 1$.

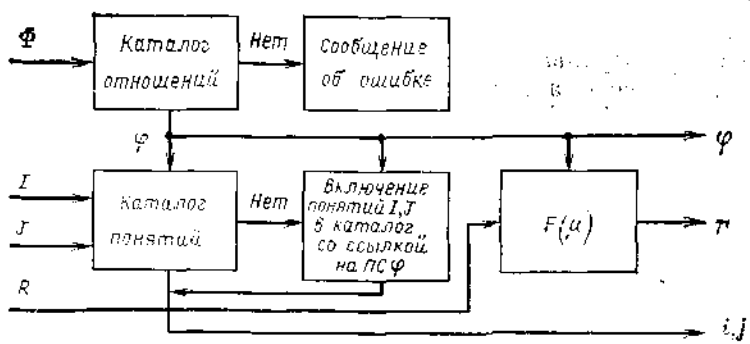
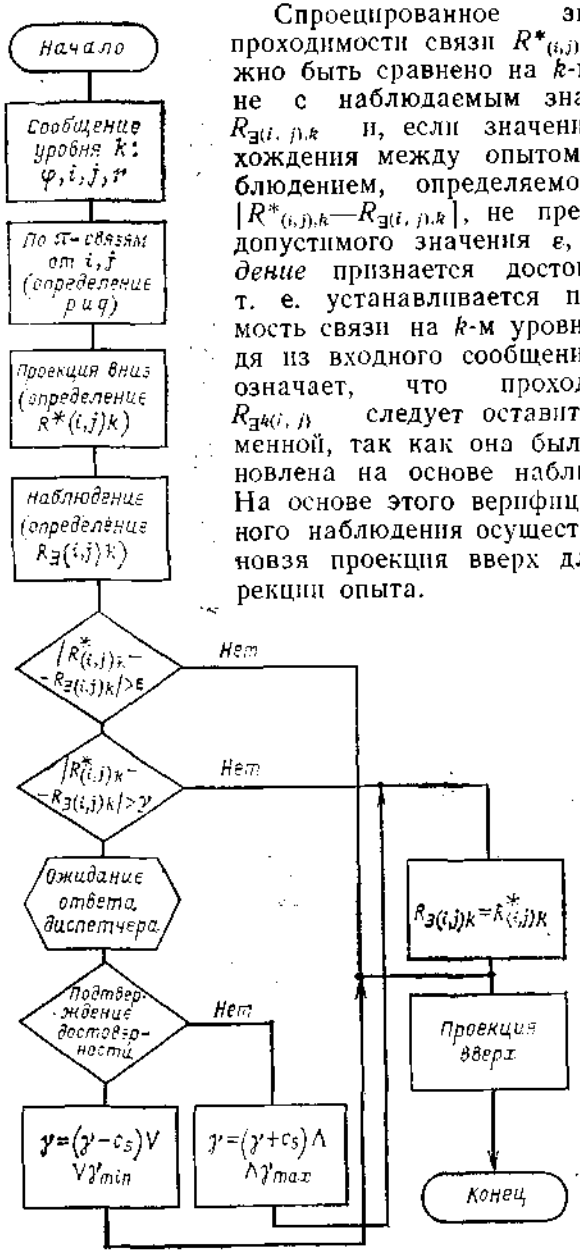


Рис. 3.6



Спроецированное значение проходимости связи $R^{*}_{(i,j),k}$ должно быть сравнено на k-м уровне с наблюдаемым значением $R_{\exists(i,j),k}$ и, если значение расхождения между опытом и наблюдением, определяемое как $|R^{*}_{(i,j),k} - R_{\exists(i,j),k}|$, не превышает допустимого значения ε, *наблюдение* признается достоверным, т. е. устанавливается проходимость связи на k-м уровне исходя из входного сообщения. Это означает, что проходимость $R_{\exists k(i,j)}$ следует оставить неизменной, так как она была установлена на основе наблюдения. На основе этого верифицированного наблюдения осуществляется новая проекция вверх для коррекции опыта.

Рис. 3.7

Если же $|R^*_{(i,j),k} - R_{\exists(i,j),k}| > \varepsilon$, нужно произвести сравнение этого расхождения между опытом и наблюдением с порогом уверенности γ модели в своем опыте. Если этот порог превзойден, проблемная сфера через диалоговую систему инициирует запрос к диспетчеру о правильности исходного сообщения. В зависимости от того, подтверждает или нет диспетчер свое исходное сообщение (или информацию, полученную от ПсЖ), в качестве проходимости связи нижнего уровня принимается наблюдение или опыт (результат проекции вниз). В зависимости от ответа диспетчера корректируется и порог уверенности γ. Если диспетчер настаивает на достоверности исходного сообщения, этот порог уменьшается (на константу c_5), если же диспетчер разделяет сомнения в достоверности входной информации, следует увеличить порог уверенности. Исходное значение γ не должно быть слишком большим, т. е. близким к 1 (иначе в сомнительных случаях система вообще не будет обращаться к человеку-диспетчеру), но не должно быть и слишком малым, близким к нулю (иначе система никогда не будет принимать решения о достоверности входной информации самостоятельно).

В заключение семантических преобразований при вводе сообщения в проблемную сферу производится процедура *проекции вверх*. Рассмотренный алгоритм семантических преобразований представлен на рис. 3.7. В этом алгоритме сочетаются различные методы обучения и самообучения активных семантических сетей, а именно: *наблюдение, межуровневая проекция вниз* и *межуровневая проекция вверх*.

3.6. Вывод ответов на вопросы

В процессе анализа вопросов и вывода ответа происходит несколько этапов преобразования и представления информации, причем каждому такому этапу соответствует свой язык представления. Переходам между уровнями представления при этом соответствуют процедуры трансляции с языка одного уровня на язык другого уровня (табл. 3.2). При этом внешними языками системы являются только входной язык ЯДРО и выходной язык ИМЯ, остальные (ЯСГ, ЯПС, ЯСС) — внутренние языки. Процесс преобразований лингвистической информации схематично может быть представлен следующим образом:

$$текст_{\text{ЯДРО}} \rightarrow смысл_{\text{ЯСС}} \rightarrow текст_{\text{ИМЯ}}.$$

Таблица 3.2

Уровни представления информации	Язык представления информации	Этапы преобразования информации	Используемые массивы
1. Поверхностный синтаксический уровень	Входной язык (ЯДРО)	Синтаксический анализ ЯДРО → ЯСГ	Словарь языка ЯДРО; таблица семантических групп ТСГ
2. Глубинный синтаксический уровень	Язык семантических групп (ЯСГ)	Поверхностный семантический анализ ЯСГ → ЯПС	$ПС_\alpha$; ТСГ
3. Поверхностный семантический уровень	Язык проблемных сфер (ЯПС)	Семантическое представление ЯПС → ЯСС	$ПС_1, \ldots, ПС_N$; ТСПС
4. Глубинный семантический уровень	Язык семантических сетей (ЯСС)	Синтаксический синтез ответа ЯСС → ИМЯ	Таблица семантических групп ТСГ; массив технологических наименований; массивы схем, таблиц
5. Уровень представления информации для ответа	Язык технологических наименований (ИМЯ)		

В функцию синтаксического анализа вопроса входят следующие процедуры:

трансляция ЯДРО→ЯСГ; выявление темы и ремы; декомпозиция вопроса на подвопросы.

В языке семантических групп ЯСГ представлены имена семантических групп ИСГ и имена элементов семантических групп ЭСГ, причем

$$<\text{ЭСГ}> := <\text{ИСГ}>, <\text{НОМЕР}>,$$

где НОМЕР — числовое выражение порядкового места элемента в данной СГ. Для удобства записи вводятся сокращенные обозначения ИСГ, например: регноны — R, объекты — O, линии — L, параметры — P, состояния — S, типы объектов — T, виды параметров W.

При трансляции ЯДРО→ЯСГ слова и словосочетания входного языка заменяются обозначениями (кодами) ИСГ и ЭСГ.

Пример 3.8. Примеры перевода: СЕВЕР — *R1*, ГЭС 12—О22, РЕГИОН — *R*, ГЭС — Т2, СОСТОЯНИЕ АВАРИЙНОЕ — *S2*.

Пример 3.9. Пример трансляции вопроса:

какой объект регион север состояние аварийный?→KOR1S2?

(символом К обозначено вопросительное слово КАКОЙ).

Для выявления темы и ремы ЯСГ-вопроса, не содержащего придаточную часть, используются следующие правила:

а) ремой является ПСГ, непосредственно следующее после вопросительного слова (*К*);

б) если после вопросительного слова следует ЭСГ, то ремой является ИСГ, подчиненной СГ, к которой принадлежит первый ЭСГ (следующий за вопросительным словом), причем этот ЭСГ входит в тему; отношения подчинения между СГ заданы заранее, например СГ *типы объектов* (Т) подчинена СГ *объекты* (О):

$$T \rightarrow O;$$

в) в вопросе без придаточной части не может быть более одной ИСГ;

г) все ЭСГ, содержащиеся в вопросе, включаются в тему.

Примеры применения правил приведены в табл. 3.3.

Декомпозиция простого (без придаточной части) вопроса определяется следующими правилами:

д) число подвопросов равно числу ЭСГ в теме, причем в тему каждого подвопроса входит один и только один ЭСГ темы основного вопроса;

е) рема всех подвопросов совпадает с ремой основного вопроса.

Примеры декомпозиции приведены в табл. 3.4.

Декомпозиция вопроса с придаточной частью выполняется в соответствии со следующими правилами (для простоты считаем, что в вопросе содержится только одна придаточная часть):

Таблица 3.3

Пример 3.10

Вопрос	Рема	Тема
KOR1S2?	O	R1, S2
KT2R1?	O (отношение подчинения СГ: $T \rightarrow O$	T2, R1
KORS2?	Вопрос ошибочен, так как содержит 2 ИСГ (O и R)	

Таблица 3.4

Пример 3.11

Основной вопрос	Декомпозиция	Рема	Тема
KOR1S2?		O	R1, S2
	KOR1?	O	R1
	KOS2?	O	S2
KT2R1?		O	T2, R1
	KOT2?	O	T2
	KOR1?	O	R1

ж) придаточная часть, отделенная от основного вопроса запятой, выделяется в отдельный подвопрос;

з) рема придаточного подвопроса входит в тему основного со специальным обозначением *; это означает, что элементы ответа на подвопрос должны дизъюнктивно войти в рему основного вопроса.

Пример приведен в табл. 3.5.

Поскольку каждый подвопрос является простым вопросом, то выявление его темы и ремы определяется правилами «а»—«г».

Рассмотренные выше правила синтаксического анализа изложены в несколько упрощенном виде по сравнению с тем, как они реализованы в системе ДВОЭ.

Основной функцией поверхностного семантического анализа вопросов в системе является трансляция с языка семантических групп в язык проблемных сфер ЯСГ→ЯПС. Поскольку проблемные сферы семантической сети представляют собой различные области знаний (предварительно классифицированные), то поверхностный семантический анализ означает выявление тех областей знаний, которые потребуются при выводе ответа на вопрос.

Таблица 3.5

Пример 3.12

Вопрос	Декомпозиция	Рема	Тема
KO, RS2?	KRS2?	R	S2
	KOR*?	O	R*

Пример 3.13

Таблица 3.6

Основной вопрос (ЯСГ)	Декомпозиция (ЯСГ)	Рема (ЯСГ)	Тема (ЯСГ)	ЯПС		ПС
				Рема	Тема	
KOR1S2?		O	R1, S2			
	KOR1?	O	R1	№ПС-0 №У:0—50	№ПС-0 №У:51	1
	KOS2?	O	S2	№ПС-0 №У:100—150	№ПС-1 №У:102	4, 7
KT2R1?		O	T2,			
	KOT2?	O	T	№ПС-0 №У-0—50	№ПС-0 №У:52	3
	KOR1?	O		№ПС-0 №У:0—50	№ПС-0 №У:51	1
KO, RS2?	KRS ?	R	S2	№ПС-0 №У:51—54	№ПС-? №У:102	1; 4; 7
	KOR*?	O		№ПС-0 №У:0—50	№ПС-0 №У:51—54	1

На языке проблемных сфер каждый узел семантической сети обозначается номером проблемной сферы № ПС и номером узла в этой проблемной сфере № У. Входящая в рему вопроса семантическая группа узлов должна выражаться на ЯПС номером ПС и граничными (начальным и конечным) номерами узлов. Определение ПС, необходимое для ответа на поставленный вопрос, по специальной ПС α было рассмотрено ранее. Далее, пользуясь таблицей семантических групп, система определяет № У для темы вопроса и граничные номера узлов для СГ ремы вопроса. Преобразования ЯСГ→ЯПС необходимо выполнить для каждого подвопроса, выделенного на этапе синтаксического анализа при декомпозиции вопроса. Тема вопроса с придаточной частью, помеченная знаком *, транслируется на ЯПС аналогично реме вопроса (номерами граничных узлов СГ), но знак * при трансляции сохраняется.

Примеры трансляции при поверхностном семантическом анализе даны в табл. 3.6.

Правая графа таблицы содержит абсолютные номера ПС, из которых предстоит собрать проблемную семантическую сеть ПСС. В графах ЯПС (тема и рема) приводится относительный номер ПС (№ПС). Так, если ПСС собирается из ПСС (абсолютные номера) 1; 4; 7, а в графе ЯПС (тема) приведен относительный номер №ПС-2, то это указывает на ПС с абсолютным номером 7. На уровне поверхностного

семантического анализа осуществляется также сборка ПСС из выбранных проблемных сфер. ПСС собирается отдельно для каждого подвопроса.

Основной функцией семантического представления (глубинного семантического анализа) является трансляция с языка проблемных сфер на язык семантических сетей (ЯПС→ЯСС). Язык семантических сетей есть язык значений возбуждения узлов семантической сети и значений проходимости связей между узлами. Для осуществления семантического представления следует выполнить следующие операции:

а) сообщить начальное возбуждение узлам, соответствующим теме вопроса (подвопроса); в простейшем случае начальное возбуждение узла можно принять максимальным, $x_0=1$;

б) осуществить пересчет возбуждения в ПСС, при котором исходя из начального возбуждения узлов определяется возбуждение узлов семантической группы, соответствующей реме вопроса;

в) выполнить операцию обобщенной конъюнкции (выбор минимального значения) над значениями возбуждений узлов СГ ремы для каждого из подвопросов;

г) результат этих операций сравнивается с *порогом ответа* p_0, если выполняется условие и $x_i-p_0>0$; узел i входит в объем понятия ответа на поставленный вопрос.

В простейшем случае величина p_0 постоянна и определена заранее. Но имеется возможность изменять порог ответа в зависимости от нечетких указаний о степени точности ответа, содержащейся в вопросе.

Операции «а»—«г» излагались для семантического представления при вопросах без придаточной части. Если в вопросе на языке ЯПС имеется придаточная часть (ссылка на группу узлов, отмеченную знаком *), то операции выполняются в следующем порядке. Сначала обрабатывается подвопрос, полученный из придаточной части (его тема не помечена знаком *); эта обработка осуществляется так, как описано выше. Затем обрабатывается подвопрос, тема которого помечена знаком *. За начальное возбуждение узлов темы этого подвопроса принимается результирующее возбуждение узлов ремы ранее рассмотренного подвопроса. Поскольку эти узлы входят в одну и ту же СГ, такая процедура практически эквивалентна дизъюнктивным операциям при отдельном возбуждении каждого из этих узлов. Естественно, одновременное возбуждение всех узлов ремы существенно ускоряет процесс вычислений.

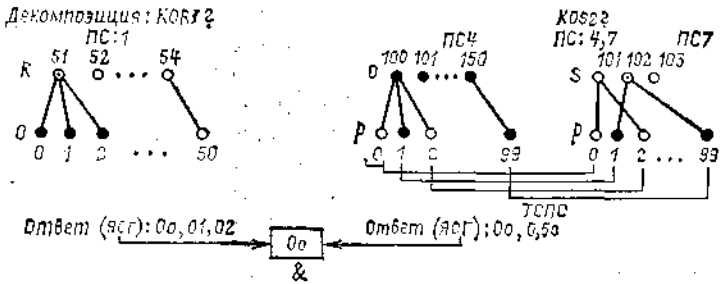


Рис. 3.8

Пример 3.14. На рис. 3.8 приведен пример семантического представления. Вопрос на ЯСГ имеет следующий вид: KOR1S2? Общий ответ на поставленный вопрос на ЯСГ выглядит О0. На рисунке использованы следующие обозначения: ⊙ помечает узлы с начальным возбуждением, а ● соответствует узлам, возбужденным при пересчете ПСС.

Простейшим *синтезом ответа является вывод списка* имен (технологических наименований) узлов сети, которые в результате семантического представления вошли в объем понятия ответа. В других случаях целесообразно выводить не сами названия узлов, а таблицы или схемы, обозначенные этими именами. При этом необходимо организовать процедуры листания информации, выводимой на экраны дисплеев.

Изменение порядка слов в вопросе и его влияние на изменение смысла вопроса учитываются при синтаксическом анализе, а именно при делении вопроса на тему и рему.

Наличие в вопросе придаточной части определяется на этапе синтаксического анализа, но окончательно учитывается при семантическом представлении путем изменения порядка операций при пересчете ПСС для вопроса с придаточной частью.

Анафорические связи (вопросы 8, 9 из примера 3.5) могут быть учтены за счет возбуждений узлов, входящих в рему предыдущего вопроса 8, которые становятся начальными возбуждениями узлов темы последующего вопроса 9. Другие умолчания учитываются на этапе поверхностного семантического анализа.

Метонимия (и, в частности, синекдоха) может быть учтена путем распространения возбуждений в ПСС по π-связям. Синонимические конструкции понимаются за счет суммирования в узлах ПСС возбуждений других узлов через проходимости связей и выбора для ответа наиболее возбужденных узлов.

Определительные наречия в вопросе могут быть учтены путем изменения порога ответа p_0 в соответствии с методами теории нечетких множеств. При этом проходимости связей от узлов, получающих начальное возбуждение, должны быть пропорциональны силе соответствующего отношения. Для вопроса 12 из примера 3.6 значения проходимости связей узла *состояние аварийное* (2) с узлами *параметров* должны быть пропорциональны степени нарушения соответствующим параметром технологического аварийного предела, а порог p_0 должен определяться наречием, употребленным в вопросе. Очевидно, что для наречия *сильно* порог p_0 должен быть выше, чем для наречия *слабо*.

Описанная вопросно-ответная система (точнее, система фактуальных знаний) реализована на ЭВМ ЕС-1010. Основные характеристики системы приводятся ниже:

язык программирования АССЕМБЛЕР ЕС-1010; число *проблемных сфер* — до 64; *хранение* массивов ПС — на магнитных дисках; число ПС, из которых собирается ПСС, — до 4; хранение собранной ПСС — МОЗУ; число узлов в одной ПС — до 255; число связей в одной ПС — до 255; реактивность — около 1 с/подвопрос; объем математического обеспечения — около 4 Кбайт; объем массивов — около 64 Кбайт (максимально).

3.7. Получение рекомендаций по управлению

Анализ ситуации в объекте управления является важнейшей функцией диспетчерского управления, в частности управления энергосистемами. Можно определить анализ ситуации как этап управления, начинающийся выявлением необходимости изучения текущей ситуации с целью оперативного воздействия на объект управления и заканчивающийся составлением плана такого воздействия. Этот план сообщается диспетчеру в виде совета. Окончательное решение о воздействии на объект и само воздействие (через соответствующие исполнительные органы) в автоматизированных системах диспетчерского управления осуществляет человек-диспетчер. Основным содержанием анализа ситуации является последовательность исследовательских действий по уточнению и классификации характеристик ситуации. Эти действия регламентируются диспетчерскими инструкциями, составленными для ряда типичных аварийных ситуаций в объекте управления (на-

пример, для энергосистемы: понижение частоты, понижение напряжения, перегрузка транзитных линий электропередачи, асинхронный ход, разделение энергосистемы на несинхронно работающие части). Инструкции составляются, как правило, в обобщенной форме, и их реализация требует от диспетчера соотнесения обобщенных знаний, содержащихся в инструкциях, с конкретной информацией о текущих характеристиках объекта (т. е. с фактуальными знаниями). Таким образом, в критических, аварийных ситуациях от диспетчера требуется припоминание большого объема инструктивной и справочной информации. Автоматизация действий диспетчера по анализу ситуации может быть достигнута путем введения в систему адаптивного поведения блока актуальных знаний — знаний о действиях (блок *9* на рис. 3.1). Этот блок должен хранить знания о действиях в виде ряда сценариев операций, необходимых для анализа ситуации. Сценарии разрабатываются на основе диспетчерских инструкций, причем каждый сценарий соответствует одному из описанных в инструкции классов ситуации в объекте управления. В процессе своей реализации каждый сценарий может обращаться к блоку фактуальных знаний (блок *8* на рис. 3.1) с запросами, уточняющими ситуацию, причем от ответов на эти запросы зависит порядок дальнейшего выполнения сценария. В случае необходимости сценарий через диалоговую подсистему (блок *5* на рис. 3.1) может обращаться к человеку-диспетчеру за информацией, которой нет в модели внешнего мира системы.

Запросы, задаваемые в процессе внутреннего диалога системой актуальных знаний системе фактуальных знаний, целесообразно передавать непосредственно на языке проблемных сфер ЯПС, однако при составлении сценариев удобно использовать входной язык ЯДРО с последующей трансляцией ЯДРО→ЯСГ→ЯПС.

Обобщенная структура сценария содержит следующие блоки: проверка условия запуска сценария, проверка условия применимости сценария, уточнение ситуации, составление плана воздействия на объект управления (выработка совета диспетчеру), выдача результатов анализа.

Наряду с элементами, формирующими внешние или внутренние запросы, сценарий содержит элементы обработки ответов. Определенный набор элементов обработки дополняет ЯПС и вместе с ним составляет язык сценариев анализа ситуации (ЯСАС).

На рис. 3.9 показан фрагмент сценария анализа ситуации «раздел энергосистемы». Для краткости и удобства обозначений запросы записаны не на ЯПС, как они в действительности представлены в сценарии, а на ЯСГ с указанием номера ПС. Блоки *1—3* сценария проверяют его применимость (сценарий применим, если число связанных линиями объектов меньше заданного числа n), остальные блоки выявляют, на какое число участков разделялась система и какие объекты входят в каждый из участков. Блок обработки *8 Создание узла* Y_i работает с ПС *объекты-участки раздела*, используемой специально для этого сценария. Выявив наличие отделившегося участка, этот блок создает в ПС узел Y_i (i— номер участка) и соединяет с этим узлом все узлы СГ *объекты*, соответствующие объектам, которые принадлежат i-му участку.

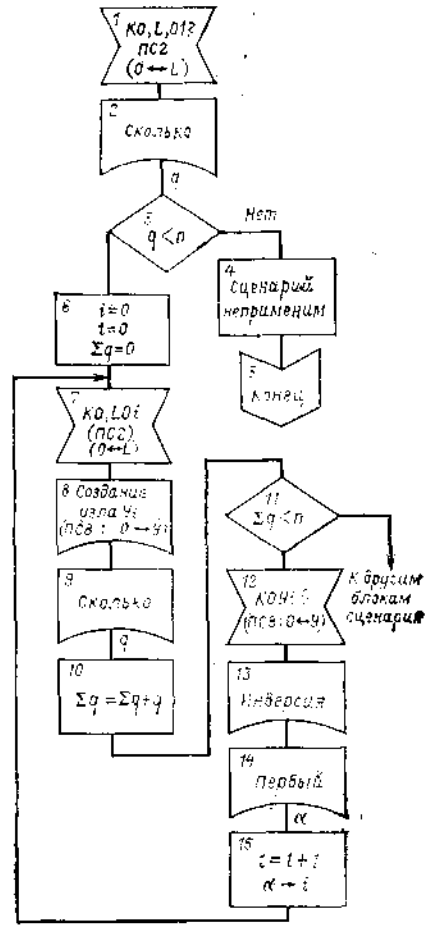


Рис. 3.9

В приведенном фрагменте используются и другие блоки обработки: *Сколько* (*2, 9*) — этот блок подсчитывает число узлов СГ ремы запроса, возбужденне которых выше порогового; *Инверсия* (*13*)—этот блок сообщает максимальное возбуждение минимально возбужденным уалам СГ ремы и, наоборот, уменьшает до минимума возбуждение наиболее возбужденных узлов; *Первый* (*14*) — определение номера первого узла СГ ремы, возбуждение которого превышает порог. Существуют и другие блоки обработки. Кроме того, на рисунке использованы следующие обозначения: l — число объектов энергосистемы; n — число участков раздела.

При принятии решений по управлению в энергосистеме наиболее часто встречается задача типа распределения ресурсов при наличии нескольких взаимодействующих критериев. Точное решение такой задачи пока невозможно. В системе ДВОЭ ее решение происходит на основании активной семантической сети. На рис. 3.10 показан фрагмент такой сети. Сеть имеет четыре уровня: I — уровень восприятия, II — уровень критериев, III — уровень драйвов (стремлений к действиям), IV — уровень действий. Стрелками обозначены возбуждающие связи. Тормозящие связи заканчиваются кружком. Возбуждение узлов первого уровня формируется в системе жесткого поведения как нечеткие переменные; например, возбуждение узла 1-2 тем больше, чем больше параметров, нарушающих технологические пределы, возбуждение узла 1-5 увеличивается пропорционально количеству отказов и сбоев ЭВМ в определенный интервал времени. Возбуждение узлов 1-1, 1-2, 1-3 определяется программным *блоком обработка* ПсЖ (*см. рис. 3.1*), узла 1-4 — блоком *сбор*, узлов 1-5, 1-6, 1-7, 1-8, 1-9 — супервизором. При этом принято

$$x_{1\text{-}1} + x_{1\text{-}2} = x_{1\text{-}5} + x_{1\text{-}6} = x_{1\text{-}7} + x_{1\text{-}8} = x_{max} = 1;$$

$$x_{1\text{-}9} = 1 - \operatorname{sign}(x_{1\text{-}7} - p) \vee 0,$$

где p — константа-порог $(0<p<1)$.

При заданных значениях возбуждения узлов уровня I по формуле рассчитываются значения возбуждений узлов уровней II и III. При этом по методу *Рефлекс* в процессе самообучения могут изменяться проходимости связей между уровнями II и III.

Узлы уровня IV соответствуют функциям программ, решение об инициировании которых должно приниматься. Узлы уровня III имеют ту же семантику, что и узлы уровня IV с теми же номерами, но возбуждения узлов уровня III (драйвов) отражают стремление системы инициировать соответствующую программу, т. е. произвести соответствующее действие. Некоторые узлы уровня восприятий непосредственно действуют на возбуждение узлов уровня драйвов, некоторые — через узлы уровня критериев. В общем случае один узел уровня критериев может возбуждать несколько узлов уровня драйвов (например, от узла I-2 возбуждаются узлы III-1, III-2 и III-3); окончательный выбор действия определяется текущим состоянием ресурса системы (связи от узлов I-7, I-8, I-9 к узлам уровня III). В конечном счете в каждый момент выбирается то дей-

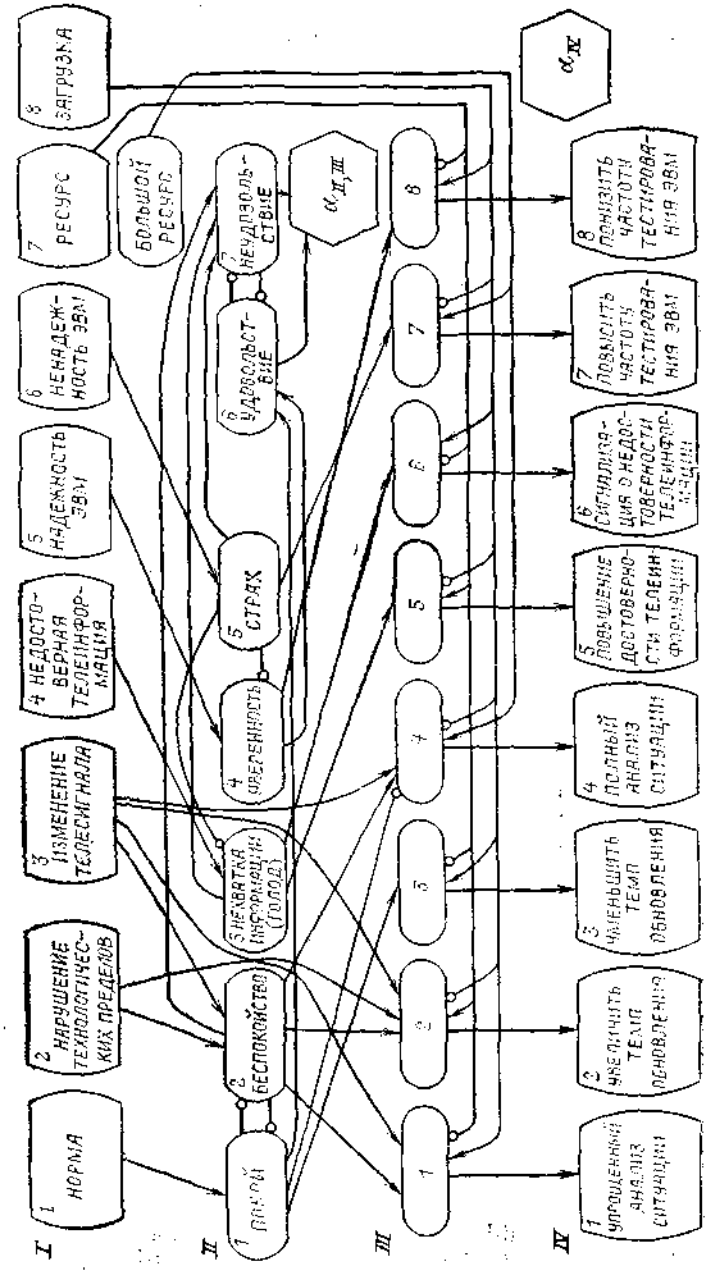


Рис. 3.10

ствие, чей узел уровня IV имеет наибольшее возбуждение (определяется активатором α_{IV}).

Выполнение инициированных программ отражается на загрузке процессора ЭВМ, т. е. на возбуждении узлов I-7, I-8, I-9; таким образом, в систему принятия решения вводится обратная связь. При имитационном моделировании системы принятия решений необходимо задавать коэффициенты загрузки процессора для каждой из программ, соответствующей узлу уровня IV сети, и длительность выполнения этой программы.

Рассмотренная система может, таким образом, стать основой для построения адаптивных супервизоров реального времени для информационных систем. В модели семантической сети (см. рис. 3.10) в виде возбуждения узлов представлены обобщенные оценки разнородной информации, влияющей на принятие решений. Эти оценки могут быть использованы для расширения возможностей диалога ПсА с диспетчером. Язык такого обобщения включает названия узлов семантической сети, названия уровней сети, некоторые служебные слова. Средством общения может быть функциональная клавиатура. Будем считать, что эти новые термины также входят в язык ЯДРО. При диалоге используется наличие активатора,

Таблица 3.7

Пример 3.15

№ п/п.	Запрос	Ответ	Примечание
1	ВОСПРИЯТИЕ?	НАРУШЕНИЕ ТЕХНОЛОГИЧЕСКИХ ПРЕДЕЛОВ ПАРАМЕТРОВ	Выбирается наиболее возбужденный узел уровня I и выводится в качестве ответа имя этого узла
2	КРИТЕРИЙ?	БЕСПОКОЙСТВО	То же для уровня II
3	БЕСПОКОЙСТВО?	НАРУШЕНИЕ ТЕХНОЛОГИЧЕСКИХ ПРЕДЕЛОВ ПАРАМЕТРОВ	Вопрос понимается как запрос о причинах беспокойства. Выбирается наиболее возбужденный узел из тех, которые порождают связи, входящие в узел II-2
4	ЖЕЛАНИЯ?	УВЕЛИЧИТЬ ТЕМП ОБНОВЛЕНИЯ	То же, что и п.п 1, 2 для уровня III

выбирающего наиболее возбужденный узел сети (или отдельного названного уровня). Семантически все запросы, инициирующие диалог с системой принятия решений, представляют собой запросы о каузативных отношениях.

В заключение приведем пример диалога (табл. 3.7).

ГЛАВА ЧЕТВЕРТАЯ

ДИАЛОГОВАЯ СИСТЕМА ДЛЯ ПЛАНИРОВАНИЯ И УПРАВЛЕНИЯ НА ТРАНСПОРТЕ (ДИСПУТ)

4.1. Общая характеристика системы

В этой главе дается описание диалоговой системы, положившей начало семейству аналогичных систем и получившей название ДИСПУТ. Специализированная диалоговая система ДИСПУТ входит в состав автоматизированной системы слежения за контейнерами. Она предназначена для получения информации из базы данных и решения в интерактивном режиме задач оперативного планирования и управления. Будут рассмотрены вопросы, относящиеся к организации системы, которая дает возможность рядовому пользователю работать со сведениями, хранимыми в базе данных.

Система ориентирована на пользователя, не являющегося специалистом в области программирования. Единственным требованием к нему является умение работать с видеотерминальным устройством или любым иным терминалом, с которого можно вводить запросы и на котором могут регистрироваться ответы.

Система рассчитана на ведение диалога с переменной инициативой. На начальной стадии работы вопросы задает машина, в результате чего пользователь выводится на требуемый режим работы. После этого инициативу на себя может взять человек.

Запросы к системе формулируются на естественном языке с лексикой, ограниченной тематикой задачи и фиксированными семантикой и прагматикой.

В качестве основы для построения интерпретатора естественного языка берется не модель языка, как это делается обычно, а модель мира, в котором должна функционировать эта система. Ограничения, накладываемые на модель мира множеством решаемых в ней задач, с неизбежностью влекут сужение языка, который используется для формулировки этих задач. Вместе с тем модель

проблемной ситуации, описываемая на таком языке, остается адекватной исходной картине мира. Таким образом, в естественном языке всегда можно выделить некоторое подмножество, где правила построения языковых конструкций управляются семантикой и прагматикой той проблемной области, для обслуживания которой они предназначены. При этом на человека, работающего с такой системой в рамках своей профессиональной деятельности, практически никаких ограничений в использовании естественного языка не накладывается.

Такой подход направлен на создание узко ориентированных на отдельную проблему и даже пользователя диалоговых систем. Только в этом случае удается избежать непреодолимых и многочисленных трудностей, связанных с построением и реализацией полных моделей языка или языкового поведения человека.

В упрощенном варианте рассматриваемый подход позволяет отказаться от традиционного решения комплекса вопросов, связанных с одновременной разработкой как структуры базы данных, так и интерпретатора естественноязыковых запросов. При этом база данных считается заданной, а интерпретатор расчленяется на два процессора: лингвистический и прагматический. Первый используется для перевода естественноязыковых конструкций во внутреннее формальное представление, инвариантное смыслу запроса. Во второй переносятся знания о прагматике задачи и, в частности, о структуре информационных массивов и системе управления базой данных.

Понятно, что работа по созданию специализированных систем трудоемка. Но только такие системы в настоящее время могут обеспечить эффективную работу в реальных условиях. Облегчить задачу создания таких систем призваны инструментальный и программный комплексы, реализующие технологию синтеза диалоговых систем высокого уровня. Однако рассмотрение таких средств выходит за рамки данной главы.

Работа по созданию системы ДИСПУТ облегчалась рядом моментов, из которых можно выделить следующие. Во-первых, анализ потенциальных запросов к системе показал, что используемая лексика насчитывает не более 300—400 единиц. Во-вторых, ни используемые слова, ни запросы не имеют многозначного толкования в рамках заданной прагматики. В-третьих, в системе используются всего три вида работы: вопросно-ответный режим, работа с моделями и оперативное внесение изменений в информационные массивы.

В результате была создана система, ориентированная на реального пользователя и реальные производственные условия. Подробные характеристики системы будут приведены в процессе описания ее функционирования. Здесь можно отметить, что она реализована на языках ФОРТРАН IV и АССЕМБЛЕР и что она может функционировать практически на любой ЭВМ единой серии в рамках ОС ЕС.

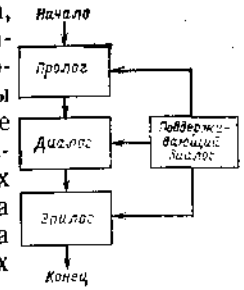


Рис. 4.1

Вся процедура общения пользователя с системой делится на ряд этапов. Каждому из этих этапов соответствует свой вид диалога (рис. 4.1).

В процессе настройки на диалог *Пролог,* который начинается с момента включения пользователем дисплея, осуществляются идентификация пользователя, выбор им режима работы (полный, сокращенный) и выбор типа решаемой задачи. Ниже показан один из фрагментов *Пролога.*

ЗДРАВСТВУЙТЕ!

СИСТЕМА ДИСПУТ К РАБОТЕ ГОТОВА. УКАЖИТЕ ТИП ЗАДАЧИ, С КОТОРОЙ ВЫ БУДЕТЕ РАБОТАТЬ, РЕЖИМ РАБОТЫ П/С И НАЖМИТЕ НА КЛАВИШУ «ВВ»:

1. ЗАКОНЧИТЬ РАБОТУ
2. ОТВЕТЫ НА ЗАПРОСЫ
3. РАБОТА С МОДЕЛЯМИ
4. ОБНОВЛЕНИЕ БАЗЫ

ВАШ ОТВЕТ: 2С

Ответ пользователя свидетельствует, что он выбрал сокращенный режим работы с задачей «Ответы на запросы».

После подготовки и настройки системы, осуществляемой автоматически, происходит собственно диалог, направленный на решение выбранной задачи; если при работе в вопросно-ответном варианте пользователь выберет работу через интерпретатор запросов, инициатива передается человеку.

Заключительный диалог *(Эпилог)* начинается с директивы ЗАКОНЧИТЬ РАБОТУ и включает в себя операции по оформлению и выдаче результатов работы на оконечные устройства ЭВМ.

СИСТЕМА РАБОТУ ЗАКАНЧИВАЕТ. УКАЖИТЕ, КУДА СЛЕДУЕТ ВЫВЕСТИ РЕЗУЛЬТАТЫ РАБОТЫ:
1. СОХРАНИТЬ ДЛЯ ПОСЛЕДУЮЩЕЙ РАБОТЫ
2. НА ШИРОКУЮ ПЕЧАТЬ
3. НА МАГНИТНУЮ ЛЕНТУ
4. НА ПЕРФОЛЕНТУ
5. ВЫВОДИТЬ НЕ НАДО
ВАШ ОТВЕТ: 1

Начиная с момента включения терминального устройства и до окончания работы ряд модулей системы загружен в память машины. В моменты обдумывания пользователем своего очередного действия ресурсы машины расходуются вхолостую. Поэтому система периодически напоминает пользователю, что она ждет от него ответа. В том случае, когда пользователь долго не работает с системой, она автоматически отключается, предварительно уведомив об этом пользователя. При этом вся незавершенная работа переносится на диск и может быть вызвана с него для продолжения работы.

Выдав системе директиву на решение задачи, пользователь должен быть всегда уверен, что система поняла его и начала решать задачу. Для этого предусмотрена периодическая выдача на экран сведений о процессе решения задачи. Две перечисленные выше функции реализует *Заполняющий диалог.*

В процессе своего функционирования диалоговая система должна решать следующие задачи:

идентифицировать пользователя;

инициировать диалог и производить настройку на выбранный тип задачи;

осуществлять работу по фиксированному сценарию;

работать с каталогом стандартных запросов;

работать с произвольными запросами, сформулированными на естественном языке;

вести интерактивную работу с моделями;

осуществлять оперативные изменения в информационных массивах;

модифицировать в интерактивном режиме некоторые программы, работающие с естественным языком;

осуществлять взаимодействие с базой данных;

управлять работой модулей, входящих в систему.

Структурная схема системы ДИСПУТ, реализующая перечисленные задачи, показана на рис. 4.2. Представленные в ней модули объединены в пять основных блоков: информационную базу (ИБ), блок управления информационной базой (БУИБ), блок ведения диалога (БВД), блок интерпретации запросов (БИЗ), блок обслуживания (БО).

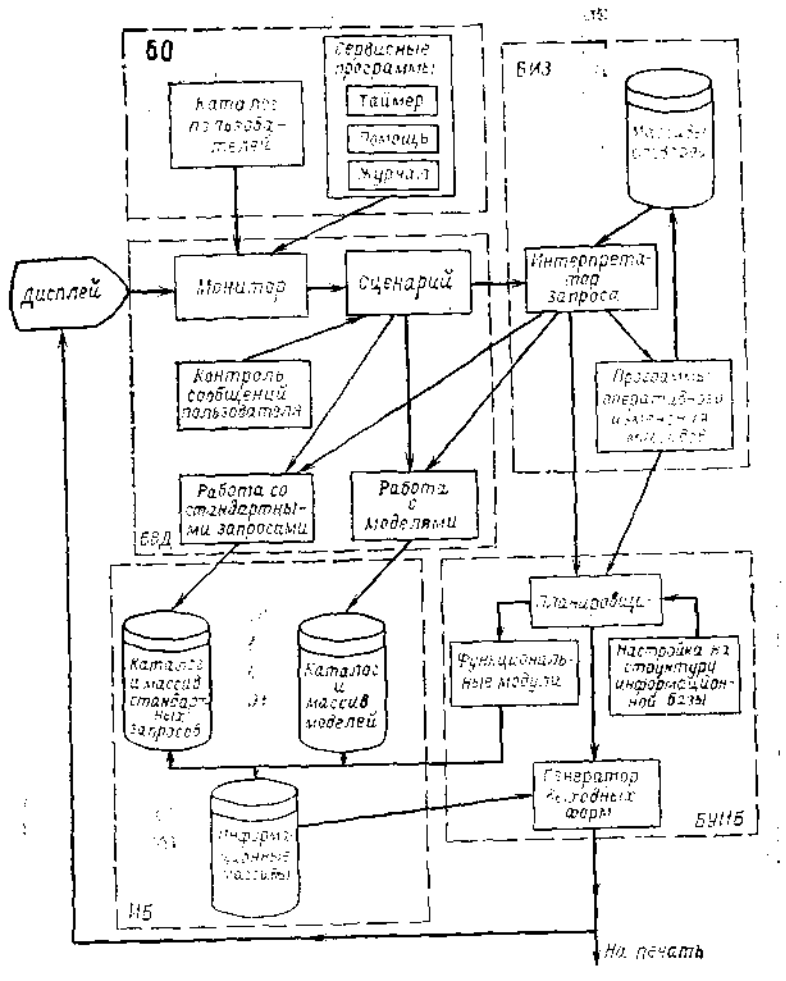


Рис. 4.2

Информационная база предназначена для хранения массивов данных, стандартных запросов и их каталога и моделей, также с каталогом. Начальное заполнение массивов осуществляется специальным комплексом программ, не входящих в систему ДИСПУТ.

Блок управления информационной базой, предназначен для планирования и выполнения поисковых, вычислительных и логических операций над данными и для формирования выходных сообщений. Работа с данными обеспечивается набором функциональных модулей, из которых собираются программы обработки. Функции сборки и выстраивания требуемой последовательности этих модулей выполняются планировщиком. Планировщик для своей работы получает задание, описанное на формальном языке, который фактически является языком манипулирования данными. Настройка на структуру информационной базы выполняется специальным модулем, входящим в блок управления. Необходимость в такой настройке возникает при переносе системы от одной группы пользователей к другой. Выходные сообщения и таблицы формируются специальным генератором и могут выдаваться на широкую печать и экран дисплея.

Блок ведения диалога предназначен для инициации диалога, выбора варианта работы с системой, обеспечения работы по выбранному варианту и выдачи задания на оформление результатов. В системе ДИСПУТ предусмотрены три варианта работы с пользователями: ответы на запросы, интерактивная работа с моделями и оперативное обновление информационных массивов. Инициация диалога осуществляется из основного сценария, с помощью которого реализуются также мониторные функции. При работе в вопросно-ответном варианте пользователю предоставлены две возможности: использовать каталог стандартных запросов или интерпретатор запросов. В каталог стандартных запросов вынесены наиболее часто встречающиеся запросы и задания на выдачу сводок и таблиц, формируемых по большому количеству данных и требующих длительных операций по обработке. Как правило, большие отчеты и сводки выводятся на широкую печать и не посылаются на дисплей из-за их плохой обозримости. При работе через интерпретатор запросов пользователь формулирует свои требования к системе на профессиональном естественном языке.

При работе с системой по второму варианту используется специальный сценарий, с помощью которого осуществляются выбор модели и загрузка ее стандартными параметрами. Изменение параметров и ограничений модели производится через интерпретатор запросов.

Третий вариант работы не является основным, поскольку заполнение и корректировка информационных массивов ведутся в пакетном режиме. Для внесения оперативных изменений на экране высвечивается форма, которая заполняется пользователем. После проведения серии проверок оперативные сведения заносятся в базу данных.

Блок интерпретации запросов служит для перевода запросов с естественного языка в фактические параметры планировщика.

Блок обслуживания содержит каталог пользователей и сервисный модуль. Последний в свою очередь включает в себя программы *Помощь, Журнал* и *Таймер.*

Каталог пользователей служит для защиты системы от несанкционированного доступа. Он содержит таблицы, где каждому пользователю поставлена в соответствие следующая информация: фамилия, имя, отчество, пароль, кодовый номер, перечень доступных массивов и перечень доступных директив. Начиная работать с системой, пользователь вводит с клавиатуры три параметра: ФИО, пароль и кодовый номер. Программа идентификации пользователя сравнивает введенные параметры с хранимыми в каталоге и в зависимости от результатов анализа разрешает или запрещает дальнейшую работу с системой. Пароль служит для защиты системы от неавторизованного доступа. Он является переменной величиной и периодически обновляется. Кодовый номер является условно-постоянной величиной. На его основе определяются приоритеты работы с системой, а также виды работ, доступные данному пользователю.

Программа *Помощь* реализует консультативный диалог и вызывается пользователем по команде «?». Она помогает новичкам быстро освоиться с системой, а постоянным пользователям напоминает по их требованию правила работы с системой на любом из этапов диалога. Массив этой программы содержит информацию, относящуюся к каждому экранному формату при работе по сценариям, и сведения, помогающие при работе с моделями или в процессе оперативного обновления массивов.

Программа *Журнал* предназначена для ведения протокола диалога. При этом преследуются три цели:

запись промежуточных результатов работы системы для выявления возможных причин ошибочной работы;

ведение статистики обращений к модулям и каталогизированным запросам и программам для последующей модернизации системы;

протоколирование всех действий пользователя.

Программа *Таймер* реализует заполняющий диалог, о котором говорилось выше.

Основным элементом, выделяющим систему ДИСПУТ из ряда традиционных диалоговых систем, является интерпретатор естественноязыковых запросов. Как уже говорилось выше, он включает в себя лингвистический и прагматический процессоры. Их описанию мы предпошлем рассмотрение базы данных системы, которая включает информационные массивы и блок управления.

4.2. База данных

В информационных массивах системы хранятся сведения о контейнерах, транспортных средствах, людских и технических ресурсах контейнерного терминала морского порта. Контейнерный терминал является по существу складом и перевалочной базой для контейнеров, прибывающих и убывающих морским, железнодорожным и автомобильным транспортом. Информационные массивы формируются на основе обширной и разнородной информации, поступающей от различных организаций министерств, вовлеченных в транспортный процесс, от капитанов судов, от грузоотправителей и грузополучателей, информации, содержащейся в документах, сопровождающих контейнеры, из разнарядок и поручений и т. д.

С системой могут работать пользователи нескольких уровней. Чем выше уровень пользователя (уровень руководящего состава), тем большее количество данных вовлекается в работу для получения ответа на запрос. При этом требования к срочности ответа, как правило, снижаются. Наоборот, на самом нижнем уровне (уровне диспетчеров контейнерного терминала) объем рабочей информации ограничен, но ответы на запросы должны выдаваться немедленно.

В связи с такими требованиями к срочности выдачи информации на нижнем уровне была произведена разбивка всего объема данных на ряд массивов. Массивы построены по последовательному и индексно-последовательному принципу. Применительно к сведениям о контейнерах сформированы следующие массивы:

1) массив контейнеров, прибывающих морем (прибывающие импортные);

2) массив контейнеров, прибывающих по железной дороге и автотранспортом (прибывающие экспортные);

3) три массива контейнеров, находящихся на контейнерном терминале:

импортные контейнеры;

экспортные контейнеры;

порожние и контейнеры в ремонте;

4) массив контейнеров, убывших морем (экспортные убывшие);

5) массив контейнеров, убывших по железной дороге и автотранспортом (импортные убывшие);

6) массив состояния контейнерного терминала с информацией о местоположении контейнеров, их номерах, видах перевозки, направлении перевозки и их техническом состоянии;

7) архивный массив;

8) массив с нарастающими итогами по работам на контейнерном терминале.

В результате такого разбиения на массивы поиск ответа на запросы пользователей нижних уровней удалось свести к поиску в одном-двух массивах. При среднем объеме массивов 800—1200 записей время поиска в них оказалось приемлемым для оперативного взаимодействия пользователя с системой.

Каждый объект (в частности, описание контейнера) характеризуется рядом параметров. Так, описание контейнера содержит более ста параметров, среди которых: шифр контейнера, имя судна, доставившего его в данный порт, сведения о состоянии контейнера, коммерческие сведения, описание груза, сведения о позиции контейнера на терминале, информация о дате прибытия и времени хранения на терминале и т. д. Размерность каждой записи до 150 байт.

Таблица 4.1

P1=1 Массивы прибывающих контейнеров	*P1*=2 Массивы контейнеров на терминале	*P1*=3 Массивы отправленных контейнеров	*P1*=4 Массив с нарастающим итогом	*P1*=5 Массив состояния контейнерного терминала
P2=1 Экспортные контейнеры	*P2*=1 Экспортные контейнеры	*P2*=1 Экспортные контейнеры	*P2*=1 Экспортные контейнеры	*P2*=1 Экспортные контейнеры
P2=2 Импортные контейнеры	*P2*=2 Импортные контейнеры	*P2*=2 Импортные контейнеры	*P2*=2 Импортные контейнеры	*P2*=2 Импортные контейнеры
—	*P2*=3 Дефектные контейнеры	*P2*=3 Архивные данные	*P2*=3 Дефектные контейнеры	*P2*=3 Дефектные контейнеры

В табл. 4.1 представлены перечисленные выше массивы вместе с их обозначениями, принятыми в системе.

Функциональные модули служат для реализации процедур информационного поиска, выполнения вычислительных и логических операций.

Информационный поиск осуществляется процедурами двух видов: процедурой поиска по ключу и процедурой поиска по заданному значению признака. Первая процедура выполняется при задании идентификатора объекта поиска и списка имен признаков, подлежащих выдаче. Ответом на такой запрос является список значений признаков, соответствующих заданному идентификатору.

Вторая процедура служит для отыскания списка объектов, обладающих признаком с заданным значением.

Прямой поиск по ключу просто реализуется в индексно-последовательном массиве, если ключ поиска совпадает с ключом упорядочения массива. Для осуществления поиска по признакам прибегают к искусственным приемам, например создают инверсные массивы. Это приводит к дублированию информации, увеличению числа носителей информации, усложнению работы по обновлению массивов и т. д.

В системе ДИСПУТ, как уже было сказано выше, присутствуют только последовательные и индексно-последовательные массивы. Экспериментально показано, что при выбранных объемах массивов поиск по признаку с простым перебором всех записей массива происходит за приемлемое время. Поэтому в системе инверсные массивы не были организованы.

Система содержит несколько функциональных модулей, осуществляющих вычислительные и логические операции:

1) модули подсчета числа объектов;

2) модуль суммирования значений нескольких заданных признаков;

3) модуль вычисления временных характеристик.

Рассмотрим более подробно первый модуль на примерах работы с массивами записей о контейнерах.

Например, для запроса: *Сколько на контейнерном терминале экспортных контейнеров, в т. ч. по странам назначения (в т. ч. по судам транспортировки (в т. ч. исправных))* необходимо:

1) произвести подсчет числа экспортных контейнеров по каждой стране назначения отдельно;

2) для каждой страны назначения подсчитать число контейнеров по каждому судну транспортировки;

3) для каждого судна найти число исправных контейнеров.

С помощью признакового поиска из информационных массивов выбираются записи с запрашиваемыми признаками. Отсортировав массив по возрастанию значений всех признаков, получим: последовательность записей, упорядоченных по возрастанию шифров стран назначения; записи с одинаковыми шифрами стран и судов, упорядоченные по значению технического состояния контейнера; записи с одинаковыми шифрами стран, упорядоченные по возрастанию шифров судов.

Такое расположение записей позволяет иметь в памяти машины только три рабочих поля: для накапливания числа контейнеров, перевозимых в каждую страну в отдельности, в страну на каждом судне в отдельности, в страну на судне при каждом техническом состоянии контейнера в отдельности. Смена значения признака при последовательном просмотре отсортированных записей означает, что в счетчике накопилось значение, соответствующее числу контейнеров, и это число можно выводить на дисплей или печать. В том же рабочем поле теперь можно накапливать следующее значение, соответствующее числу контейнеров, для нового значения признака.

Такой способ накапливания числа контейнеров с однократной сортировкой возможен только в том случае, когда запрашиваемые признаки имеют разные уровни вложенности и нет признаков с одним и тем же уровнем. В приведенном выше примере уровни вложенности распределяются следующим образом: шифр страны имеет нулевой уровень, шифр судна — первый уровень, шифр технического состояния — второй уровень.

Если в запросе содержится несколько признаков с одним уровнем вложенности, то для получения числа контейнеров необходима многократная сортировка. Для ответа на запрос: *Сколько на терминале экспортных контейнеров (в т. ч. по странам назначения (в т. ч. по судам транспортировки (в т. ч. исправных, по типоразмерам, по грузам в контейнерах)))* потребуется трижды сортировать отобранный массив записей по трем ключам:

первая сортировка — по возрастанию шифров страны, судна, технического состояния;

вторая — по возрастанию шифров страны, судна, типоразмеров контейнера;

третья — по возрастанию шифров страны, судна, шифра груза в контейнере.

Для этого запроса уровни вложенности имеют следующие значения: шифр страны — нулевой уровень, шифр судна — первый уровень, шифр технического состояния — второй уровень, шифр типоразмера — второй уровень, шифр груза — второй уровень.

Время, затрачиваемое на три сортировки массива и три последовательных просмотра отсортированных массивов для получения значения, равного числу контейнеров, достаточно велико. Если позволяет объем оперативной памяти и ЭВМ обладает хорошим быстродействием, более эффективным является другой способ определения числа контейнеров. Используя этот способ, можно работать при любом наборе уровней вложенности признаков и при любом числе признаков с одинаковым уровнем. Способ заключается в одновременном накапливании в памяти машины числа контейнеров при всех различных значениях запрашиваемых признаков. Число рабочих полей для получения числа контейнеров зависит от числа признаков «в том числе» и множества значений запрашиваемых признаков.

Просматривая последовательно массив отобранных записей, анализируя значения признаков в них, можно каждый раз определить, в какие из рабочих полей с информацией о числе контейнеров надо добавлять единицу. С окончанием просмотра массива заканчивается подсчет числа контейнеров по всем значениям признаков. Результаты просмотра теперь могут быть выданы на внешние устройства.

В математическом обеспечении системы ДИСПУТ присутствуют оба способа вычисления функции «Число контейнеров». *Это позволяет использовать лучший из них* в каждом конкретном случае эксплуатации системы.

Планировщик базы данных выполняет функции распределения обязанностей между модулями управления базой данных. В результате работы интерпретатора запросов в блок управления информационными массивами передаются фактические параметры для планировщика. В обязанности планировщика входит:

1) генерирование работы с реальными массивами данных;

2) создание рабочих таблиц: поисковых предписаний и их значений, ключей сортировки, признаков «в том числе» при поиске ответа на вопросы типа «сколько», сумм;

3) определение набора модулей, обеспечивающих ответ на заданный вопрос, и управление последовательностью их выполнения.

Таблица описания массивов для каждого типа и для каждого признака записи содержит: шифр показателя, адрес показателя в записи, длину показателя и формат его представления.

Справочная таблица информационных массивов содержит: шифр информационного массива, адрес таблицы описания массива, метку массива на внешнем носителе.

Последовательность функциональных модулей, реализующая требуемую функцию работы с массивами, имеет древовидную структуру. В рассматриваемой версии системы ДИСПУТ выбирается любая первая ветка из множества допустимых на этой структуре.

Функциональные модули блока управления обеспечивают: сортировку рабочего массива по созданной таблице признаков, подсчет числа объектов, вычисление сумм значений заданных признаков, выборку и подготовку к печати сумм для ответа на запрос.

Генератор выходных форм обеспечивает: формирование и печать заголовка ответа на запрос, формирование и печать строки ответа на запрос, формирование экранных сообщений, организацию работы с экраном. Генератор выходных форм работает со справочным массивом признаков, который используется для формирования словесного ответа на запрос. Это индексно-последовательный массив, где ключом является объединение шифра показателя и шифра значения показателя. Каждый показатель в этом массиве содержит: шифр показателя, шифр значения показателя, название значения показателя или название показателя, общее для всех значений, формат для табличной выдачи — длину поля, число строк названия в таблице, построчное название показателя.

С помощью генератора выходных форм можно конструировать таблицы и сводки для выдачи их на печать. При этом последовательность расположения показателей в таблице может задаваться порядком их следования в запросе.

4.3. Лингвистический процессор

Выше уже говорилось, что основная задача лингвистического процессора заключается в переводе исходного запроса во внутреннее формальное его представление.

Основанием для разработки лингвистического процессора служат: структура информационных массивов базы данных и перечень возможных запросов пользователей к системе. Предлагаемый способ анализа запросов, кото-

рый выполняется в лингвистическом процессоре, пригоден для информационных массивов, построенных по принципу «признак — значение», т. е. каждая запись, содержащая описание объекта, характеризуется набором признаков, принимающих значения из заданного множества.

Набор потенциальных запросов пользователя служит для формирования словарей системы и выявления возможных конструкций предложений. Приведем несколько примеров запросов к системе, взятых из реальных ситуаций. Наиболее многочисленна группа запросов о числе контейнеров: «Сколько контейнеров прибывает на терминал на судне «Светлогорск», в том числе по номерам?», «Сколько импортных контейнеров на терминале, подлежащих отгрузке на автотранспорт, на железнодорожный транспорт, на речной транспорт, в том числе по номерам?», «Количество дефектных контейнеров на терминале» и т. п. Примерно такова же по объему группа запросов о временных характеристиках: «Сколько времени с момента поступления на терминал хранятся экспортные контейнеры, в том числе по объединениям Министерства внешней торговли, по странам назначения, не имеющие поручений на отгрузку, в том числе по номерам? «Время хранения порожних контейнеров по типоразмерам (в том числе дефектных и по номерам)», «Сколько контейнеро-суток хранятся импортные контейнеры по грузополучателям?». Существуют запросы, требующие перечислить контейнеры, обладающие рядом фиксированных параметров, и запросы, которые относятся к каким-либо характеристикам одного отдельно взятого контейнера, например: «Перечислить контейнеры, которые будут отправляться в страну «Италию» до 20 : 05 : 80 года». «Для контейнера МММУ000427, подлежащего отправке железной дорогой в город «Москву», назвать содержащийся в нем груз». «Вид перевозки и станция назначения контейнера ВДХ2006870».

Примеры показывают, что пользователь довольно свободно может формулировать свои запросы, оставаясь в рамках синтаксических конструкций, естественных для делового языка запросов. Тем не менее система накладывает на пользователя некоторые ограничения. Во-первых, это небогатая лексика, которая в рассматриваемой версии составляет около 300 слов. Опыт показывает, что словарь лексики расширяется медленно и для данной проблемной области близок к насыщению. Во-вторых, задание вложенных процедур сортировки, упорядочения и подсчета должно производиться соответствующей расстановкой скобок. Примеры такого типа запросов приводились в § 4.2. В-третьих, в запросах не допускается постановка точки в любом ином месте, кроме конца предложения. Запятые важны только в том случае, если их отсутствие вносит неоднозначность в толкование предложения. В-четвертых, имена собственные и имена грузов обрамляются кавычками. Это сделано для того, чтобы не утяжелять словарь и не терять в эффективности системы.

При вводе запроса допустимы сокращения слов и некоторые их искажения, приходящиеся на конец слова. Специальный блок морфологического анализа в рассматриваемой версии системы отсутствует.

В лингвистическом процессоре используется синтактико-семантическое кодирование слов исходного предложения с его последующим грамматическим анализом. Лингвистический процессор включает в себя (рис. 4.3): словарь словоформ, кодировщик, редактор, грамматический анализатор, расширитель словаря, семантическую сеть и обучатель (программу «обучения» процессора).

Для работы с текстами на естественном языке система снабжена общими и специальными «знаниями». Общие знания хранятся в семантической сети в виде ролевых фреймов. Они используются для выявления темы диалога о контейнерах, терминале, грузоотправителях, портах и т. п. Специальные знания служат для анализа предло-

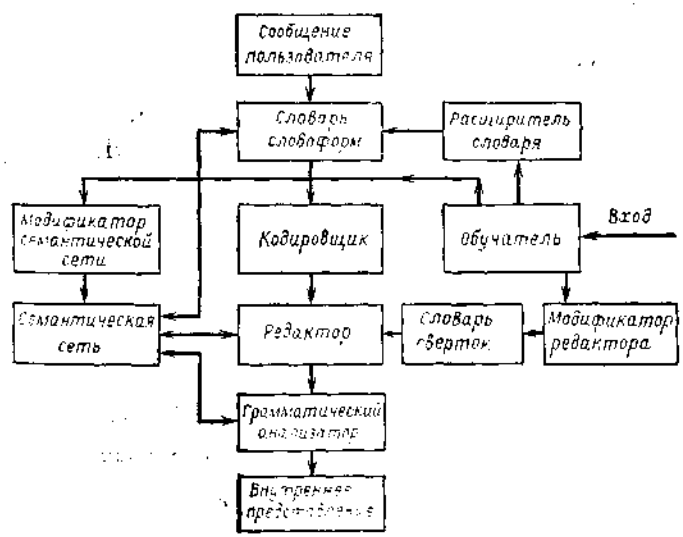


Рис. 4.3

жений в рамках выбранной тематики диалога. Эти знания включают декларативную и процедурную составляющие. Декларативная часть знаний хранится в словарях, а процедурная — в грамматическом анализаторе.

Лингвистический процессор работает в следующей последовательности. Вводимый запрос пословно сопоставляется со словарем, и каждой входной словоформе ставится в соответствие ее синтактико-семантический код. Коды несут синтаксическую и семантическую нагрузку, о них будет сказано ниже при рассмотрении словаря. Отсутствие слова в словаре ведет к приостановке работы системы и выдаче соответствующего сообщения пользователю. Он может либо переформулировать запрос, либо добавить в словарь лексики неизвестное системе слово, если такое добавление не затрагивает существенным образом грамматики системы.

По мере нахождения в словаре слов, составляющих запрос, производятся обращение к семантической сети и заполнение слотов ролевых фреймов. При этом активируются соответствующие компоненты редактора и настраивается схема анализа грамматики.

Закодированный запрос обрабатывается редактором, где он освобождается от избыточной информации. Основной анализ происходит в грамматическом анализаторе, который переводит сообщение во внутреннее представление. Внутреннее представление задается следующей формой:

<внутреннее представление>::=<тип запроса> <описание>

<тип запроса>::=ВОПРОС О ЧИСЛЕ ОБЪЕКТОВ|ВОПРОС О ВРЕМЕНИ|ВОПРОС ОБ ОБЪЕКТЕ|ТРЕБОВАНИЕ ПЕРЕЧИСЛИТЬ...

<описание>::=<описание 1>|<описание 2>|<описание 3>

<описание 1>::={АТРИБУТ[<имя атрибута>]} {ГЛАГОЛ [{<имя модельности>} {имя глагола>}{<имя обстоятельства или признака>} {<номер>}]} {В ТОМ ЧИСЛЕ [<элемент ВТЧ>]} {[<примечание>]}

<описание 2>::=<описание 1> {С МОМЕНТА [<имя глагола>]}

<описание 3>::=<номер> <примечание> ВОПРОС ОБ ОБЪЕКТЕ <номер> ПРИЗНАКИ <имя признака> {ГЛАГОЛ [{<имя модальности>} {<имя глагола>} {<имя обстоятельства или признака>} {<номер>}]}

<элемент ВТЧ>::=<имя атрибута>|<имя признака>|<имя обстоятельства>

<примечание>::=ПРИМЕЧАНИЕ: [<номер] ОЗНАЧАЕТ [<имя собственное>].

Квадратные скобки указывают на возможность многократного повторения того, что заключено в них. Фигурные скобки заключают в себе факультативное описание.

Перевод во внутреннее представление заканчивает работу лингвистического процессора. На каждом из этапов анализа система может встретиться с конструкциями, не предусмотренными при ее начальном заполнении. В этих случаях анализ прекращается и организуется диалог с пользователем по фиксированному сценарию для выяснения возможности дальнейшего продолжения анализа. Однако в большинстве случаев проще бывает переформулировать запрос. Конечный пользователь лишен возможности пользоваться средствами «обучения» системы. Такая привилегия предоставляется только специалисту, ведущему систему.

Система может обрабатывать лишь одно отдельно взятое предложение. Эллипсисы и анафорические отсылки не обрабатываются. В системе также не содержится механизм, который позволял бы восстанавливать смысл незнакомых слов или интерпретировать неизвестные ей конструкции на основе анализа контекста.

Рассмотрим более подробно работу основных модулей лингвистического процессора.

Словарь лексики состоит из массива словарных статей. Каждая статья (табл. 4.2) содержит три зоны: зону связи с семантической сетью, зону словоформы и зону синтактико-семантического кода.

В зоне связи с семантической сетью хранится адрес соответствующего понятия в сети. Прямое обращение из словаря в семантическую сеть служит для быстрого фор-

Таблица 4.2

Адрес в семантической сети	Синтактико-семантический код	Словоформа
C06	602	Пое * зд
A74	704	Грузоп * олучатель
A62	802	Эксп * ортный
...	...	...

мирования догадки о теме диалога. Слова исходного запроса возбуждают соответствующие ситуационные фреймы и образуют в совокупности зону внимания в семантической сети. Догадки подтверждаются или опровергаются на последующих этапах анализа при прохождении запроса через редактор и грамматический анализатор.

Приписывание синтактико-семантических кодов словоформам производится в соответствии с грамматикой, которая определяется проблемной областью и прагматикой задачи. В некоторых случаях она отличается от традиционной русской грамматики. Грамматика системы содержит десять основных лексических классов:

1) атрибуты, например *импортный, порожний, с грузом;*

2) простые глаголы, например; *затарить, ремонт;*

3) глаголы, требующие после себя обстоятельства, например *выгружать, находиться, направление, прием;*

4) модальности, например *требовать, осталось, подлежать;*

5) обстоятельства места, например *терминал, судно, поезд;*

6) обстоятельства времени, например *момент, дата;*

7) признаки, например *порт, типоразмер, страна, груз, вес;*

8) управляющие слова, опорные — *контейнер, номер, время* и т. д.; служебные — *сколько, какой, как* и т. п.;

9) вспомогательные слова, например *вид, всего, упаковка;*

10) предлоги, например *на, в, с.*

Из приведенных примеров видно, что используемая грамматика имеет отличия от традиционной. Так, слова *ремонт* и *прием* отнесены к глагольным формам, а слово *время* отнесено к опорным словам, и т. д. Однако подчеркнем, что в рамках выбранной прагматики такая классификация слов является истинной.

Словоформы в словаре разбиты на гнезда, соответствующие буквам алфавита. В пределах гнезд существует строгая упорядоченность слов по алфавиту. Сами гнезда по алфавиту могут не упорядочиваться. Новое гнездо формируется по мере поступления из запросов словоформ, начинающихся с ранее не встречавшейся в словаре буквы.

Сопоставление очередного слова запроса с массивом словаря производится, начиная с первого гнезда. Сопоставление по первой букве ведется до тех пор, пока не будет найдено нужное гнездо. В пределах одного гнезда поиск продолжается до максимального (по числу букв) совпадения словоформы запроса и словоформы словаря. В словаре хранятся лишь попарно различимые последовательности букв, составляющие начало каждого слова. Как уже говорилось ранее, в данной версии системы отсутствует блок морфологического анализа. Однако ограниченный словарь позволяет пока обходиться без блока, который может замедлить анализ и потребовать дополнительных ресурсов памяти.

Кодировщик служит для начальной обработки сообщений пользователя. Обработка включает в себя:

1) вычленение законченного предложения;
2) нахождение вложенных конструкций;
3) нахождение имен собственных;
4) нахождение числовых констант;
5) анализ знаков препинания;
6) кодирование словоформ.

Конец предложения определяется по точке или знаку вопроса. Поэтому любые сокращения при формулировке запроса нельзя вводить с расстановкой точек. Например, сокращение *в т. ч.* вместо полной формы *в том числе* недопустимо. Допустимой формой было бы *в тч* или *втч.*

Пословное кодирование предложения начинается с побуквенного сопоставления первого слова текста со словарем. Если сравниваемое слово будет найдено в словаре, то его код считывается в результирующий код предложения.

Запросы, включающие вложенные конструкции, которые перечисляют или уточняют компоненты поиска, встречаются довольно часто. Вложенные конструкции начинаются либо с открывающих скобок, либо с выражения *в том числе*. При кодировании началу и концу вложенной конструкции присваивается определенный код, имеющий статус слова, который входит в результирующий код предложения.

Имена собственные находятся по обрамляющим их кавычкам. При анализе предложения кодировщик помещает содержимое между кавычками в специальный массив текстовых констант, а в общий код предложения записывается формируемый уникальный код имени собственного.

Аналогичным способом обрабатываются числа или смешанные буквенно-числовые конструкции, образующие слитную словоформу. В отличие от имен собственных числа в кавычки могут не заключаться. Конструкции, содержащей числа, присваивается вычисляемый код, а сама конструкция выносится в специальный массив для дальнейшей обработки.

В раннем варианте системы полностью игнорировались все запятые во вводимом запросе. Такое решение было принято в связи с тем, что в условиях реальной работы трудно рассчитывать на правильную расстановку знаков препинания. Однако опытная проверка системы показала, что в некоторых случаях запятые существенно меняют смысл запроса. Например, контейнеры могут быть экспортными, могут быть транзитными. Но есть категория контейнеров, которые относятся к транзитным экспортным. В этом случае игнорирование запятых между однородными членами могло бы привести к неправильной интерпретации запроса. Поэтому в описываемой версии системы запятые кодируются и сохраняются до того момента, пока не будет снята неоднозначность в толковании запроса. Как правило, это происходит в редакторе.

В результате работы кодировщика создаются код предложения и массив констант, куда входят имена собственные и числа.

Редактор служит для уменьшения избыточности запроса перед его анализом в грамматике. В нем используется массив декларативных знаний об устойчивых лексических конструкциях, которые имеют фиксированную семантическую интерпретацию в рамках данной системы. На этапе редактирования уточняется также первоначальная догадка о теме диалога.

Найденные в массиве знаний лексические конструкции подтверждают или опровергают гипотезу о теме. В соответствии с этим производится увеличение или снижение весов в первоначальной «зоне внимания» семантической сети. В случае снижения весов связей и возможной переориентации «зоны внимания» производится повторное обращение к массиву знаний редактора.

Таблица 4.3

Имя темы	Номер правила	Код первой словоформы	Код второй словоформы	Индикатор места	Результирующий код
Все	76	4 (*какое*)	2 (*колич*)	+1	2 (*сколько*)
Конт	23	600	100 (*на*)	—1	650
терм		(*терминал*)			(*на-терминале*)
	24	650	0	0	0 (*удалить из текста*)
		(*на-терм*)			
Все	28	33	34 (*том*)	—1	30 (*том-числе*)
		(*число*)			
Все	6	30 (*том-числе*)	103 (*в*)	—1	30 (*в-том-числе*)
...	...	...	...	...	...

Массив декларативных знаний редактора задан в виде правил склеивания кодов словоформ. В свою очередь правила склеивания представлены в системе в виде таблицы (табл. 4.3).

В этой таблице в графе «Индикатор места» стоит число, определяющее место второй словоформы при анализе предложения. Это число равно: +1, если словоформа справа; —1, если словоформа слева; 0, если надо опустить словоформу; —2, если словоформа слева и ее надо сохранить в результирующем коде.

Правило работы с кодами предусматривает:

1) попарную склейку двух кодов с присвоением результирующей лексической конструкции кода одной из анализируемых словоформ или нового кода;

2) выбрасывание из дальнейшего анализа одного из кодов словоформ;

3) циклическое свертывание кодов словоформ, если исходная лексическая конструкция включает более двух словоформ.

Идентификация пары кодов, подлежащих склейке, может начинаться с первой или со второй по порядку словоформы, что указывается в правиле склейки. Во втором случае первый по порядку код может быть уже результатом склейки. Этот прием используется для циклического свертывания кодов.

Результатом работы редактора является код запроса, в котором сохраняются только те коды словоформ и коды лексических конструкций, которые необходимы для однозначного понимания запроса в рамках выбранной прагматики.

Грамматический анализатор производит основной анализ предложения. Перед описанием этого модуля заметим, что основная масса запросов объединяется темой «Контейнер». Поэтому работа грамматики будет иллюстрироваться этой темой.

Грамматический анализ заключается в распознавании и выделении групп словоформ, составляющих в совокупности все предложение, в нахождении основного отношения, высказанного в запросе, а также групп слов, описывающих объекты, которые входят в это отношение. Содержащиеся в грамматическом анализаторе знания заданы в операционной форме.

Если в процессе анализа выяснится, что при выбранной теме нельзя произвести разбор предложения, то анализ возвращается на шаг назад и из семантической сети выбирается альтернативный вариант темы со следую-

щим по значению весом. Вновь включается в работу редактор и т. д.

Применительно к теме «Контейнер» удобно выделить четыре типа запросов: вопросы о числе контейнеров, вопросы о времени, вопросы о самом контейнере и его характеристиках, требование перечислить контейнеры с заданным набором признаков.

Подчеркнем, что выбор такого числа типов запросов обусловлен удобством их интерпретации пользователем, а не возможностями системы. Это число может меняться и менялось в зависимости от потребностей и вкусов пользователей.

В результате анализа необходимо выяснить: 1) тип запроса; 2) список атрибутов; 3) список глагольных форм; 4) список основных признаков; 5) список дополнительных признаков; 6) уровни вложенности дополнительных признаков.

Перечисленные параметры вместе с массивами имен собственных и числовых констант составляют внутреннее представление запроса на выходе из лингвистического процессора.

Запрос по теме «Контейнер» в структурном виде показан на рис. 4.4. Грамматический анализатор после начального выбора темы в кодировщике и ее уточнения в редакторе собирается из соответствующих данной теме

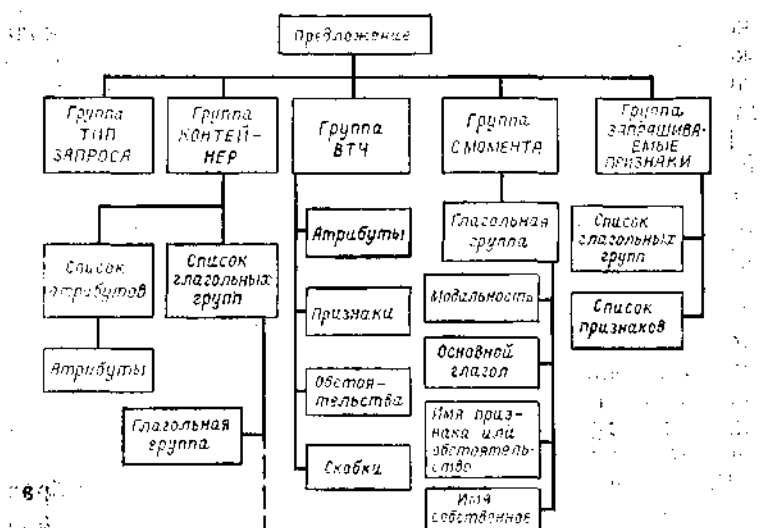


Рис. 4.4

блоков. В рассматриваемом случае она включает в себя головной блок, ведающий идентификацией начала групп, и блоки разбора этих групп.

Атрибуты, как правило, представлены прилагательными русского языка. Глагольная группа состоит из: модального глагола или его эквивалента, основного глагола, дополнения или имени признака, имени собственного. Дополнение обычно выражено именем существительным винительным с предлогом или без него. Все составляющие глагольной формы могут присутствовать в ней факультативно.

Список элементов группы *в том числе* состоит из атрибутов, признаков, обстоятельств места и указателей вложенности. Обстоятельства места определяют сортировку массивов.

Уместно отметить, что семантически атрибуты задают одновременно имя некоторого признака и его значение. Например, слово *импортный* задает имя признака ВИД ПЕРЕВОЗОК и его значение ИМПОРТ. Слова, отнесенные к категории «признак», задают только имя признака, но не его значение (например, *грузовладелец* ГРУЗОВЛАДЕЛЕЦ=?). Слова, отнесенные к категории «обстоятельство места», аналогичны атрибутам (например, *поезд*→ВИД ТРАНСПОРТА=ЖД), но могут одновременно служить именем другого признака, значение которого не задано или задается другим словом (например, *поезд*→ПОЕЗД=НОМЕР ПОЕЗДА).

Еще два типа слитных фрагментов образуют группы С МОМЕНТА и группу ЗАПРАШИВАЕМЫЕ ПРИЗНАКИ.

В используемом алгоритме анализа предполагается, что та часть описания группы объектов, которая соответствует списку атрибутов и списку глагольных форм, должна образовывать вместе с наименованием объекта сплошной фрагмент текста. При этом атрибуты могут стоять как до, так и после имени объекта. Каждая глагольная группа образует слитный фрагмент текста. Глагольные группы могут стоять по обе стороны от имени объекта. Они могут следовать одна после другой или перемежаться атрибутами.

Часть описания, которая соответствует группе «в том числе», должна образовывать слитный фрагмент текста.

Головной блок узнает начало группы по соответствующему ключевому слову, с которого эта группа может начинаться. Например, для группы КОНТЕЙНЕР это слово — *контейнер* или любой атрибут: *импортный, по-*

рожний и т. п. Для группы ВТЧ (*в том числе*) — это словосочетание *в том числе* или открывающая скобка. Для группы С МОМЕНТА — это словосочетание *с момента* и т. д.

После идентификации начала соответствующей группы головной блок передает управление блоку разбора этой группы и по окончании работы последнего переходит к идентификации очередной группы. Некоторые слова типа *сколько, время, шифр* и т. п., рассматриваются как особые группы. Кроме того, головной блок фиксирует конец разбора по концу предложения.

Головной блок опознает тип запроса. По умолчанию принимается, что запрос по теме «Контейнер» относится к четвертому типу (требование перечислить). Например, предложение: *Экспортные контейнеры* означает, что необходимо дать перечень экспортных контейнеров по номерам. Последнее обстоятельство тоже задается по умолчанию. Слова типа *сколько*, а также наличие группы контейнера означают, что вопрос может относиться к типу один или два. Наличие слов *время* или группы С МОМЕНТА означает, что запрос относится к типу два. Запросы типа три идентифицируются по присутствию в них группы ЗАПРАШИВАЕМЫЕ ПРИЗНАКИ.

При разборе групп соответствующими блоками заполняются таблицы атрибутов, глагольных форм и признаков. Каждая группа высшего уровня должна входить в предложение не более одного раза.

Блок группы КОНТЕЙНЕР функционирует по следующему алгоритму. Если группа начинается с атрибута, то словоформы (а вернее, коды после редактора) перебираются подряд до имени объекта — слова *контейнер*. Все они должны быть атрибутами. В противном случае разбор прекращается с указанием на ошибку. Атрибуты заносятся в список атрибутов. Само слово, обозначающее объект (например, *контейнер*), при этом опускается, так как в этом контексте оно не несет никакой семантической нагрузки.

Если следующее слово после слова *контейнер* есть атрибут, то оно заносится в список атрибутов. Если же очередное слово — глагол, то начинается разбор глагольной группы, по окончании которого анализируется следующее по очереди слово. Если это слово не глагол и не атрибут, то разбор группы объекта заканчивается.

Блок глагольных форм формирует список, состоящий из четырех элементов: модальности, основного глагола, обстоятельства или признака и имени собственного.

Разбор глагольной группы начинается со слова, обязательно являющегося глаголом. Если это модальный глагол или его субститут, то код глагола фиксируется как первый элемент списка (МОДАЛЬНОСТЬ). Следующим словом также обязательно должен быть глагол, но не модальный. Этот глагол заносится во вторую позицию списка (ГЛАГОЛ). Если найденный глагол входит в число «простых», то разбор группы завершается. Если же он требует после себя определения, то производится поиск относящегося к нему обстоятельства (или обстоятельства с предлогом). Если проверка подтвердит факт наличия обстоятельства, то оно заносится в третью позицию списка. Место обстоятельства может занимать также признак. Оно также заносится в третью позицию (ОБСТОЯТЕЛЬСТВО ИЛИ ПРИЗНАК). За обстоятельством или признаком может идти имя собственное, которое помещается в четвертую позицию (ИМЯ). На этом разбор глагольной группы заканчивается. Разбор оканчивается также и на промежуточных этапах, когда за обязательным компонентом списка не следуют факультативные элементы.

Разбор группы ВТЧ начинается со словосочетания *в том числе* или открывающей скобки. Коды этих лексических единиц не используются далее и служат только для идентификации начала группы. Коды слов, входящие в группу ВТЧ, последовательно заносятся в список ВТЧ. Кандидатами для занесения в этот список могут быть лишь имена признаков или обстоятельства, атрибуты, словосочетание *по номерам*, скобка открывающая или скобка закрывающая. Имя признака или обстоятельство при этом могут быть с предшествующим предлогом *по* и словами типа *шифр, номер, наименование* и т. п., которые опускаются при анализе. Повторное использование словосочетания *в том числе* считается относящимся к первому его употреблению и поэтому при анализе игнорируется. Любое иное слово, кроме перечисленных выше, служит признаком окончания группы.

Каждая открывающая скобка содержательно означает переход на более глубокий уровень поиска, а каждая закрывающая — возвращение к предыдущему.

Блоки групп С МОМЕНТА и ЗАПРАШИВАЕМЫЕ ПРИЗНАКИ работают аналогично блоку глагольных форм. Разница заключается в том, что словосочетание *с момента* и выделенный набор признаков служат для идентификации этих групп и установления типа запроса.

4.4. Прагматический процессор

Прагматический процессор предназначен для перевода результатов грамматического анализа исходного запроса в задание планировщику базы данных.

Задание планировщику оформляется в виде последовательности параметров, которая включает в себя: 1) код функции, выполняемой над массивами данных; 2) коды массива и подмассива; 3) список основных признаков; 4) список дополнительных признаков; 5) список запрашиваемых признаков; 6) указатель момента времени события.

Функциональная схема прагматического процессора представлена на рис. 4.5. Код функции, выполняемой над массивами базы данных, однозначно определяется по типу запроса на основании таблицы соответствия «тип запроса — код функции».

Список основных признаков состоит из ключа, кода признака и значения признака. Ключ может принимать

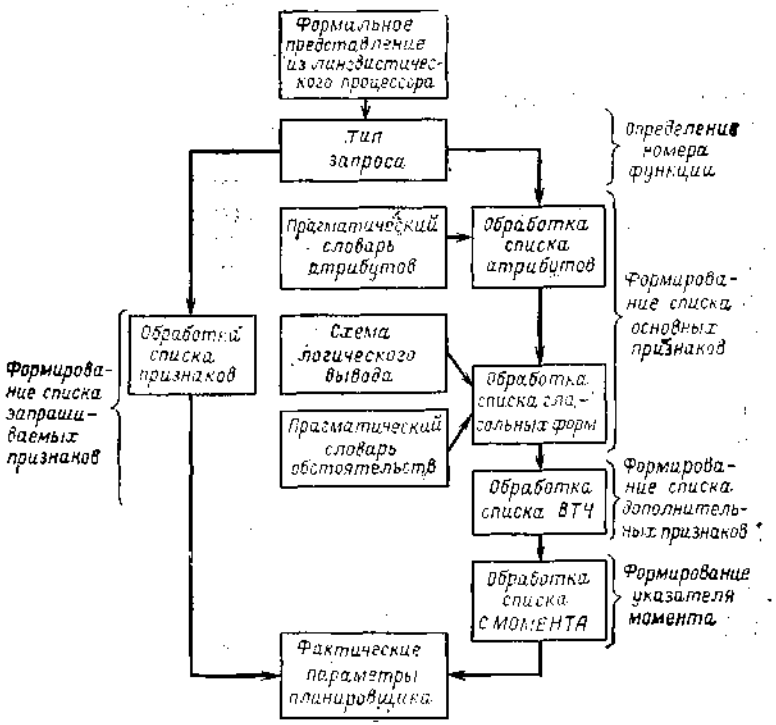


Рис. 4.5

Таблица 4.4

Имя атрибута	Код признака	Значение признака	Указатель подмассивов
801 (импорт)	760 (вид перевязок)	61682	2
802 (экспорт)	760 (вид перевязок)	61938	1
810 (исправный)	761 (техническое состояние)	241	0
...	...	...	...

значения +1 или —1. На основе этого списка из массивов данных выбираются объекты, у которых упомянутые в списке признаки принимают заданное значение при условии, что ключ равен +1, и значение, отличное от заданного, при условии, что ключ равен —1. Для формирования списка основных признаков используются списки атрибутов и глагольных форм, полученные из лингвистического процессора.

Атрибуты рассматриваются как знаки, указывающие одновременно номер признака и его значение. Поэтому для обработки списка атрибутов в программу вводится прагматический словарь атрибутов. Каждая статья этого словаря содержит четыре позиции: имя атрибута, код признака, значение признака, указатель подмассивов базы данных (табл. 4.4).

Указатель подмассивов в четвертой позиции этой таблицы принимает значение, равное нулю в том случае, когда код подмассива не может быть однозначно определен по атрибуту.

В программе организованы просмотр списка атрибутов и поиск атрибута в первой позиции словаря. Если при этом код атрибута будет найден, то код соответствующего признака вместе со значением заносится в массив основных признаков. Ключ списка устанавливается равным +1. Если атрибут не будет найден в прагматическом словаре, то выводится сообщение: «В прагматическом словаре нет атрибута такого-то» и программа продолжает работу.

После окончания работы с атрибутами программа переходит к работе с глагольными формами. В отличие от декларативного задания знаний для работы с атрибутами при работе с глагольными формами оказалось более удобным представлять знания в процедурной форме.

Простые глаголы, заведомо идущие без обстоятельств, обрабатываются подобно атрибутам, т. е. коду глагола приписывается определенный признак и его значение, которые затем заносятся в массив основных признаков.

Обработка глаголов, требующих после себя обстоятельства, происходит по специальным эвристическим правилам. Так, глаголы *прибывать* и *выгрузить* указывают на группу массивов прибытия, т. е. устанавливается $P1=1$. Глагол *отправить* при отсутствии индикатора модальности указывает на группу массивов отправления ($P1=3$) и группу массивов терминальных ($P1=2$) при наличии индикатора модальности. Глаголы *погрузить* и *отгрузить* означают, что контейнеры находятся на терминале, т. е. $P1=2$. При отсутствии модальности глагол *выгрузить* означает, что признак выгрузки присутствует, а при наличии предшествующего глагола типа *осталось*, что признак выгрузки отсутствует. Аналогично при отсутствии индикатора модальности глаголы типа *отгрузить*, *погрузить* означают, что признак погрузки присутствует, а совместное их употребление со словом типа *осталось*, что признак погрузки отсутствует. Соответствующая информация заносится в список основных признаков.

На следующем этапе производится обработка обстоятельства в глагольной группе, если оно там присутствует. Прежде всего по обстоятельству и глаголу выводится направление движения контейнеров — с суши на море или с моря на сушу. При этом определяется значение подмассива *P2*. Например, если что-то выгружено с судна или погружено на поезд, то движение объекта происходит с моря на сушу и его надо искать среди импортных ($P2=1$). Если что-то выгружено с поезда, то движение направлено с суши на море и $P2=2$.

Далее по обстоятельству определяется код вида транспорта. Для этого используются декларативные знания, представленные в программе в виде табл. 4.5.

При работе программы в этой таблице ищется код рассматриваемого обстоятельства. Если он там присутствует, то в массив основных признаков заносится пара: код вида транспорта — его значение. Код вида транспорта принят равным 750, а значение считывается во второй позиции таблицы.

Таблица 4.5

Код обстоятельства	Значение признака	Код признака
601 (*судно*)	61936	601 (*судно*)
603 (*ж/д*)	61684	602 (*поезд*)
620 (*море*)	61936	601 (*судно*)
		...

Третья позиция таблицы используется в том случае, когда за обстоятельством следует его имя собственное, например это может быть имя судна или номер поезда. Число, содержащееся в третьей позиции, служит в этом случае кодом признака, а его значением является имя собственное, присутствующее в запросе. В массив основных признаков заносятся код признака из третьей позиции таблицы и номер ссылки на массив имен собственных в качестве его значения.

При отсутствии искомого обстоятельства в прагматическом словаре обстоятельств выводится соответствующее сообщение и программа продолжает работу. Просмотром и обработкой всех глагольных форм заканчивается формирование массива основных признаков.

Для формирования списка дополнительных признаков применяются следующие эвристические правила.

Устанавливается начальный уровень вложенности, равный нулю, и начинается просмотр элементов списка ВТЧ. Если элемент списка совпадает с кодом признака, то этот код заносится в список дополнительных признаков с указанием «перебрать все значения».

В третью позицию списка дополнительных признаков заносится текущее значение уровня вложенности.

Если элемент списка ВТЧ — атрибут, то следует обращение к прагматическому словарю атрибутов. В нем отыскивается строка, соответствующая данному атрибуту, считывается код признака и его значение и заносится в массив дополнительных признаков с указанием текущего уровня вложенности. Если элементом списка является обстоятельство типа *море*, *железная дорога*, *автотранспорт*, то в массив дополнительных признаков заносится код вида транспорта. Его значение отыскивается в прагматическом словаре обстоятельств. Если элемент списка ВТЧ — обстоятельство типа *судно*, *поезд*, *платформа*, *машина*, то код признака отыскивается в третьей позиции прагматического словаря обстоятельств и заносится в массив дополнительных признаков с указанием «перебрать все значения». Если в элементы списка ВТЧ входит конструкция *по номерам*, то в массив дополнительных признаков заносится код признака *номер контейнера* с указанием «перебрать все значения». Наконец, если в качестве элемента списка ВТЧ фигурирует открывающая скобка, то текущий уровень вложенности увеличивается на единицу. Если скобка закрывающая, то уровень вложенности уменьшается на единицу.

При наличии указателей *с момента* или *дата* для вычисления значения признака используется основной глагол глагольной формы. Если его код обозначить через $Г$, то значение соответствующего признака вычисляется по формуле

$$B = (Г - (Г/100)\,100)\,2 + 930.$$

Таким образом находятся коды признака момента для запросов о времени и дате события. Эта же информация может служить для определения подмассива *P2*.

При обработке запросов, требующих выдать то или иное значение признака для заданного контейнера, переменной B присваивается код запрашиваемого признака.

Прагматический процессор является фактически модулем интерфейса между лингвистическим процессором и базой данных. При смене структуры и содержимого базы данных он должен быть переписан. В нем содержатся прагматические знания о вполне конкретной базе данных со свойственными ей способами и средствами управления массивами и со структурой этих массивов. Кроме того, в прагматический процессор заложены знания о конкретной проблемной среде и об отношениях между объектами этой среды. Поэтому всякая попытка сделать такой процессор универсальным приведет к созданию громоздкой системы, отнимающей большие ресурсы у ЭВМ, значительно замедляющей процесс анализа запроса и в конечном итоге непригодной для практического использования в реальных условиях.

4.5. Примеры

Для иллюстрации работы интерпретатора запросов приведем несколько примеров прохождения запросов через лингвистический и прагматический процессоры и поиска ответов в базе данных.

Пример 4.1. Запрос: *Количество экспортных контейнеров на терминале.*

Комментарий. Запрос выражен в виде утвердительного предложения. Вместо слова *количество* можно было бы употребить любое другое слово или словосочетание, обозначающее количество, например, *сколько, число, какое количество, как много* и т. п. Наконец, определение *экспортных* может находиться после слова *контейнеров*.

Запрос после кодировщика: 2 (*кол**) 802 (*эксп**) 0 (*конт**) 100 (*на **) 600 (*тер**).

Комментарий. В скобках приведены словоформы в том виде, в котором они хранятся в словаре лексики. Сокращенные начертания слов, записанные в словаре, позволяют формулировать запросы, не заботясь о правильности написания оставшихся частей словоформ. Их можно не писать вообще. После кодировщика запрос представляет собой последовательность из пяти чисел, каждое из которых несет синтаксическую и семантическую нагрузку. Так, коды 2 и 0 означают, что эти слова принадлежат к числу опорных, коды которых принимают значения от 0 до 99. Код 802 означает, что слово относится к классу атрибутов. Коды от 100 до 199 принадлежат предлогам, а от 600 до 699 — обстоятельствам места.

Запрос после редактора: 2 (*количество*) 802 (*экспортных*) 0 (*контейнеров*).

Комментарий. Поскольку в исходном запросе идет речь о контейнерах, расположенных на терминале, которые в рамках прагматики системы нигде в ином месте располагаться не могут, то словосочетание *на терминале* здесь избыточно. Поэтому в редакторе сначала производится свертка двух кодов 100 и 600 и результирующий код исключается из дальнейшего рассмотрения. В грамматический анализатор поступает последовательность из трех чисел.

Запрос после грамматического анализатора:

ВОПРОС О ЧИСЛЕ КОНТЕЙНЕРОВ
АТРИБУТ 802

Комментарий. Принадлежность запроса к одному из существующих классов определяется кодом 2 (*количество*) и отсутствием каких-либо иных признаков, указывающих на его принадлежность к другим классам. Единственным указателем поиска является атрибут 802 (*экспортные*). В таком виде запрос поступает на вход прагматического процессора.

Запрос после прагматического процессора:

ФУНКЦИЯ 1
ИСКАТЬ В МАССИВАХ P1 = 2 P2 = 1
ЧИСЛО ОСНОВНЫХ ПРИЗНАКОВ 1
1 760 61938
ЧИСЛО ДОПОЛНИТЕЛЬНЫХ ПРИЗНАКОВ 0

Комментарий. Функция 1 набирается из процедур, обеспечивающих подсчет объектов с заданными параметрами. В качестве параметра выступает единственный основной признак 760 (*вид перевозок*) со значением, записанным в третьей позиции таблицы. Код ключа со значением, равным +1, записан в первой позиции таблицы. Признаки ВТЧ отсутствуют, и этому параметру соответствующей процедуры присвоено нулевое значение.

После выработки планировщиком задания на поиск и после его выполнения система выдаст следующий ответ:

ВСЕГО 168

Все подсчеты ведутся на данный момент или относительно данного момента, если в самом запросе момент времени не оговорен особо.

Пример 4.2. Запрос: *Перечислить дефектные контейнеры по датам выгрузки по номерам.*

Комментарий. Запятую после слова *выгрузки* можно не ставить. Ее же можно заменить союзом «и». Слово *перечислить* также можно опустить. По умолчанию будет выдан перечень.

Запрос после кодировщика: 12 (*перч**) 812 (*деф**) 0 (*конт**) 101 (*по **) 901 (*дат**) 320 (*выг**) 101 (*по **) 1 (*ном**)

Комментарий. Помимо присутствовавших в первом запросе классов здесь появились два новых — обстоятельство времени 901 и глагол 320.

Запрос после редактора: 12 812 0 101 901 320 101 1.

Комментарий. В редакторе запрос не претерпел изменений в сравнении с тем, каким он был после кодировщика.

Запрос после грамматического анализатора:

ТРЕБОВАНИЕ ПЕРЕЧИСЛИТЬ
АТРИБУТ 812
В ТОМ ЧИСЛЕ 970

Комментарий. Указателем места поиска является атрибут 812 (*дефектные*), а признаками поиска, причем входящими в категорию дополнительных, являются *дата-выгрузки* 970 и *номер* 1.

Запрос после прагматического процессора:

ФУНКЦИЯ 5
ИСКАТЬ В МАССИВАХ P1=2 P2=0
ЧИСЛО ОСНОВНЫХ ПРИЗНАКОВ 1
1 761 240
ЧИСЛО ДОПОЛНИТЕЛЬНЫХ ПРИЗНАКОВ 2
100 888888 0
970 888888 0

Комментарий. В качестве параметров функции 5 (*перечислить*) выступают три признака: один основной и два дополнительных. Основной признак 761 (*техническое состояние*) принимает фиксированное значение из третьей позиции таблицы. Дополнительные признаки 100 и 970 во второй позиции таблицы имеют специальный код значения, который предписывает использующей его процедуре просмотр всех записей в массиве *P2*. Оба дополнительных признака имеют нулевой уровень вложенности.

Ответ системы будет сформирован генератором выходных форм в следующем виде:

КОД Т/С	ДАТА ВЫГ	ШИФР КОНТ
0	200480	ММУ17789012
0	200480	ММУ18790123
0	250480	ММУ20069421
.	.	.
.	.	.
.	.	.
0	150580	МПС132007
0	170580	МПС229840

В этой таблице приняты следующие условные обозначения. Код технического состояния (КОД Т/С) принимает значение, равное нулю, для дефектных контейнеров. Дата записывается в виде последовательности из шести цифр, которые попарно обозначают число, месяц и год выгрузки контейнера. Как правило, задаваемые системе вопросы не превосходят по сложности приведенные выше. Однако иногда возникает необходимость в выдаче справок, довольно сложных по структуре. Приведем пример запроса, требующего сложного поиска и формирования сложных таблиц.

Пример 4.3. Запрос: *Сколько экспортные контейнеры хранятся на терминале (по номерам (по датам выгрузки и по ячейкам терминала) по типоразмерам (по странам назначения (по грузам)) по техническому состоянию) с момента выгрузки?*

Комментарий. Время хранения контейнеров исчисляется в контейнеро-сутках. Необходимо выдать сведения о времени хранения экспортных контейнеров на терминале в различных разрезах с несколькими уровнями вложенности. Следует отметить, что запрос содержит эллипсис — после слова *сколько* отсутствует слово *времени*. Человек легко восстанавливает его на основе тех общих знаний о мире и о грамматике, которые он получает в процессе обучения. Например, гипотеза о времени в общем контексте запроса могла бы быть высказана уже при анализе морфологических характеристик слова *экспортные*. Однако, как уже говорилось ранее, в рассматриваемом варианте системы нет ни блока морфологического анализа, ни контекстного механизма. Поэтому только подтверждение пользователем интерпретации запроса в лингвистическом процессоре может обеспечить выдачу правильного результата. В данном конкретном случае положение снасает выражение *с момента*, которое может быть употреблено только в контексте запросов о времени.

Запрос после кодировщика: 2 (*ск**) 802 (*эксп**) 0 (*конт**) 302 (*хр**) 100 (*на **) 600 (*тер**) 7 (*отк. ск.*) 101 (*по **) 1 (*ном**) 7 (*отк. ск.*) 101 (*по **) 901 (*дат**) 320 (*выг**) 31 (*и **) 101 (*по **) 607 (*яч**) 600 (*тер**) 8 (*зак. ск.*) 101 (*по**) 712 (*тип**) 7 (*отк. ск.*) 101 (*по **) 719 (*стр**) 360 (*наз**) 7 (*отк. ск*) 101 (*по **) 722 (*груза**) 8 (*зак. ск.*) 8 (*зак. ск.*) 101 (*по**) 54 (*техн**) 761 (*состо**) (8 *зак. ск.*) 104 (*с **) 900 (*мом**) 320 (*выг**).

Комментарий. Открывающие и закрывающие скобки воспринимаются как слова.

Запрос после редактора: 2 802 0 302 7 101 1 7 101 901 320 101 607 600 8 101 712 7 101 709 7 101 722 8 8 101 761 8 902 320

Запрос после грамматического анализатора:

```
ВОПРОС О ВРЕМЕНИ
АТРИБУТ . . . . . 802
С МОМЕНТА . . . . . 320
В ТОМ ЧИСЛЕ . . . . . 1    7
                         970   607   600   8
                   712   7
                         709   7
                               722   8   8
                   761   8
```

Комментарий. Как уже говорилось выше, принадлежность данного запроса к категории запросов о времени выявлено на основе анализа словосочетания *с момента*. Сложную структуру имеет список признаков ВТЧ. Три признака имеют нулевой уровень вложенности: 1 (*номер*), 712 (*типоразмер*), 761 (*техсостояние*). Две группы признаков имеют первый, более глубокий уровень: 970 (*дата-выгрузки*), 607 (*ячейка*), 600 (*терминал*) и 709 (*страна-назначения*). Наконец, признак 722 (*груз*) имеет самый глубокий уровень вложенности.

Запрос после прагматического процессора:

```
В СЛОВАРЕ НЕТ ОБСТОЯТЕЛЬСТВА . . . . . 600
ФУНКЦИЯ 2
ИСКАТЬ В МАССИВАХ        P1=2        P2=1
ЧИСЛО ОСНОВНЫХ ПРИЗНАКОВ . . . . . 2
                0          760        61938   (экспортные перевозки)
                0          970       888888   (дата выгрузки)
ЧИСЛО ДОПОЛНИТЕЛЬНЫХ ПРИЗНАКОВ 6
(номер)          100       888888        0
(ячейка)         607       888888        1
(типоразмер)     712       888888        1
(страна назначения) 709    888888        1
(груз)           722       888888        2
(техсостояние)   761       888888        0
```

Комментарий. В прагматическом словаре обстоятельств данной системы отсутствует обстоятельство места *терминал*, о чем система печатает сообщение и продолжает свою работу. На правильности выдачи ответа это не может сказаться, так как прагматический словарь содержит только те сведения, которые соответствуют знаниям системы о базе данных. В базе же данных хранятся сведения лишь о контейнерах, находящихся на терминале и нигде более. В числе основных признаков присутствует признак 970 (*дата-выгрузки*), полученный на основе интерпретации специальной глагольной формы *с-момента*. Однако такой же признак содержался в списке признаков ВТЧ, поступивших из грамматики. Он должен был присутствовать в сформированной таблице дополнительных признаков. В этом случае он дважды был бы учтен программами поиска основных и дополнительных признаков и дважды присутствовал бы в выходной таблице. Поэтому в прагматическом процессоре предусмотрены проверка таблиц основных и дополнительных признаков и устранение дублирования в пользу основных признаков, когда это не нарушает иерархии сортировки.

4.6. Перспективы развития системы

Приведенное описание системы ДИСПУТ имеет обобщенный характер. Различные реализации систем этого семейства включают в себя только те модули и процедуры, которые необходимы для эффективной работы в рамках заданной проблемной области.

Работа по совершенствованию системы ведется в нескольких направлениях. Во-первых, совершенствуется организация вычислительного процесса для действующих систем, что позволяет использовать их на машинах младших серий при сохранении хорошего времени реакции на запросы пользователей.

Во-вторых, расширяются возможности проектируемых систем за счет введения новых блоков и совершенствования алгоритмов работы старых. Так, изменены принцип построения словаря и способ его хранения в памяти ЭВМ. Он выполнен в виде дерева с сохранением всего слова полностью и с указанием его морфологических и синтактико-семантических характеристик, отображающих грамматическую роль слова в предложении и его прагматическую ориентацию в возможных темах диалога. Вводятся блоки морфологического анализа и синтеза. Блок морфологического анализа работает лишь в том случае, когда без него невозможен однозначный анализ всего предложения. В качестве одной из тем диалога введена тема о самой системе: о словарях, о содержимом базы данных, о ее структуре и т. п. Наконец, очень важным нововведением является контекстный механизм, позволяющий обрабатывать анафорические отсылки и эллипсы. Это позволяет задавать последовательность вопросов, не тратя усилий на набор повторяющихся слов и конструкций.

В-третьих, развиваются технология и обеспечивающие ее средства автоматизации проектирования таких систем.

Наиболее сложным в этом направлении является механизм автоматического синтеза процедурных грамматик на основе их табличного описания.

В-четвертых, создаются средства, позволяющие ориентировать готовые системы на индивидуальных пользователей, работающих с ними. В частности, это касается формирования персонально-ориентированных словарей на базе существующих словарей системы.

В-пятых, создаются набор эффективных процедур, осуществляющих необходимые операции над базой данных, и механизм гетерархического управления этими процедурами.

ГЛАВА ПЯТАЯ

МНОГОЦЕЛЕВАЯ ИНТЕЛЛЕКТУАЛЬНАЯ ВОПРОСНО-ОТВЕТНАЯ СИСТЕМА (МИВОС)

5.1. Общая характеристика системы

В настоящей главе рассматриваются теоретические и модельные основы, а также вопросы реализации системы МИВОС. Создание этой системы велось исходя из отчетливого понимания того факта, что, несмотря на большое число разработок в этой быстро прогрессирующей области исследований, единой, законченной теории машинного понимания естественного языка и методологии разработки естественноязыковых систем пока не существует. Поэтому основные цели проекта МИВОС заключались в создании:

экспериментальной базы для дальнейших исследований в области машинного понимания естественного языка;

средств построения прикладных систем общения;

прототипного программного обеспечения подобных систем.

В реализации МИВОС применялась библиотечная система поддержки разработки математического обеспечения (БСПР МО) для ЭВМ БЭСМ-6. Выбор данной системы определялся возможностью использования в рамках БСПР МО уже разработанных средств автоматизации программирования, а также обеспечения быстрой и качественной разработки новых средств на базе метаязыка РЕФАЛ, ориентированного на символьные преобразования произвольных текстов.

Использование БСПР МО для реализации естественноязыковых систем позволяет, кроме того, соблюсти и такие важные принципы разработки большого программного продукта, как моделирование ключевых проектных решений; реализация прототипа системы на языках высокого уровня; стандартизация межмодульного интерфейса; введение функциональной избыточности и параметрической универсальности на уровне отдельных частей системы, а также организация качественного сопровождения всего проекта программной документацией.

При разработке системы МИВОС рассматривались два варианта: построение системы на одной ЭВМ и многомашинный вариант. Общая структура первого варианта показана на рис. 5.1. На этом рисунке использованы следующие обозначения: БД — база данных (информационная база для прикладного модуля); БЗ — база знаний (информационная база для языкового модуля); ППП — пакеты прикладных программ; ЕЯС — естественноязыковая система (система, базирующаяся на естественном языке); ОС — операционная система; T_i — терминал i-го пользователя. Общая структура многомашинного варианта при развертывании системы в виде двухуровневой звездообразной сети ЭВМ показана на рис. 5.2.

Каждый из этих вариантов имеет свои достоинства и недостатки. Так, при реализации ЕЯС на одной ЭВМ система получается более однородной и простой, а следовательно, и более быстродействующей. Вместе с тем для обеспечения эффективного диалога со многими терминалами необходимо использование мощной ЭВМ, а это в свою очередь ведет к нерациональной загрузке ее оборудования. Так, например, центральный процессор, с одной стороны, простаивает в ожидании запросов от терминалов, а с другой — перегружен работой по поддержанию диалога (анализ и интерпретация сообщений, синтез ответов и т. д.).

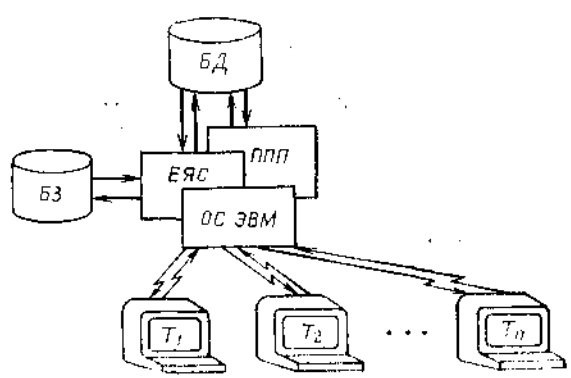


Рис. 5.1

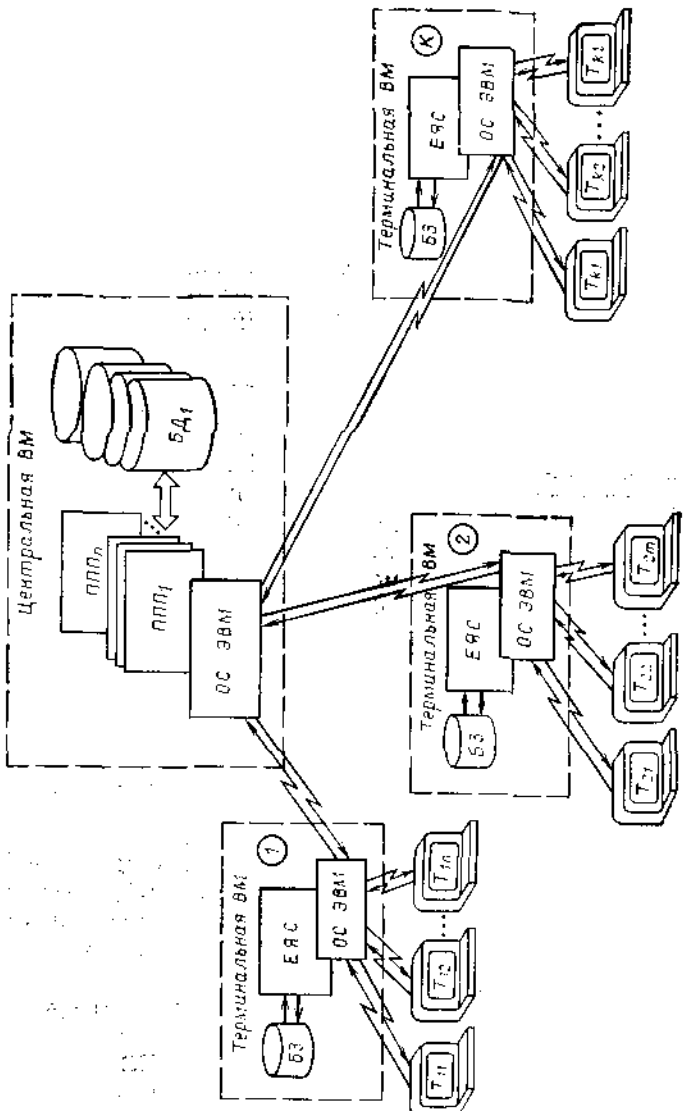


Рис. 5.2

Поэтому, на наш взгляд, более продуктивным является многомашинный вариант ЕЯС. При этом на интеллектуальный терминал (в этом качестве, как правило, будет выступать мини-ЭВМ) целесообразно возложить такие функции ЕЯС, как общение с пользователем, включающее анализ его сообщений и синтез ответов, а также формирование заявок — заданий на обработку в центральной ЭВМ. Такой подход позволяет сбалансировать загрузку оборудования всего комплекса и существенно увеличить производительность работы пользователей. Вместе с тем многомашинная реализация ЕЯС существенно более трудоемка и требует значительно более тщательной проработки функционального уровня всего программного обеспечения.

С учетом вышесказанного первая версия МИВОС развертывалась на одной ЭВМ БЭСМ-6.

Система МИВОС представляет собой комплекс программных средств (операционный компонент), ориентированных на обработку и поддержание знаний, выраженных в специальных системах представления (информационный компонент). *Важнейшей особенностью системы является* проблемная независимость операционного компонента, так как программные средства МИВОС не содержат никаких предположений о предметной области, входном языке, системе семантического представления и возможностях прикладных модулей, а ориентируются исключительно на системы представления знаний о них. Такой подход предполагает, что эти системы представления должны обладать достаточной мощностью для выражения различных теорий и моделей анализа языка, а также учета специфики широкого класса предметных областей. Это позволяет представить методику использования МИВОС в экспериментальных и прикладных целях в виде схемы, показанной на рис. 5.3.

Независимость операционного компонента приводит также к тому, что в общем случае процесс анализа входных текстов в системе МИВОС нельзя представлять в виде уже ставшей традиционной для лингвистических процессоров последовательности этапов морфологического анализа, синтаксического анализа, семантического анализа, семантической интерпретации, так как такая последовательность, будучи заложенной в программное обеспечение системы, представляет собой элемент знаний о модели языка, внесенной в операционный компонент. В гл. 1 уже говорилось, что помимо последовательного способа выполнения отдельных этапов лингвистического анализа существуют смешанные способы. Смешанное выполнение,

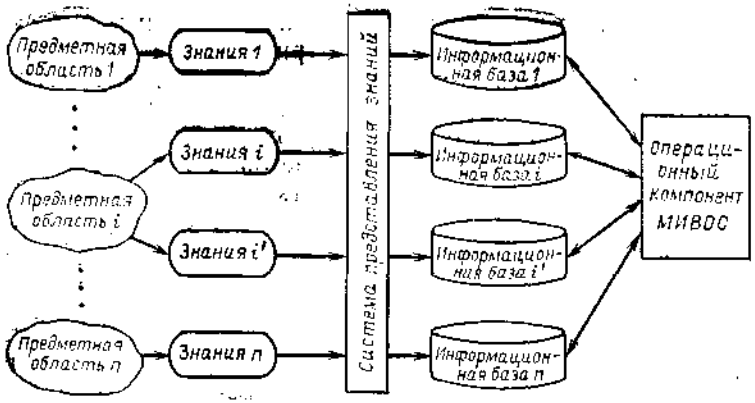


Рис. 5.3

являясь следствием гетерархического подхода к организации системы, позволяет существенно повысить эффективность анализа за счет придания ему большей гибкости и сокращения перебора вследствие замены перебора вширь на управляемый перебор вглубь.

С учетом этих соображений в системе МИВОС используется такая функциональная структура, которая, отображая лишь наиболее общие представления о процессе анализа языка, допускает как последовательное, так и смешанное выполнение отдельных этапов анализа.

Процесс обработки текстов в языковом модуле в общем случае включает два этапа: анализ и интерпретацию. Целью анализа является получение формального представления смысла входного текста в терминах некоторого семантического языка. На этапе интерпретации определяется содержание текста с точки зрения имеющихся в системе знаний о предметной области.

Поэтому операционный компонент МИВОС удобно представить в виде двух процессоров, выполняющих анализ и интерпретацию соответственно. В силу проблемной независимости операционного компонента МИВОС конкретное содержание и последовательность действий, совершаемых над поступающей в процессоры информацией, определяются имеющимися в системе знаниями. Эти знания включают в себя в качестве компонентов информацию нескольких типов: лексико-морфологическую, синтаксическую, семантическую и прагматическую. Конкретизация знаний определяется в конечном счете используемыми в той или иной реализации естественноязыковой

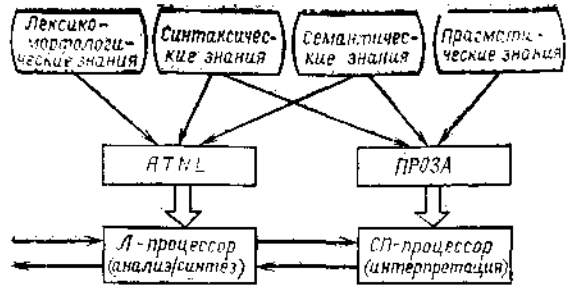


Рис. 5.4

системы моделями (теориями) процессов анализа и интерпретации. Поэтому при разработке МИВОС особое внимание уделялось созданию языка описания знаний, или, как говорят, системы представления знаний, позволяющей, с одной стороны, выражать знания, базирующиеся на достаточно широком классе моделей, а с другой — допускающей эффективную реализацию.

В системе МИВОС используются две системы представления: ATNL и ПРОЗА, которые называют системами представления лингвистических и проблемных знаний соответственно. Система ATNL предназначена для выражения знаний, необходимых на этапе анализа входных текстов, а ПРОЗА — для интерпретации. Использование двух различных систем представления обусловлено желанием сбалансировать в общем случае противоречивые требования эффективности и общности по всей системе в целом.

В общей схеме организации МИВОС, представленной на рис. 5.4, можно выделить три уровня: модельный (описание предметной области и языка), логический (представление этих описаний) и физический (реализация используемых представлений). По существу, уровень реализации представляет собой операционный компонент, включающий лингвистический и семантико-прагматический процессоры. Первый из них (Л-процессор) предназначен для анализа, а в дальнейшем и синтеза текстов, а второй (СП-процессор) выполняет семантическую и прагматическую интерпретации. Информационный компонент МИВОС составляют знания (модельный уровень), выраженные в системах представления ATNL и ПРОЗА.

5.2. ATNL-система представления лингвистических знаний

Как отмечалось выше, ATNL-система применяется для представления лингвистических знаний, используемых при

анализе (синтезе) естественноязыковых текстов. При разработке такой системы учитывалось, что:

лингвистические знания, необходимые для анализа, определяются в первую очередь моделью (теорией языка);

при реализации модели могут использоваться различные алгоритмические языки. При этом обычно модель погружается, «зашивается» в программу анализатора;

поскольку одной из особенностей МИВОС является независимость программного обеспечения от модели языка, то для реализации ее оказалась необходимой разработка такого представления и, следовательно, языка, который позволил бы учесть особенности самого широкого класса моделей и допустить эффективную реализацию.

В качестве основы такого представления в системе МИВОС используется формализм расширенных сетей переходов ATN (Augmented Transition Network).

Расширенные сети переходов устроены следующим образом. Имеется конечный набор диаграмм переходов. Состояниями этих диаграмм являются вершины, соответствующие вспомогательным символам формальной грамматики, а дуги помечены как терминальными, так и нетерминальными символами. Кроме того, дуги помечены условиями, определяющими переход по данной дуге, и множеством действий по построению структуры, которые реализуются при осуществлении перехода по рассматриваемой дуге. Если дуга помечена нетерминальным символом, то это интерпретируется как обращение к диаграмме с таким начальным состоянием. При этом состояние исходной диаграммы, в которую шла анализируемая дуга, запоминается в магазине. Такая конструкция позволяет осуществлять рекурсивные процедуры. Другими словами, ATN для представления грамматики используют по существу диаграммы переходов. Однако условие прохождения дуги теперь распадается на необходимое и достаточное (истинность некоторого предиката, значения переменных которого зависят в общем случае от предыдущих этапов анализа). В расширенной сети переходов введены именуемые регистры (переменных) со стековой структурой и достаточно представительный набор действий, позволяющий проверять, изменять, удалять и объединять их содержимое. Порядком и моментами выполнения действий управляет теперь сам программист, что обеспечивает поддержку режима смешанной обработки.

Пример 5.1. Для иллюстрации сказанного рассмотрим сначала, как работает обычная диаграмма переходов, соответствующая

автоматной грамматике. Для нее характерно то, что дуги отмечаются только терминальными символами. На рис. 5.5 показана такая диаграмма переходов. С помощью ее можно образовать множество фраз известного стихотворения в переводе С. Маршака. Недетерминированность означает, что последовательность предложений может выда-

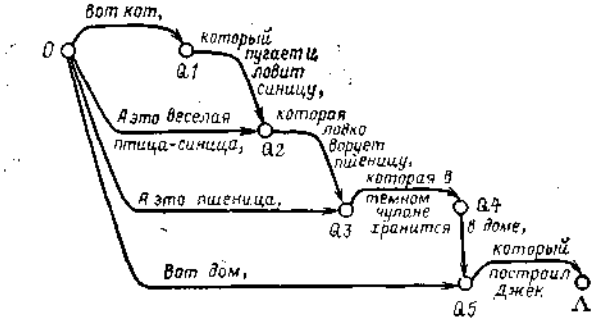


Рис. 5.5

ваться такой диаграммой переходов в любом порядке. Эту же диаграмму переходов можно использовать и при анализе всех четырех фраз.

Пример 5.2. На рис. 5.6 показана диаграмма переходов, соответствующая контекстно-свободной грамматике и предназначенная для анализа тех же четырех фраз стихотворения. Здесь дуги нагружены нетерминальными и терминальными символами. Нетерминальным является символ ГГ (глагольная группа), а остальные (терминальные) символы суть: ЧАСТИЦА — а, вот; относительное местоимение (ОТН. МЕСТОИМ.) — который, которая; указательное местоимение (МЕСТОИМЕНИЕ) — это; существительное (СУЩ.) — Джек, дом, кот, птица, чулан, пшеница, синица; ПРИЛАГАТЕЛЬНОЕ — веселая, темный; ГЛАГОЛ — построил, хранится, ворует, ловит, пугает; СОЮЗ — и; ПРЕДЛОГ — в; НАРЕЧИЕ — ловко.

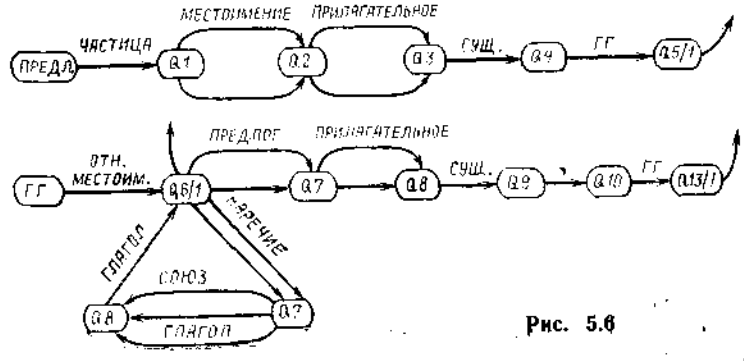


Рис. 5.6

Однако ни автоматная грамматика, ни контекстно-свободная грамматика не позволяют порождать трансформационные преобразования фраз, о которых речь шла в гл. 1 книги. Расширенные сети переходов обладают такой возможностью. Именно поэтому они нашли такое широкое применение в наиболее продвинутых диалоговых системах.

В системе МИВОС расширенные сети переходов являются методологическим базисом создания специального языка представления лингвистических знаний.

Реализация представления, очевидно, предполагает использование некоторого алгоритмического языка. Так, например, в системе ДИЛОС таким языком является LISP, а в известной системе ПОЭТ — PL/1 и АССЕМБЛЕР. Однако для адекватного и удобного описания анализаторов предпочтительнее использовать специальный язык программирования высокого уровня, выступающий в качестве и системы представления лингвистических знаний, и средства реализации анализаторов.

В системе МИВОС таким языком является модифицированный язык расширенных сетей переходов ATNL (Augmented Transition Network Language). ATNL — МИВОС относится к классу языков сентенциального типа и обладает следующими характерными свойствами:

универсальностью в классе решаемых задач;

мощными изобразительными средствами для описания широкого класса моделей языков и всевозможных стратегий и алгоритмов анализа;

использованием рекурсий произвольной глубины и сложности;

высокой структурностью программ;

характерными механизмами языков программирования искусственного интеллекта (сопоставление по образцу, автоматический возврат, средства отладки).

Таким образом, лингвистические знания выражаются в системе МИВОС в виде ATNL-программы, которая в общем случае состоит из трех разделов:

словарей, используемых при анализе (VOCAB);

нестандартных функций, повышающих эффективность разбора (DEFINE);

алгоритма анализа входных текстов (NET).

Учитывая вышесказанное, структуру ATNL-программы можно описать следующим образом:

```
((VOCAB...)
    (DEFINE...)
        (NET...))
```


Разделы VOCAB и NET являются обязательными разделами ATNL-программы, а раздел DEFINE — факультативным.

Раздел VOCAB в свою очередь состоит из двух секций: VOCDS и VOCVL. В первой из них определяется, если это необходимо, морфологическая структура используемых в языке общения слов. Так, например, если хотим описать слово с помощью составляющих, рассмотренных в гл. 1, в секции VOCDS появится шаблон вида

```
СЛОВО ((ПРЕФИКС, NIL)+СЛ=ОБР=ОСН+СУФ!+
       +(ОК, NIL)+(ПОСТФ, NIL)).
```

Для указания структуры составляющих шаблона используется знак «+»; знак «!» указывает на возможность повторения конструкции, после которой он стоит; круглые скобки объединяют альтернативные варианты составляющих слов.

Составляющие, поименованные в шаблонах из секции VOCDS, должны быть описаны в секции VOCVL. Кроме того, в этой же секции описываются и словари словоформ, для которых морфологический анализ не требуется. Каждый словарь в секции VOCVL определяется своим именем и телом, которое в свою очередь может быть задано явным перечислением словарных статей, а также функциями READ или FILE. Первая из них специфицирует ввод словаря с внешнего носителя, а вторая определяет доступ к словарям, введенным заранее. Словарная статья содержит поле ключа (словоформа, префикс, основа и т. п.) и поле именованных характеристик. Значения характеристик могут быть как литералами (например, МУЖ для рода или ИМ, ВИН для падежа), так и функциями. В последнем случае значение соответствующей характеристики, одно или несколько, вычисляется. Наконец, значением характеристики может быть образец, свободные переменные которого заполняются в процессе анализа.

Раздел DEFINE, как указывалось выше, содержит определения нестандартных функций. Такие функции могут быть использованы для описания тех действий, которые в языке ATNL не определены или выполняются неэффективно, а также для интерфейса ATNL с другими системами программирования.

Последним разделом ATNL-программы является описание сети, определяющей алгоритм анализа. Структура этого раздела представляется следующим образом:

```
(NET
((<точка-входа-в-сеть>...)),
```

где на месте многоточия располагается собственно описание сети. Неформально такая сеть является совокупностью мультиграфов, вершины которых соответствуют нетерминальным символам грамматики языка общения, а дуги помечены терминалами (конкретными словоформами, литералами или категориями, наименования которых совпадают с именами словарей), пустыми символами или нетерминалами (именами вершин-состояний).

Описание сети в языке ATNL фрагментируется по кустам. При этом каждый куст задается своей вершиной и исходящими из нее дугами. Формат описания куста следующий:

(<состояние> <дуга> {<дуга>}).

По типам пометок дуги делятся на CAT PUSH и TST. Кроме того, для индикации заключительных состояний в сети вводится дуга POP.

В соответствии с формализмом расширенных сетей переходов для прохождения любой из дуг требуется выполнение необходимых и достаточных условий. Достаточные условия являются предикатами, определенными на множестве значений именованных регистров, заполнение которых осуществляется действиями, выполняемыми при прохождении других дуг сети. Необходимые условия различны для разных типов дуг и обеспечиваются некоторыми встроенными механизмами. В частности, для дуг TST и POP необходимые условия выполняются *по умолчанию* всегда. Для дуги CAT необходимое условие принимает значение T (истина), если текущее слово входного текста совпадает с одним из слов в заданном на дуге словаре (с учетом морфологического анализа или без него) либо текстуально совпадает с пометкой дуги. Вычисление истинности необходимого условия на дуге PUSH связано с рекурсивным обращением к заданному подграфу сети и возможностью выхода из него через одно из заключительных состояний. В случае неудачи на подграфе срабатывает механизм автоматического возврата.

В рассматриваемой версии языка ATNL достаточные условия задаются с помощью предикатов:

(AND <аргумент> <аргумент> {<аргумент>}),
(OR <аргумент> <аргумент> {<аргумент>}),
(NOT <аргумент>),
а также
(EQUAL <аргумент> <аргумент> {<аргумент>}),
(EQOF <аргумент-1> <аргумент-1>) и
(MATH<образец> <аргумент-1>).

Первые четыре предиката являются условиями без побочного эффекта, а в качестве аргументов допускают тождественные условия *T*, ( ) и NIL, формы или любой из приведенных предикатов.

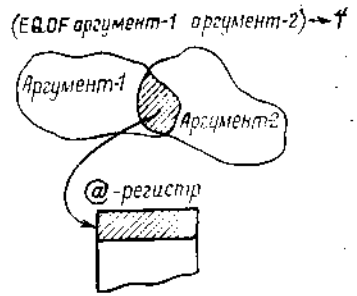


Рис. 5.7

По сравнению с начальным языком Вудса в ATNL введены условия с побочным эффектом EQOF и MATCH. Первое из них служит для проверки непустоты теоретико-множественного пересечения значений двух аргументов, каждый из которых может быть переменной (в этом случае берется значение соответствующего регистра) или формой (рис. 5.7). Побочный эффект EQOF-условия заключается в том, что в специальный системный регистр

@ работающий по принципу LIFO-стека*, заносится

общая часть значений аргументов (см. рис. 5.7). С помощью MATCH-условия проверяется возможность сопоставления значения второго аргумента с образцом — первым аргументом. При этом означиваются переменные образца и в случае удачи производится замена левой его части на правую, которая помещается в специальный системный регистр # (рис. 5.8). Таким образом, в верхуш-

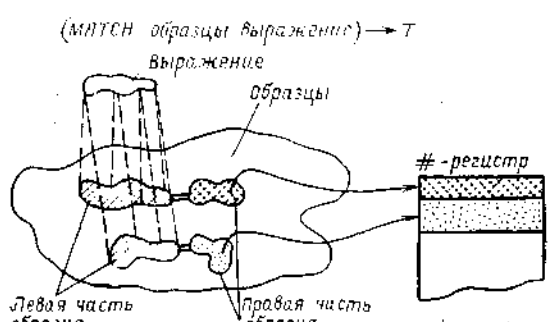


Рис. 5.8

* LIFO есть дисциплина обслуживания по принципу: последний пришел, первым обслуживаешься.

ке #-регистра, работающего по принципу LIFO-стека, образуются одна или несколько записей с результатами успешного отождествления выражения и образца из последнего истинного MATCH-условия.

Важно отметить, что стеки @ и # не очищаются автоматически при переходе от одной фразы языка общения к другой. Поэтому они не только используются в процессе означивания составляющих в пределах одной фразы, но и обеспечивают возможность обработки анафорических ссылок (см. рис. 5.8).

Проход по дуге сопровождается, как правило, выполнением некоторых действий, связанных с записью значений в определенные именованные регистры-стеки. Так, действие SETR специфицирует присваивание на текущем уровне рекурсии, SENDR позволяет передать значение на уровень ниже, а LIFTR — на уровень выше текущего. Кроме того, значение может быть вытолкнуто из стека действием UP и погружено в стек действием DOWN. В последнем случае в верхушке соответствующего регистра образуется пустая позиция.

Для перехода к следующему кусту используются заключительные действия. В ATNL имеются три типа заключительных действий:

переход к следующему кусту без считывания информации из входной строки (JUMP);

переход со считыванием крайнего левого (TC) или крайнего правого (OT) слова из оставшейся части входной строки;

переход по значению системного регистра # (GO).

Использование двух разновидностей перехода со считыванием позволяет увеличивать эффективность разбора и, что более важно, применять ATNL для реализации анализаторов речи.

Пример 5.3. Эту возможность можно проиллюстрировать на примере анализа слов-перевертышей. Соответствующая диаграмма дана на рис. 5.9.

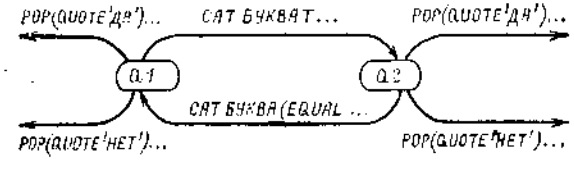


Рис. 5.9

Запись имеет следующий вид:

```
<СЛОВО=«ПЕРЕВЕРТЫШ»>::=<БУКВА><СЛОВО=
=«ПЕРЕВЕРТЫШ»> <БУКВА>
|<БУКВА>|<ПУСТО>
<БУКВА>::=А|Б|В|Г|...|Я
<ПУСТО>::=
((VOCAB
      БУКВА ((А)(Б)(В)(Г)...(Я)))
(NET Q1
  (Q1
     (CAT БУКВА T (SETR RБ *) (OT Q2))
     (POP(QUOTE'ДА')(EQUAL * NIL))
     (POP(QUOTE' НЕТ') T))
  ((Q2
     (CAT БУКВА(EQUAL (GETF RБ)*)(UP RБ)(TO Q1))
     (POP (QUOTE'ДА')(EQUAL * NIL))
     (POP QUOTE ' НЕТ')T)))))
```

Среди заключительных действий особое место занимает переход по значению системного регистра, заполнение которого осуществляется в результате побочного эффекта предиката MATCH. Так как первый терм каждой позиции #-регистра является состоянием, с помощью перехода GO легко реализуются всевозможные семафоры (рис. 5.10):

```
(A
  (TST (EQUAL#NIL)(JUMP Q2))
  (TST T...(GO)))
(B
  (TST (EQUAL#NIL)(JUMP Q3))
  (TST T...(GO)))
```

Считывание значений регистров и формирование за-

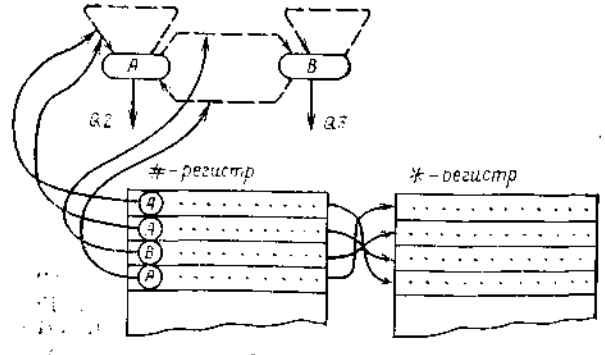


Рис. 5.10

данных структур осуществляется в ATNL с помощью форм. Набор базовых форм обеспечивает считывание информации с сохранением (GETF и GETS) и без сохранения (GETR, *, @, #). При этом форма GETF используется для чтения значений регистров, а GETS — значений именованных характеристик, расположенных в момент обращения к данной форме в верхушке системного регистра. Заметим, что обычно в этом регистре хранится словарная статья, соответствующая обрабатываемому в данный момент слову. Для считывания информации из системных регистров в ATNL введены формы *, @ и #, которые являются по существу сокращенной записью форм (GETR WORD), (GETR ALEF) и (GETR DIES) соответственно. Форма QUOTE служит в ATNL для экранирования ее аргумента от вычисления значения и полностью аналогична соответствующей функции языка LISP. Собственно формирование структур осуществляется посредством форм APPEND, LIST и BUILDQ. Первые две из них позволяют сформировать список значений своих аргументов и соответствуют аналогичным функциям языка LISP. Что же касается формы BUILDQ, обращение к которой имеет вид (BUILDQ ({<образец-структуры>}) {<переменная>}), то среди стандартных LISP-функций аналогов ей нет. В качестве значения эта форма выдает структуру, полученную из образца, путем вычисления значений входящих в него форм, а также путем регулярной замены символов «+» из образца (первый аргумент) на значения соответствующих им по порядку переменных (второй аргумент).

В данном параграфе описаны лишь основные средства языка ATNL и принципы функционирования ATNL-программ. Полный синтаксис языка приводится ниже, а примеры ATNL-программ рассматриваются в § 5.4.

5.3. ПРОЗА — система представления проблемных знаний

Как отмечалось выше, для представления проблемных знаний в МИВОС используется система ПРОЗА, в терминах которой выражается большая часть информации семантического и прагматического характера. Эти знания определяются используемой моделью предметной области,

задачами, решаемыми прикладным модулем естественно-языковой системы, и входными информационными структурами прикладного модуля.

Для определенности будем полагать, что естественно-языковая система, построенная на базе МИВОС, предназначена для поиска ответов на вопросы относительно текущих, прошлых и будущих (гипотетических) состояний предметной области.

В качестве универсального модельного уровня представления проблемных знаний в системе МИВОС используется формализм, главными компонентами которого являются семантическая сеть и сценарии, вкладываемые в нее. Семантическая сеть служит для фиксации текущего состояния предметной области. Вершины сети делятся на вершины-объекты, соответствующие реальным или абстрактным объектам предметной области, и вершины-предикаты, выражающие отношения между объектами либо характеристики объектов или отношений. В простейших случаях вершины-объекты обозначаются существительными, а вершины-предикаты — глаголами и прилагательными. Все вершины в сети вне зависимости от их типа делятся на общие, описывающие в совокупности закономерности, присущие предметной области, и фактуальные, которые служат для описания текущего состояния предметной области. Дугам семантической сети соответствуют семантические связи, такие как подмножество (*U*), элемент (*E*), характеристика (*H*), значение (*V*), часть (*P*), агент (*A*), объект (*O*), место (*L*), инструмент (*I*) и т. д.

Следующим компонентом общего формализма, лежащего в основе системы представления ПРОЗА, является сценарий, который представляет собой семантико-прагматическую модель целого класса однотипных (с точки зрения отношений между объектами) ситуаций или событий, происходящих в предметной области. Так, например, на рис. 5.11, поясняющем понятие сценария, показаны два класса событий; при этом события одного класса заключаются в изменении пространственной, а другого — субъектной локализации объекта.

В общем случае сценарий может рассматриваться как функция, отображающая структуры, которые получены в результате анализа входных текстов, описывающих однотипные события или ситуации, в прагматически обусловленные функции (действия), выражающие реакцию системы на эти тексты. В частности, такими функциями могут быть функции поиска — трансформации семантической сети.

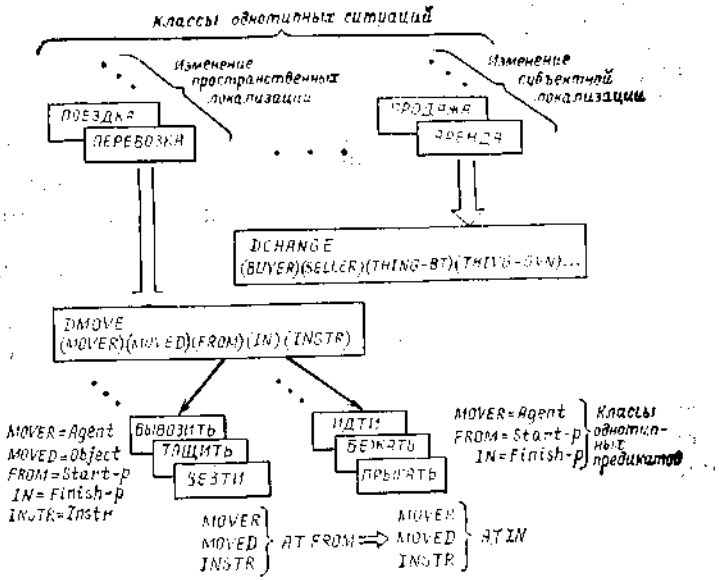


Рис. 5.11

В сценарии выделяются две основные части: декларативная — заголовок и процедурная — тело. Заголовок служит для описания участников, или «действующих лиц», а также «декораций» события или ситуации и состоит из набора параметров — «ролей» сценария, получающих значения в процессе семантической интерпретации — активации сценария. Тело сценария содержит набор функций поиска — трансформации, аргументы которых задаются в виде множеств отношений между ролями сценария. Поэтому после активации, т. е. означивания ролей сценария, каждый аргумент преобразуется в набор образцов, соответствующих некоторой подструктуре (семантическому графу) семантической сети. Для сценария DMOVE, например, в случае обработки сообщения эти функции определяют две трансформации. Первая соответствует неспецифицированному событию, приводящему к ситуации, являющейся предусловием основного события, вторая осуществляет изменения в сети, соответствующие основному событию.

Изложенные выше модельные основы представления проблемных знаний на логическом уровне выражаются в виде ПРОЗА — программ, управляющих работой СП-процессора МИВОС. В языке ПРОЗА предусматривается

организация знаний в виде структурированного множества понятийных единиц — фреймов. Характерными свойствами языка ПРОЗА являются сочетание декларативных и процедурных описаний в рамках одной структуры данных; широкие возможности по определению структур высших уровней (суперфреймов), а также использование централизованного (встроенного) и децентрализованного (программируемого) управления процессами семантико-прагматической интерпретации.

ПРОЗА-программа состоит из трех частей: секции описания (SDECL), секции внесистемных функций (SFUNC) и секции управления (SCONT). Секция описания определяет суперфреймы — отдельные подмножества взаимосвязанных фреймов, служащие для описания объектов, их свойств и отношений между объектами (FNET), событий (FSCEN), контекста диалога (FTEXT), а также некоторых отношений между событиями, например временных (FAFT), причинно-следственных (FCAUS) и др. Заметим, что секция описания является открытой, а введение дополнительных разделов требует определения их семантики в секции SFUNC.

В секции управления определяется дисциплина включения различных функций обработки фреймов. Порядок включения функций зависит от используемой в конкретной реализации естественноязыковой системы модели понимания и составляет (на логическом уровне) управляющую программу работы СП-процессора.

Базовой структурой данных языка ПРОЗА является фрейм, синтаксис которого может быть определен в виде

(<имя-фрейма>)
(<описатель-1> = [<описание>][{<функция>}])
. . . . . .
(<описатель-N> = [<описание>] [{<функция>}])).

Имя-фрейма идентифицирует структуру и носит ссылочный характер, поскольку смысл определяемого им понятия раскрывается через значения описателей фрейма. Описатели именуют связи представляемого фреймом понятия с другими понятиями и делятся на системные, семантика которых понятна СП-процессору, и несистемные, выражаемые либо через системные описатели и фреймы из раздела FNET либо определяемые на языке низкого уровня в секции SFUNC. Значения описателей могут быть заданы декларативно с помощью описаний, процедурно — с помощью присоединенных функций и ассоциативно, в этом случае в качестве значения принимается значение

соответствующего описателя в более общем фрейме, примером которого является данный. Кроме того, для задания значения описателя может быть использована комбинация перечисленных выше способов.

Таким образом, фрейм может рассматриваться как сетевая структура с узлами, соответствующими как конкретным значениям, так и образцам, переменные которых принимают значения из специфицированных множеств фреймов. Поэтому основным встроенным механизмом языка ПРОЗА является механизм означивания этих переменных, на основании которого осуществляется автоматическая конкретизация фреймов, представляющих проблемные знания, с учетом информации, поступающей на вход СП-процессора.

Рассмотрим некоторые из ATNL- и ПРОЗА-программ, разработка которых проводилась с целью уточнения набора функциональных средств языков ATNL и ПРОЗА, проверки программного обеспечения, реализующего процессоры МИВОС, а также исследования различных моделей языка.

Рассматриваемые ниже ATNL-программы позволяют Л-процессору МИВОС анализировать фразы, относящиеся к двум различным подмножествам языка. Первое подмножество является примером «истории». В данном случае это описания ситуаций, в которые попадал интеллектуальный робот. Второе подмножество составили фразы из сообщений Гидрометеоцентра СССР. Эти фразы представляют интерес в связи с тем, что сводки погоды выражаются на весьма ограниченном с семантической точки зрения языке, имеющем много общего с языками общения с базами данных.

Как отмечалось выше, знания, необходимые для анализа и в дальнейшем представляемые в виде ATNL-программы, определяются исходя из модели языка. В качестве такой модели в первой программе использовалась падежная грамматика. Эта модель позволяет выделить следующие знания о языке:

морфологические — сведения о морфологическом строе слов языка, которые, например, позволяют установить, что *пришел* есть форма прошедшего времени, мужского рода и единственного числа глагола *прийти;*

лексические — сведения о словарном составе языка, включающие для каждой основы ряд неизменяемых синтаксических и семантических характеристик, используемых при анализе;

синтаксические — сведения о способах выражения синтаксических категорий, таких как, например, группа существительного (ГС), глагольная группа (ГГ) и т. д., в поверхностных структурах языка;

семантические — сведения об используемом для данного подмножества языка наборе глубинных падежей и об ограничениях на заполнители падежных мест глаголов и их производных;

прагматические — сведения о структуре лингвистических фреймов, в терминах которых должны выдаваться результаты анализа.

Перечисленные выше знания отображаются в разделах VOCAB и NET соответствующей ATNL-программы.

Пример 5.4. Фрагмент словаря глаголов может иметь, например, такой вид:

```
(VOCAB
  . . . . . .
  ГЛАГОЛ
    ('ПРИШЕЛ'
      (ВРЕМЯ='ПР') (ЧИСЛО='Е') (РОД='М') (ВИД=
      'СОВ') (НОРМА='ПРИЙТИ')
      (ОБРАЗЦЫ=
        ((ПРЕДЛОГ=) (ГС=+) (ПАДЕЖ='И') (КГ=
        'ЧЕЛ') (ПР='ОДУШ')=A('A') (ГС) (ПР))
          ((ПРЕДЛОГ='В', НА') (ГС=+) (ПАДЕЖ='В')
        (КГ='ФОБ') (ПР='МЕСТО')=F('F')  (ГС) (ПР))
```

Ключом словарной статьи является здесь словоформа *пришел,* т. е. в данной программе морфологические знания представлены целиком декларативно. Содержимое словарной статьи может быть разделено на две части. Первую часть составляют значения ряда характеристик глагола, таких как наклонение, время, число и т. д. Заметим, что набор характеристик индивидуален для каждого словаря раздела VOCAB.

Вторую часть словарной статьи составляют образцы, каждый из которых имеет левую и правую части. Левая часть задает ограничения на заполнитель глубинного падежа, которые могут быть как синтаксического (отсутствие или наличие определенных предлогов, поверхностный падеж группы существительного), так и семантического (категория, признак) характера. Правая часть образца специфицирует имя глубинного падежа (*A* — агент, *F* — конечная точка, *S* — начальная точка и т. д.) и форму выражения его заполнителя. Знак «+» обозначает свободную переменную, которая в данном случае отождествляется с синтаксической категорией ГС.

Если бы захотели воспользоваться средствами описания морфологии, доступными в рамках ATNL для разработчика Л-процессора, потребовалось бы внести в секцию VOCDS описание шаблона глагола вида

ГЛАГОЛ (ОСНОВА+СУФ!+(ОКОНЧ, NIL)),

а в секции VOCVL вместо словаря ГЛАГОЛ появились бы словари ОСНОВА, СУФ и ОКОНЧ. При этом такие морфологические характеристики, как РОД и ЧИСЛО, связывались бы с окончаниями, ВРЕМЯ и ВИД — с суффиксами, а НОРМА и ОБРАЗЦЫ были бы отнесены к соответствующей основе.

Раздел NET рассматриваемой ATNL-программы состоит из описаний 45 вершин расширенной сети переходов, определяющих собственно алгоритм анализа входных вопросов и сообщений. Графическое представление фрагмента верхнего уровня сети (в упрощенной нотации) приведено на рис. 5.12. Этот фрагмент показывает, как выполняется анализ сообщений, синтаксическая структура которых имеет вид {ГС} ГГ {ГС}, например «Вечером робот пришел в лес», «В лесу находится пруд» и т. п.

Как следует из рис. 5.12, прохождение сети сопровождается обращениями к подсетям нижних уровней, об этом говорят дуги PUSH ГС и PUSH ГГ. В состоянии $Q31$ логическая функция MATCH осуществляет сопоставление образцов глагола, полученных в результате прохода по дуге PUSH ГГ, со структурами, накопленными в регистрах при проходах по дугам подсети ГС. В случае успешного сопоставления в специальный системный регистр # заносятся правые части образцов. После этого в состоянии Q222 специальное действие GO вызывает переходы в состояния, соответствующие падежам, накопленным в регистре #. Возврат в состояние Q222 для выбора следующего падежа осуществляется стандартными средствами ATNL и, как правило, сопровождается формированием отдельных составляющих выходной структуры. Построение полного лингвистического фрейма осуществляется с помощью форм на заключительной дуге POP.

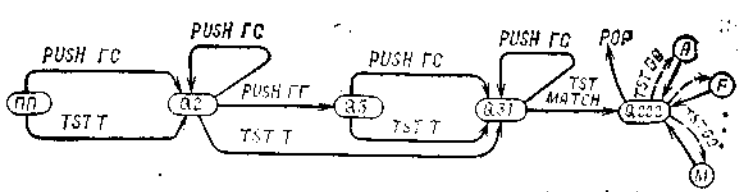


Рис. 5.12

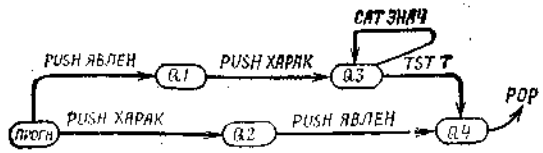


Рис. 5.13

Следующая ATNL-программа была разработана на основе модели семантической грамматики. В данном случае семантическая грамматика это обычная КС-грамматика, нетерминалы которой соответствуют определенным смысловым составляющим входных текстов, а терминалы — семантическим классам, образуемым словами и словосочетаниями входного языка. При этом каждое слово или словосочетание может входить только в один класс. Поэтому лексические знания, содержащиеся в разделе VOCAB этой программы, составляют сведения о том, к какому семантическому классу относится слово или словосочетание. Все остальные виды знаний представлены разделом NET, содержащим описание расширенной сети переходов, построенной на основании семантической грамматики. Фрагмент верхнего уровня этой сети показан на рис. 5.13. Понятия ЯВЛЕН и ХАРАК являются нетерминалами данной грамматики и соответствуют описанию явления погоды и совокупности пространственных и временных характеристик. В качестве примера приведем два предложения, разбор которых осуществляется в соответствии с верхним и нижним путями сети, изображенной на рис. 5.13. При этом в скобки с нижними индексами будем заключать отдельные составляющие:

(Минимальная температура)$_{\text{ЯВЛЕН}}$ (Сегодня ночью в Москве)$_{\text{ХАРАК}}$

(2 градуса)$_{\text{ЗНАЧ}}$;

(В последующие двое суток)$_{\text{ХАРАК}}$ (Сильный туман)$_{\text{ЯВЛЕН}}$

Интересно сравнить временные характеристики анализа, проводимого в соответствии с моделями различных типов. Графики зависимости времени анализа от длины входных предложений, полученные при реализации этого и ряда других Л-процессоров в МИВОС на ЭВМ БЭСМ-6, приведены на рис. 5.14. На этом рисунке сплошной, пунктирной и штрихпунктирной линиями соответственно показаны экспериментальные зависимости для ATNL-интерпретатора, ATNL-конвертора и ATNL-компилятора.

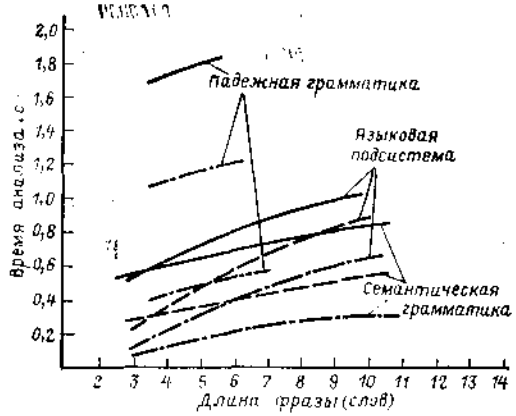


Рис. 5.14

Остановимся теперь на наиболее важных моментах построения ПРОЗА-программ, определяющих семантическую и прагматическую интерпретацию лингвистических фреймов, получаемых в результате анализа входных текстов Л-процессором. Как отмечалось выше, структура ПРОЗА-программы имеет вид

```
(ПРОЗА
        (SCONT . . . . . .)
      [(SFUNC . . . . . .)]
        (SDECL . . . . . .))
```

Обработка входных лингвистических фреймов СП-процессором МИВОС осуществляется на основании управляющей информации из секции SCONT, присоединенных функций, определяющих значения описателей фреймов из секции SDECL, а также встроенными механизмами языка. Секция управления ПРОЗА-программы имеет обычно следующий вид:

(SCONT (COND (e_1p_1) (e_2p_2)...(e_np_n))...),

где e_i — предикат, аргументами которого являются значения описателей лингвистического фрейма, а p_i — действия, выполняемые СП-процессором. В качестве примера приведем фрагмент секции управления:

```
(SCONT
  (COND(  . . . . . .)
    ((AND (EQUAL ЛФРЕЙМ. НАКЛ (QUOTE ИЗ))
          (EQUAL ЛФРЕЙМ. ВРЕМЯ (QUOTE
          ПРОШ))
          (EQUAL ЛФРЕЙМ. ВИД (QUOTE СОВ)))
     ((GO FIND FSCEN ЛФРЕЙ М. ПРЕДИКАТ))
      (PUT FAFT FTEXT) (OUT (FTEXT) ...
      (RETURN)))
      (  . . . . . .)))
```

Аргументами предикатов условного выражения являются значения описателей входного лингвистического фрейма. Этот предикат принимает значение Т для лингвистического фрейма

```
(ЛФРЕЙМ
ПРЕДИКАТ='ПРИЙТИ'
    НАКЛ='ИЗ'
  ВРЕМЯ='СОВ'
       A='РОБОТ'
       F='ЛЕС'),
```

полученного в результате анализа фразы «Робот пришел в лес».

Основные действия СП-процессора специфицируются в данном случае функциями GO FIND, PUT и OUT. Функция FIND находит среди множества фреймов из раздела FSCEN такой, имя которого совпадает со значением описателя ПРЕДИКАТ лингвистического фрейма. В данном случае будет найден следующий фрейм:

```
(ПРИЙТИ
        E=(DMOVE
  MOVER=ЛФРЕЙМ. A
   FROM=ЛФРЕЙМ. S(FIND FNET LOC(AT. OBJ=
          MOVER))
     IN=ЛФРЕЙМ. F
 ERASE=#
 ASSERT=#))
```

Этот фрейм имеет единственный описатель, значение которого указывает на прототипный фрейм DMOVE и доопределяет значения используемых в дальнейшем его описателей. Так, в качестве значения описателя FROM может быть взято значение описателя S лингвистического фрейма при условии, что оно удовлетворяет более общим ограничениям, составляющим значение описателя FROM прототипного фрейма. Если же описатель S отсутствует (как в данном случае), то функция FIND найдет в разделе FNET фрейм AT, описывающий положение (LOC) объекта, являющегося значением описателя MOVER. Знаки «#», используемые в качестве значений описателей ERASE и ASSERT, говорят о том, что из значений соответствующих описателей прототипного фрейма должны

быть исключены все описатели, кроме MOVER, FROM и IN.

Прототипный фрейм DMOVE имеет следующий вид:

```
(DMOVE
      E=FSCEN
  MOVER=ОД-ОБ
  MOVED=ФИЗ-ОБ
   FROM=МЕСТО
     IN=МЕСТО
  INSTR=ТРАНСПОРТ
  ERASE=DESTROY FNET (AT.
                      OBJECT=AND (MOVER
                                  MOVED
                                  INSTR)
                             LOC=FROM
        ASSERT=CREATE FNET (AT.
                      OBJECT=AND (MOVER
                                  MOVED
                                  INSTR)
                             LOC=IN))
```

Функция GO допускает встроенные механизмы, создающие на основании фреймов DMOVE, ПРИЙТИ, FNET и ЛФРЕЙМ фрейм FTEXT следующего вида:

```
(FTEXT
     ACT=DMOVE
  MOVER='РОБОТ'
   FROM='ЛАБОРАТОРИЯ'
     IN='ЛЕС'
  ERASE=DESTROY  FNET  (AT.  OBJ = 'РОБОТ'
                        LOC='ЛАБОРАТОРИЯ')
  ASSERT=CREATE  FNET  (AT.  OBJ = 'РОБОТ'
                        LOC='ЛЕС'))
```

В этом фрейме присутствуют две функции DESTROY и CREATE, выполнение которых ведет к удалению из FNET фрейма, утверждающего, что робот находится в лаборатории, и созданию фрейма, отражающего новое местонахождение робота.

Следующие действия СП-процессора определяются функциями PUT и OUT. Функция PUT позволяет присоединить полученный фрейм FTEXT в качестве значения одного из описателей к фрейму FAFT, фиксирующему последовательность событий в мире, а OUT — вывести фрейм FTEXT на дисплей для контроля. Дальнейшее функционирование СП-процессора определяется реакцией пользователя системы или другими действиями. Как правило, работа СП-процессора заканчивается действием RETURN, специфицирующим завершение ПРОЗА-программы. После этого управление передается вспомогательным механизмам операционной компоненты МИВОС, генерирующим сообщение о том, что система готова к восприятию нового текста.

5.4. Реализация программного обеспечения МИВОС

Изложенные принципы реализации естественноязыковых систем легли в основу реализации проекта МИВОС. При этом, как указывалось выше, одной из основных целей реализации МИВОС была разработка и проверка на практике средств автоматизации проектирования естественноязыковых систем (ЕЯС). В качестве инструментальной использовалась ЭВМ БЭСМ-6.

На рис. 5.15 показана иерархическая диаграмма информационно-программного обеспечения МИВОС. Как следует из этой диаграммы, МИВОС базируется на штатном программном обеспечении ЭВМ и системах представления знаний. Помимо операционной системы ДИСПАК, мониторной системы ДУБНА и диалогового модуля СЕРВИС, разработанного в ВЦ АН СССР, для реализа-

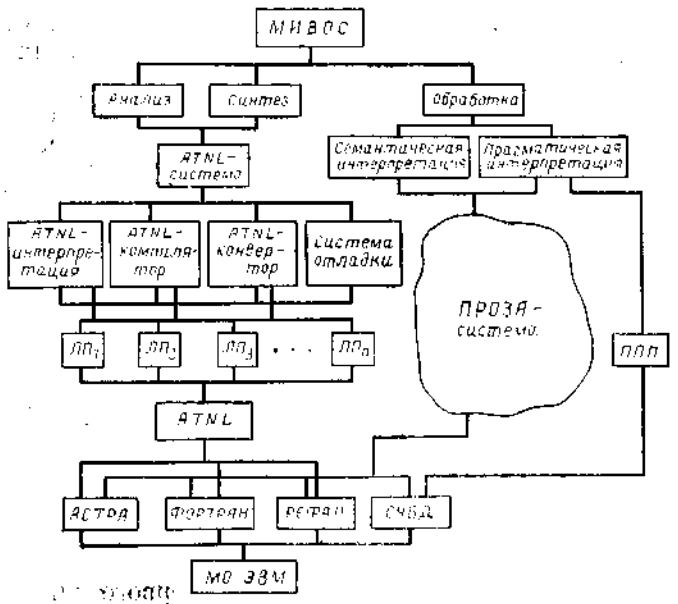


Рис. 5.15

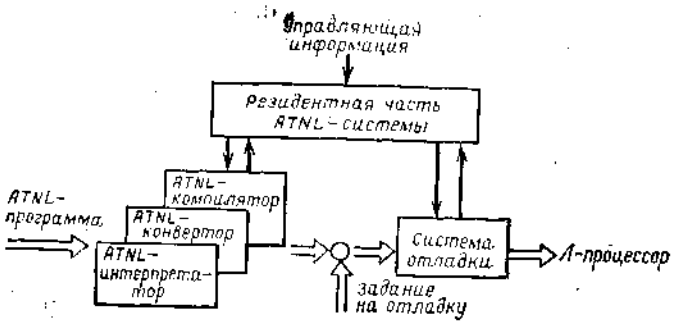


Рис. 5.16

ции собственно программного обеспечения МИВОС наиболее часто используются языки АСТРА, ФОРТРАН и РЕФАЛ. Разработка прикладных лингвистических процессоров ($ЛП_i$) осуществляется на базе языка ATNL и поддерживается соответствующими ATNL-трансляторами. Система разработки семантических процессоров ПРОЗА имеет аналогичную информационно-программную структуру и поэтому на рис. 5.15 не детализируется. Собственно доступ к данным и обработка их в соответствии с заданными целями осуществляются с помощью систем управления базами данных, разворачиваемых на основе архивной системы МАРС-6, и пакетов прикладных программ.

В соответствии с общей структурой МИВОС реализация ее программного обеспечения предполагает программную реализацию систем представления знаний, используемых в данной системе. Как указывалось выше, одной из них является ATNL-система представления, ориентированная на реализацию лингвистических процессоров языков общения конкретных прикладных систем.

В основу ATNL-системы, функциональная структура которой представлена на рис. 5.16, положен специальный язык программирования ATNL. Помимо спектра ATNL-трансляторов и системы отладки, входные языки которых обсуждаются ниже, в состав ATNL-системы входят и управляющие модули, объединенные в резидентную часть. Основной целью работы этих модулей является стыковка ATNL-системы со штатным программным обеспечением ЭВМ БЭСМ-6, а также обработка управляющих ходом трансляции ATNL-программ директив.

С помощью управляющих директив первого типа задается вызов ATNL-системы, а также определяются носители исходных текстов ATNL-программ и результирующих модулей Л-процессоров. Определяются эти директивы правилами языка управления заданиями БСПР МО БЭСМ-6, используемой в проекте МИВОС, и в конечном счете, преобразуются в эквивалентные последовательности директив мониторной системы ДУБНА.

Директивы второго типа относятся к собственным управляющим директивам ATNL-системы. С их помощью определяются тип ATNL-транслятора и необходимость в системе отладки, а также уровень подробности диагностической информации ATNL-транслятора и режим печати выходных документов трансляции.

Опишем входной язык ATNL-системы.

Описание языка структурировано по уровням; при необходимости разъясняется семантика тех или иных конструкций. Для записи синтаксических правил используется расширенная нотация Бэкуса — Наура.

1. Структура ATNL-программы

Д1. <ATNL-программа>::=(<словари> [<нестандартные-функции>] <сеть>)
Д1.1. <словари>::=(VOCAB [<секция-шаблонов>] <секция-словарей>)
Д1.1.1. <секция-шаблонов>::=(VOCDS {<шаблон>})
Д1.1.2. <секция-словарей>::=(VOCVL{<словарь>})
Д1.2. <нестандартные-функции>::=(DEFINE {<описание-функции>})
Д1.3. <сеть>::=(NET(<начальное-состояние> <куст> {<куст>}))
Д1.3.1. <начальное-состояние>::=<состояние>
Д1.3.2. <состояние>::=<идентификатор>

2. Описание шаблона и словаря

Д2А. <шаблон>::=<имя-структуры> (<тело-шаблона>)
Д2А.1. <имя-структуры>::=<идентификатор>
Д2А.2. <тело-шаблона>::=<элем-шаблона>[!]{ + <элем-шаблона>[!]}
Д2А.2.1. <элем-шаблона>::=<имя-словаря> | (<эл-списка-шабл>[!] {, <эл-списка-шабл>[!]})
Д2А.2.1.1. <эл-списка-шабл>::=<имя-словаря> |<литерал>| NIL
Д2Б. <словарь>::=<имя-словаря> (<тело-словаря>)
Д2Б.1. <имя-словаря>::=<идентификатор>
Д2Б.2. <тело-словаря>::=READ | FILE <описание-доступа>[{<словарная-статья>}

Д2Б.2.1. <словарная-статья>::=(<слово>{<характеристика>})

Д2Б.2.1.1. <слово>::=<литерал>

Д2Б.2.1.2. <характеристика>::=(<признак>=<значение>)

Д2Б.2.1.3. <признак>::=<идентификатор>

Д2Б.2.1.4. <значение>::=<литерал>{,<литерал>}|<образец>

С2.2. Функция READ специфицирует ввод тела словаря с перфокарт; при этом введенная информация обрабатывается непосредственно после ввода. Функция FILE определяет доступ к словарным статьям, введенным заранее; при этом, очевидно, <описание-доступа> конкретизируется на уровне реализации языка.

3. Описание функции

С3. Раздел нестандартных функций вводится в язык для повышения эффективности работы, интерфейса с другими блоками естественноязыковой системы, а также штатным программным обеспечением ЭВМ.

С3.1. Для повышения эффективности работы нестандартные функции программируются на специальных языках.

С3.2. Набор языков программирования нестандартных функций определяется конкретной реализацией.

4. Описание сети разбора

С4. В ATNL сеть разбора фрагментируется по кустам.

Д4. <куст>::=(<состояние> <дуга> {<дуга>})

Д4.1. <дуга>::=(CAT <категория> <условие> {<действие>} <заключительное-действие>)
|(TST<условие> {<действие>} <заключительное-действие>)|(PUSH <состояние> <условие> {<действие>} <заключительное-действие>)
|(POP <форма> <условие>)

Д4.1.1. <категория>::=<литерал>|<имя-словаря>|<имя-шаблона>

С4.1.1. Конкретизация категории на CAT-дуге как литерала предполагает текстуальное совпадение с этим литералом очередного слова из входной фразы; если же используется вторая альтернатива, очередное слово должно совпасть с одним из слов указанного словаря; в последнем случае предполагается морфологический разбор очередного слова.

5. Описание условий

Д5. <условие>::<условие-без-побочного-эффекта>|<условие-с-побочным-эффектом>

Д5.1. <условие-без-побочного-эффекта>::=<тождественное-условие>|<И-условие>|<НЛИ-условие>|<НЕ-условие>|<РАВНО-условие>

Д5.1.1. <тождественное-условие>::=T|NIL|( )

Д5.1.2. <И-условие>::=(AND<аргумент> <аргумент>{<аргумент>})

Д5.1.3. <ИЛИ-условие>::=(OR<аргумент> <аргумент> {<аргумент>})

Д5.1.4. <НЕ-условие>::=(NOT<аргумент>)

Д5.1.5. <РАВНО-условие>::=(EQUAL <аргумент> <аргумент> {<аргумент>})

Д5.1.6. <аргумент>::=<условие>|<форма>

Д5.2. <условие-с-побочным-эффектом>::=<пересечение>|<сопоставление>

Д5.2.1. <пересечение>::=(EQOF <аргумент-1> <аргумент-1>)

Д5.2.2. <сопоставление>::=(MATCH <образец> <аргумент-1>)

Д5.2.3. <аргумент-1>::=<переменная>|<форма>

Д5.2.4. <переменная>::=<идентификатор>

С5.2. Побочный эффект предикатов EQOF и MATCH заключается в заполнении определенных системных регистров.

С5.2.1. Значение предиката EQOF — истина (T) в том и только в том случае, когда пересечение значений его аргументов не пусто.

С5.2.1.1. Побочный эффект предиката EQOF состоит в том, что в специальный системный регистр @ ALFF заносится общая часть значений его аргументов.

С5.2.1.2. @ регистр работает по принципу LIFO-стека; таким образом в его верхушке всегда находится ненустое пересечение значений аргумента последнего предиката EQOF.

С5.2.2. Предикат MATCH принимает значение истина (T), если и только если сопоставление конкретного выражения, которое является значением второго аргумента, с образцом — первым аргументом — закончилось успешно.

С5.2.2.1. Побочный эффект предиката MATCH — занесение в специальный системный регистр #DIES выработанного по определенным правилам результата (одного или нескольких) отождествления.

С5.2.2.2. #-регистр работает по принципу LIFO-стека; таким образом, в его верхушке всегда располагается последний вариант результата отождествления выражения и образца из последнего истинного предиката MATCH.

6. Образцы

Д6. <образец>::=<переменная> | ({<описание образца>})

Д6.1. <описание-образца>::=<форма>|(<левая-часть> = <правая-часть>)

Д6.1.1. <левая-часть>::= {<элемент-образца>}

Д6.1.1.1. <элемент-образца>::=(<переменная>= <значение-элемента> {, <значение-элемента>})

Д6.1.1.2. <значение-элемента>::=+|NIL|<пусто> |<литерал>

Д6.1.1.2.1. <пусто>::=

Д6.1.2. <правая-часть>::=<состояние> { <элемент-результата>}

Д6.1.2.1. <элемент-результата>::=+|,|.|= |<целое>|<литерал>|<переменная>| ({<элемент-результата>})

С6.1. Описание образца может быть вычисляемым (альтернатива <форма> в определении Д6.1) или задаваться явно. В конечном счете и в том, и в другом случае каждое из альтернативных описаний имеет левую часть (собственно образец) и правую — результат отождествления образца с конкретным выражением (см. предикат MATCH) в необходимой для дальнейшей работы форме.

С6.1.1. Символ «+» в качестве значения переменной элемента образца специфицирует произвольное значение этой переменной; символы NIL и <пусто> эквивалентны и определяют отсутствие значения у заданной переменной.

С6.1.2. В правой части описания образца можно использовать лишь те переменные, которые были определены в его левой части; при этом используется значение указанной переменной. Символы «+», «,», «.» и «=» в правой части образца специфицируют сами себя, что позволяет осуществить динамическое формирование новых образцов.

7. Описание действий

Д7. <действие>::=(SETR <переменная> <форма>) |(SENDR<переменная> <форма>)| (LIFTR<переменная> <форма>) | (UP<переменная>)| (DOWN <переменная>)

С7. Действия SETR, SENDR и LIFTR специфицируют присваивание значений заданным регистрам (переменным), а действия UP и DOWN — сдвиг указателей, связанных с каждой из переменных, работающих по принципу LIFO-стека.

8. Описание форм

Д8. <форма>::=(GETR<переменная>)| (GETF<переменная>) | (GETS <признак>) |(APPEND <форма> <форма>) | (LIST {<форма>}) | (QUOTE{<Q-аргумент>}) |(BUILDQ ({<образец-структуры>}) {<переменная>}) | * | @ | #

Д8.1. <Q-аргумент>::=+| , | . | = | <целое> |<литерал> | <переменная> | . <переменная>| ({<Q-аргумент>})

Д8.2. <образец-структуры>::=+ | = | <литерал>| <переменная> | <форма> | ({ <образец-структуры>})

С8. Формы GETR, GETF, GETS, *, @ и # предназначены для считывания значений указанных регистров (переменных). При этом формы *, @ и # эквивалентны формам (GETR, WORD), (GETR ALEF) и (GETR DIES) соответственно. Регистр * (WORD) — системный регистр, куда поступает очередное слово из входной строки. Работает регистр *, как и все остальные регистры, по принципу LIFO-стека.

С8.1. Форма QUOTE служит для экранирования Q-аргумента от вычисления его значения.

С8.2. С помощью формы BUILDQ формируются структуры, значения которых вычисляются путем замены всех символов «+» из первого аргумента на значения соответствующих им по порядку переменных из второго аргумента. При этом формы, входящие в первый аргумент, вычисляются.

9. Описание заключительных действий

Д9. <заключительное-действие>::=<переход-без-считывания> | <переход-со-считыванием>

Д9.1. <переход-без-считывания>::=(JUMP<имя-перехода>)

Д9.2. <переход-со-считыванием>::=(TO <имя-перехода>)| (OT <имя-перехода>) |(GO)

Д9.3. <имя-перехода>::=<состояние> | . <переменная>

С9. Заключительные действия JUMP, TO, OT и GO предназначены для перехода из одного состояния сети в другое.

С9.2. Заключительные действия TO, OT и GO сопровождаются считыванием определенной информации в системный регистр * (WORD).

С9.2.1. При использовании заключительного действия TO (OT) очередное (последнее) слово из входной строки считывается в верхушки LIF9-стека * (WORD).

С9.2.2. При использовании заключительного действия GO из верхушки #-регистра выбирается первый терм, который и определяет новое состояние сети. Остальное выражение из верхушки #-регистра переносится в верхушку регистра *.

10. Базовые понятия

С10. Алфавит языка ATNL содержит цифры, буквы латинского и русского алфавитов, а также специальные знаки, набор которых определяется конкретной реализацией.

С10.1. К основным символам языка ATNL относятся ключевые слова (в синтаксических определениях выделены курсивом), а также специальные знаки «-|-», «=», «.», «,», «*», « @ », #. При отсутствии знаков @ и # во входном алфавите ЭВМ, на которой реализуется ATNL, они могут быть заменены эквивалентными им формами (см. С8).

Д10. <базовое-понятие>::=<идентификатор> | <целое> | <литерал>

Д10.1. <идентификатор>::=<буква> { <буква>| <цифра>|— }

Д10.2. <целое>::= <цифра> { <цифра>}

Д10.3. <литерал>::= ' { <любой знак алфавита ATNL>}'

С10.1.1. Идентификаторами обозначаются переменные (шаблоны, словари, регистры), признаки и состояния. При этом множество признаков может быть подмножеством переменных, но множества переменных и состояний, а также шаблонов и словарей не пересекаются.

Важную роль в общей системе поддержки разработки и функционирования естественноязыковых систем играют системы отладки. В качестве примера подобных систем рассмотрим язык отладки ATNL-программ, реализованный в рамках представления лингвистических знаний МИВОС.

Как правило, лингвистические процессоры на базе ATNL являются достаточно сложными рекурсивными программами. Поэтому одним из главных факторов увеличения эффективности отладки ATNL-программ является отладка на уровне входного языка.

Следующий важный фактор, влияющий на эффективность отладки, — уровень детализации отладочной информации, поступающей пользователю. В принципе пользователь должен иметь средства для пошаговой прокрутки всей программы с выдачей полной информации обо всех изменениях на каждом шаге.

Вместе с тем излишняя детализация отладочной информации затрудняет и замедляет процесс отладки. Поэтому для пользователя языка отладки большое значение

имеют средства управления уровнем детализации выдаваемой информации.

С точки зрения успешного использования языка отладки на практике не менее важными факторами являются его простота, гибкость, лаконичность, а также эффективность реализации.

Рассмотренные факторы, влияющие на выбор языка отладки, конечно, противоречивы и не могут быть учтены в равной степени. Вместе с тем в рамках языка отладки ATNL-программ, обсуждаемого ниже, удалось найти необходимый для практики компромисс.

Язык отладки ATNL-программ обеспечивает их пошаговую прокрутку на этапе исполнения во всех режимах с выдачей полной диагностической информации требуемого уровня детализации. Для упрощения процесса отладки широко используется принцип *по умолчанию*. Таким образом обеспечивается гибкость и лаконичность.

Задание на отладку идентифицируется последовательностью специальных директив. Все директивы мнемоничны, что обеспечивает простоту языка.

Ниже приводятся форматы директив отладки и обсуждается информация, выдачу которой они специфицируют.

Начало задания на отладку идентифицирует директива

//TRACE␣[ALL ␣{<состояние>}].

Параметр ALL вводит все известные системе режимы прокрутки во всех указанных состояниях. Если последовательность состояний не задана, производится «сплошная» прокрутка ATNL-программы во всех состояниях сети отлаживаемого лингвистического процессора. Отсутствие параметра ALL указывает на выборочную отладку, режимы которой конкретизируются в последующих директивах.

Конец задания на отладку маркируется директивой

//END␣TRACE или сокращенно// ET. Наличие этой директивы в отладочном режиме обязательно.

Как указывалось выше, режимы отладки задаются с помощью специальных директив. В обсуждаемой версии языка отладки ATNL-программ введены следующие режимы.

Прокрутка по именам состояний и времени их достижения (trace on La Bel and Ti Me) задается директивой

//LBTM␣[{<состояние>}].

При использовании данной директивы пользователю сообщаются имя состояния и время его достижения для всех указанных в параметре директивы точек сети. Если последовательность состояний не указана, заданная информация выдается для всех состояний сети.

Прокрутка по неразобранной части входного текста trace on SENTence определяется директивой

//SENT␣[{<состояние>}]. Неразобранная часть предложения при использовании этой директивы выдается пользователю после лексического преобразования в виде (<слово>) (<слово>) ... (<слово>). Смысл параметра [{<состояние>}] одинаков для всех директив языка отладки и полностью совпадает с семантикой его использования в предыдущей директиве.

Прокрутка по номеру пройденной в заданном состоянии дуги (trace on YES Arc) идентифицируется с помощью директивы

//YESA␣[{<состояние>}]. При наличии в задании на отладку указанной директивы пользователю сообщается порядковый номер (слева направо) той дуги заданного куста сети, для которой истинны необходимые и достаточные условия ее прохождения.

Прокрутка по состоянию стека возвратов (trace on RETurn Registor) специфицируется директивной

//RETR␣[{<состояние>}]. Использование этой директивы позволяет восстановить динамическое дерево разбора исходного текста.

Прокрутка по состоянию системных регистров (trace on SYStem Registor) определяется с помощью директивы //SYSR [{<состояние> [: { <имя-системного-регистра>}]}]. По использовании данной директивы в каждом из указаных состояний пользователю выдается состояние указанных системных регистров. В качестве имени системного регистра можно использовать символы ATNL *, @ ,# или их латинские эквиваленты WORD, ALEF, DIES.

Прокрутка по состоянию используемых на SETR-уровне регистров (trage on SETr Registor) задается директивой

//SETR␣[{<состояние> [:{<имя-регистра>}]}]. Смысл параметра директивы SETR аналогичен смыслу предыдущей директивы. В качестве имени регистра используются идентификаторы переменных из ATNL-программы.

Прокрутка по состоянию используемых на LIFTR и SENDR-уровнях регистров (trace on LiFTr Registor; trace on SENDR Registor) специализируется соответственно директивами

//LIFTR␣[{<состояние> [: { <имя-регистра>}]}] и

//SNDR ␣[{<состояние> [: { <имя-регистра>}]}].

Напомним, что LIFTR-уровень на единицу выше, а SENDR-уровень на единицу ниже текущего SETR-уровня.

Порядок директив, определяющих режим прокрутки в задании на отладку, значения не имеет.

В одном задании на отладку может быть несколько директив, определяющих одинаковый режим прокрутки. В этом случае объединяются параметры одинаковых режимов и соответствующая директива принимается к исполнению.

Таким образом, рассмотренный язык отладки ATNL-программ перекрывает наиболее распространенные режимы прокрутки и способствует существенному повышению отладки лингвистических процессоров на базе ATNL.

5.5. ATNL-трансляторы и их реализация на ЭВМ

Как следует из приведенной выше структуры ATNL-системы, трансляция ATNL-программ может быть выполнена по одной из трех схем (интерпретация, конвертирование или компиляция), каждая из которых обладает своими достоинствами и недостатками. Так, для схемы интерпретации характерны, по-видимому, самая большая простота реализации ATNL и внесения изменений во входной язык ATNL-интерпретатора, но минимальная скорость решения задач анализа. С другой стороны, при компиляции скорость решения определяется уровнем выходного языка компилятора и глубиной оптимизации ATNL-программ и может быть максимизирована в определенных пределах, но изменения входного языка ATNL-компилятора существенно более сложны. В схеме конвертирования, которая по указанным критериям занимает промежуточное положение, скорость решения определяется качеством реализации выходного языка ATNL-конвертора. Простота изменений входного языка определяется уровнем его отличия от выходного языка.

В проекте МИВОС все три транслятора ATNL-системы базируются на метаалгоритмическом языке рекурсивных функций РЕФАЛ, разработаином в ИПМ АН СССР. При этом ATNL-интерпретатор реализован на версии этого языка РЕФАЛ/2 для ЭВМ БЭСМ-6; этот же язык используется и для реализации, и в качестве выходного языка ATNL-конвертора. Что же касается ATNL-компилятора, то он реализуется также на языке РЕФАЛ/2, но в качестве выходного здесь используется макроассемблер мини-ЭВМ СМ-4, что позволяет перейти к многомашинному варнанту системы.

Выбор языка РЕФАЛ в качестве базового при реализации ATNL-системы обусловлен легкостью использования в этом языке рекурсий произвольной глубины и сложности, наличием эффективных средств распознавания различных структур данных и достаточно мощными средствами их обработки. Немаловажным является и тот факт,

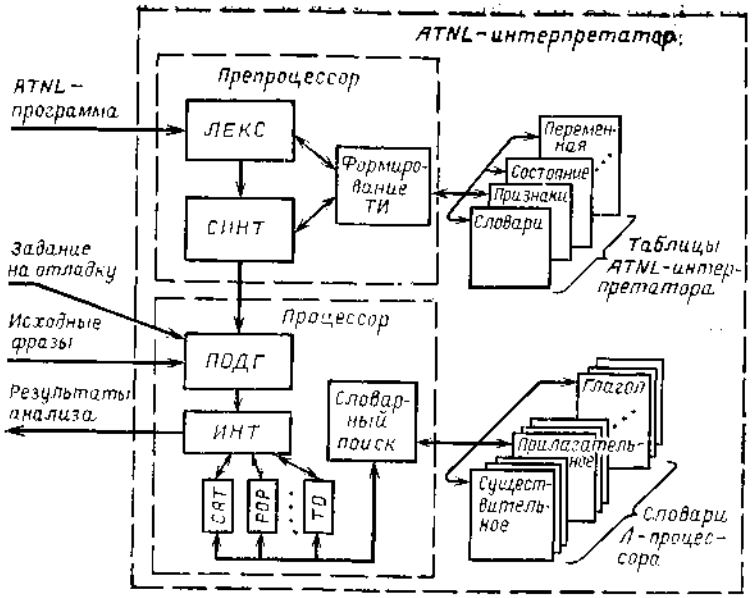


Рис. 5.17

что в настоящее время реализации языка РЕФАЛ имеются практически на всех отечественных и основных зарубежных ЭВМ.

Рассмотрим ATNL-интерпретатор и специфику реализации его на метаязыке РЕФАЛ.

В общей схеме ATNL-интерпретатора, представленной на рис. 5.17, можно выделить два основных этапа: препроцессию и процессию.

Этап препроцессии в свою очередь распадается на предварительное преобразование исходного текста ЛЕКС, повышающее эффективность дальнейшей обработки, и грамматический разбор СИНТ, в процессе которого проверяется синтаксис описания Л-процессора и строится его промежуточное представление.

На этапе препроцессии заполняются таблицы интерпретатора и формируются словари, которые используются на этапе процессии.

Этап процессии ATNL-интерпретатора включает в себя: блок подготовки ПОДГ, в функции которого входят обработка задания на отладку ATNL-программы и формирование результирующего интерпретируемого представления Л-процессора, а также лексическое преобразование входного текста и активация процесса его разбора;

собственно блок интерпретации ИНТ, состоящий из совокупности подпрограмм интерпретации отдельных конструкций языка ATNL в определенном внутреннем представлении.

Реализация ATNL-интерпретатора с использованием метаязыка РЕФАЛ требует некоторых пояснений. Дело в том, что с точки зрения общей классификации трансляторы, создаваемые на базе метаязыка РЕФАЛ (в дальнейшем РЕФАЛ-трансляторы), относятся к классу синтаксически управляемых символьных процессоров. При этом РЕФАЛ-транслятор является рекурсивной функцией, определенной на множестве синтаксически правильных входных программ, и состоит из подфункций, рекурсивно вызывающих друг друга. Отдельные модули транслятора (ЛЕКС, СИНТ и т. д.) программируются в виде РЕФАЛ-предложений и машинных операций и составляют поле правил РЕФАЛ-процессора (рис. 5.18).

Логической базой данных в РЕФАЛ-трансляторах являются исходный текст и его промежуточные представления, таблицы трансляции и выходной текст. В качестве единой физической базы данных может использоваться списочная структура поля зрения. Однако на практике списочная память РЕФАЛ-процессора применяется, как правило, только для обработки текстов. Таблицы транслятора располагаются в линейной памяти, а для записи (считывания) информации служат специальные машинные операции.

Для интерфейса между РЕФАЛ-транслятором и системным программным обеспечением ЭВМ на входе используется инициатор трансляции, а на выходе — терминатор. В функции инициатора входят загрузка РЕФАЛ-процессора, формирование списочной структуры поля зрения и активация некоторой РЕФАЛ-функции. Как правило,

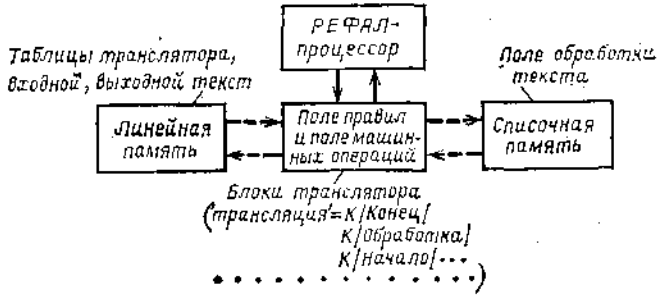


Рис. 5.18

активируемая функция содержит РЕФАЛ-предложение типа

'ТРАНСЛЯЦИЯ'=К/КОНЕЦ/К/ОБРАБОТ-
КА/К/НАЧАЛО/ ...

Выход из РЕФАЛ-транслятора осуществляется с помощью терминатора, предназначенного для организации дальнейшей работы ЭВМ.

В случае ATNL-интерпретатора инициатор трансляции описывается предложением вида

'ATNL'=К/ИНТ/К/ПОДГ/К/СИНТ/К/ЛЕКС/ ....

Как следует из приведенного предложения, первым блоком ATNL-интерпретатора является блок лексического анализа (самой внутренней, а следовательно, и исполняемой в первую очередь будет функция К/ЛЕКС/.). Все преобразования, выполняемые здесь, вытекают из требований к ЛЕКС в обычных трансляторах и принципов эффективного программирования на языке РЕФАЛ.

В результате «традиционных» преобразований текста исходной ATNL-программы (удаление незначащих символов, свертка идентификаторов и ключевых слов, обработка литералов и т. д.) формируется строка разбора или, как говорят, *S*-строка. В РЕФАЛ-трансляторах, кроме того, осуществляется скобочная разметка *S*-строки, в результате которой получается S_R-строка.

Последнее преобразование является принципиальным этапом лексического анализа в РЕФАЛ-трансляторах, так как в отсутствие его результатом ЛЕКС будет линейный список. Скобочная структуризация позволяет явным образом отразить укрупненную иерархию понятий, используемых в исходном тексте, а специфика РЕФАЛ — эффективная обработка скобочных структур. За счет этого существенно увеличивается скорость синтаксического анализа и упрощаются его алгоритмы.

Заметим, что исходная программа очень хорошо структурирована скобками в силу LISP-подобности языка ATNL. Поэтому дополнительной скобочной структуризации в ATNL-интерпретаторе не делается.

Синтаксический анализ в ATNL-интерпретаторе строится по нисходящей модифицированной схеме, содержит два просмотра и инициируется РЕФАЛ-функцией вида

'СИНТ'=К/ОШИБКА/('5').
e_x=К/СИНТ2/К/СИНТ1/e_x ..

Первое из предложений специфицирует аварийный выход из ATNL-интерпретатора в случае ошибок на этапе лексического анализа, а второе — инициирует работу блока СИНТ.

На первом просмотре (функция СИНТ1) осуществляются проверка структуры ATNL-программы и фиксация в таблице имен (ТИ), сформированной в процессе ЛЕКС, типов наименований словарей (функция SVOC), нестандартных функций SDEF и состояний (SNET):

'СИНТ1'(w_1w_3)=К/SVOC/w_1.( )К/SNET/w_3.
$w_1w_2w_3$=К/SVOC/w_1.К/SDEF/w_2.К/SNET/w_3.
e_x=К/ОШИБКА/('6'e_x.() () ()

Синтаксический анализ отдельных конструкций проводится на втором просмотре и выполняется в основном по восходящей схеме. Так, например, синтаксический анализ ATNL-форм производится с помощью функции SФРМ и вызываемых из нее функций следующим образом:

'SФРМ' (/GETR/e_x)=(/GETR/К/АРГR/e_x.)
(/GETF/e_x)=(/GETF/К/АРГR/e_x.)
. . . . . .
(/LIST/e_x)=(/LIST/К/АРГL/e_x.)
(/BUILDQ/e_x)=(/BUILDQ/К/АРГВ/e_x.)
e_y=К/ОШИБКА/('41'e_y).
. . . . . .
'АРГL' w_1e_x=К/SФРМ/w_1. К/АРГL/e_x.

Заметим, что обработка форм *, @ и # на этапе СИНТ, как следует из приведенных выше функций, ничем не отличается от обработки других форм. Это достигается заменой их на стандартную форму GETR, аргументом которой является специальная системная переменная, еще в процессе ЛЕКС. Таким образом, увеличивается однородность последующих блоков ATNL-интерпретатора.

В результате синтаксического анализа исходная ATNL-программа трансформируется в совокупности кустов (рис. 5.19), к каждому из которых обеспечен прямой доступ.

Процессор ATNL-интерпретатора состоит из двух блоков: ПОДГ и ИНТ. В блоке ПОДГ реализуется анализ директив языка отладки и подключение соответствующих им РЕФАЛ-модулей к промежуточному представлению ATNL-программы. Таким образом, при реализации языка отладки в ATNL-интерпретаторе используется принцип конвертирования директив отладки в обращения к специальным отладочным функциям. Такой подход выбран

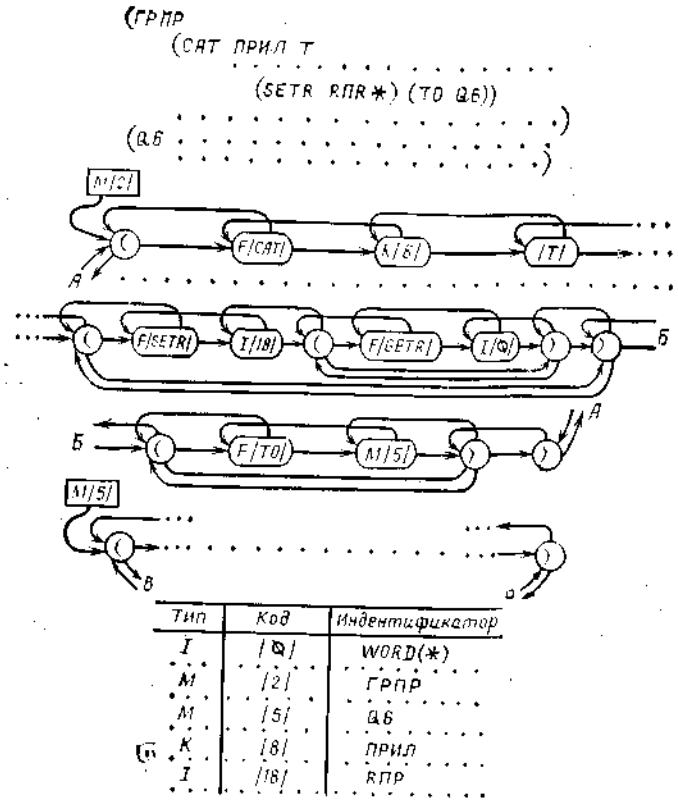


Рис. 5.19

для увеличения скорости работы Л-процессоров в отладочном режиме.

В основу собственно процессора ATNL-интерпретатора положена модульная структура. При этом каждая дуга, форма, действие, условие интерпретируются отдельными блоками в соответствии с их семантикой. Помимо модулей интерпретации процессор содержит и управляющий модуль (ИНТ), основные функции которого сводятся к:

выделению из куста первой дуги с последующей интерпретацией ее (в случае неуспеха выбирается следующая по порядку дуга);

осуществлению перехода в заданное состояние по заключительным действиям (при этом считывается очередной куст и организуется его интерпретация);

запоминанию информации, необходимой для возврата из подграфа;

восстановлению информации при неуспехе на подграфе;

подготовке регистров при обращении к подграфу и возврату из него.

Для иллюстрации работы процессора ниже приводится фрагмент управляющего модуля ИНТ и модуля интерпретации САТ-дуги:

```
'ИНТ' /NIL/(F1e2)e3=К/ИНТ/KF1e2.e3.
      /NIL/=К/ПРОВЕРКА1/К/СТЕК//SВОЗВР/..
      /T/e1=К/ПРОВЕРКА2/К/СТЕК//SВОЗВР/.
      (К/ИСХФРАЗА/.).
      (F1e2)e3=К/ИНТ/KF1e2.e3.
'ПРОВЕРКА 1' '*'=К/ВЫВОД/' — разбор невозможен'.
. . . . . .
'ПРОВЕРКА 2' '*'()=К/ВЫВОД/' — разбор закончен'.
. . . . . .
САТ /T//T/ex=К/EVAL/ex.
    /T//NIL/e1=/NIL/
    /NIL/e1=/NIL/
    /T/w1ex=К/САТ//Т/К/УСЛОВИЕ/w1.ex.
    R1e2=К/САТ/К/СЛПОИСК/R1KR1.(К/WORD/.).e2.
```

Аналогичным образом реализуются и другие модули интерпретации.

Общий объем ATNL-интерпретатора примерно 20 К, при этом собственно процессор занимает 5,7 К. Быстродействие ATNL-интерпретатора существенно зависит от

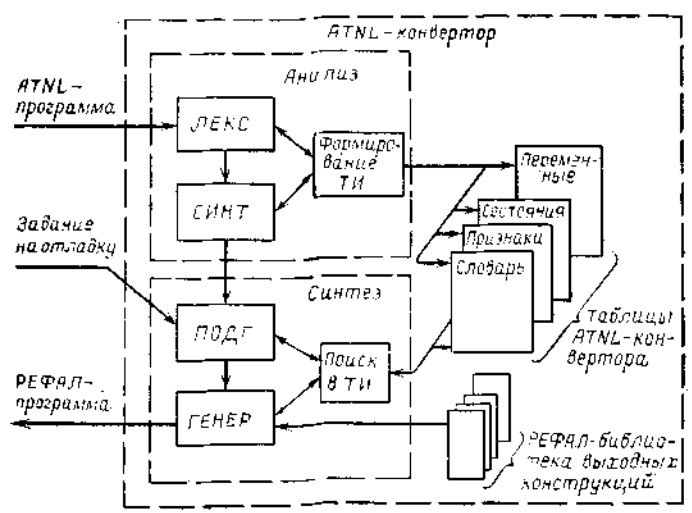


Рис. 5.20

сложности Л-процессора и колеблется для реализованных ATNL-программ от 0,150 до 0,382 с/слово.

Между ATNL-интерпретатором и ATNL-конвертором, как следует из общей схемы последнего, представленной на рис. 5.20, имеется много общего. Действительно, блоки ЛЕКС, СИНТ и формирование ТИ из интерпретатора переходят в конвертор полностью. Кроме того, в конверторе используется и та часть блока ПОДГ, где обрабатывается задание на отладку ATNL-программ. Однако теперь блок ПОДГ значительно усложняется в целях получения внутреннего представления ATNL-программ, удобного для генерации эквивалентной выходной программы на языке РЕФАЛ. Вместо блока ИНТ в ATNL-конверторе используется блок ГЕНЕР, основные функции которого связаны с регулярной заменой конструкций промежуточного представления на соответствующие им фрагменты РЕФАЛ-программы.

В целом блок ГЕНЕР достаточно прост, а представление об алгоритмах его работы можно получить из сравнения ATNL-программы на входе конвертора и соответствующего ему фрагмента выходной РЕФАЛ-программы:

```
        ATNL-программа:
(Q3
        (CAT ЗНАЧ T (SETR RBH1 *) (TO Q3))
        (CAT ЗНА 1 T (SETR R3H 1 *) (TO Q3))
        (TST T(SETR R3HE(BUILDQ ((+)(+))
        R3HA R3H1)(JUMP Q4)))
        РЕФАЛ-программа:
'Q3'ex= K/b1Q3/K/ЗНАЧ/ex ..+
        K/b2Q3/K/ЗНА1/ex ..+
        K/b3Q3/ex( ).
'b1Q3'  (e1)(e2)(e3)(w4e5)e6(e7)/T/+
        K/Q3/(e1(/R3HA/e7))(e2)(e3)(e5)e6w4.
        (e1)(e2)(e3)( )e6(e7)/T/+
        K/Q3/(e1(/R3HA/e7))(e2)(e3)( )e6( ).
'b3Q3' R (e1(/R3HA/e2)e3)e4)(e5)+
        K/b4Q3/(e1e3)e4(e5(e2)).
        e1(e5)+K/b4Q3/e1(e5()).
        . . . . . .
'b5Q3'  (ex)ey((e1)(e2))+
        K/Q4/ex(/R3HE/(e1)(e2)))ey.
```

Как нетрудно заметить, в программе ATNL-конвертора не требуется работа блока ИНТ, вносящего существенные временные издержки при интерпретации. Кроме того, игнорируются тождественно истинные достаточные условия, так что выходная программа конвертора более оптимальна, чем ее промежуточное интерпретируемое представление. Таким образом учет специфики конвертируемой ATNL-программы позволяет увеличить быстродействие получаемого Л-процессора. Для примера на программах, обсуждавшихся при анализе характеристик ATNL-интерпретатора, быстродействие сконвертированных программ увеличилось в среднем в 1,5—2 раза.

Предварительные расчеты показывают, что реализация ATNL-компилятора позволит увеличить быстродействие еще в 2,5—3 раза.

Таким образом, скорость разбора фраз языков общения составит на ЭВМ БЭСМ-6 от 0,03 до 0,13 с/слово, а это уже цифры, близкие к скорости реакции человека.

Отсюда следует, что система представления на базе языка ATNL может быть с успехом использована на практике.

ГЛАВА ШЕСТАЯ

ДИАЛОГОВАЯ ИНФОРМАЦИОННО-ЛОГИЧЕСКАЯ СИСТЕМА (ДИЛОС)

6.1. Назначение и общая характеристика системы

Диалоговая информационно-логическая система относится к новому классу программных систем, предназначенных для посредничества между стандартным программным обеспечением ЭВМ и пользователями — специалистами в различных прикладных областях, для которых программирование не является обязательным атрибутом профессии. Система ДИЛОС, с одной стороны, обладает модельной базой данных (МБД), в которой специальными формальными средствами отображается информация об исследуемых предметных областях (ПО), с другой стороны — содержит описания прикладных программ и данных, используемых при решении различных практических задач. Сами программы и данные вместе с обслуживающими их трансляторами, загрузчиками, редакторами и другими традиционными сервисными подсистемами образуют вычислительную среду, которую мы называем *главной базой данных* (ГБД).

В основе идеологии системы ДИЛОС лежит специально разработанный формальный язык (*Ф-язык*), относительно которого необходимо отметить следующее:

а) как формальный аппарат для представления знаний о различных ПО Ф-язык устанавливает синтаксис описания Ф-объектов, заполняющих МБД;

б) как средство для внутреннего отображения смысла входных естественноязыковых запросов Ф-язык определяет синтаксис Ф-выражений, которые можно считать носителями смысловых фрагментов, извлекаемых из заданного на входе текста;

в) как инструмент для обработки Ф-объектов и интерпретации Ф-выражений Ф-язык характеризуется набором стандартных функций, которые обеспечивают взаимодействие процессоров* ДИЛОС и их общение с внешней по отношению к ДИЛОС вычислительной средой.

Информация от пользователя поступает на вход системы в виде печатного текста на естественном языке (русском, английском и др.). После предварительной морфологической обработки, осуществляемой препроцессором (ПреП), и соотнесения слов с заранее заданным словарем

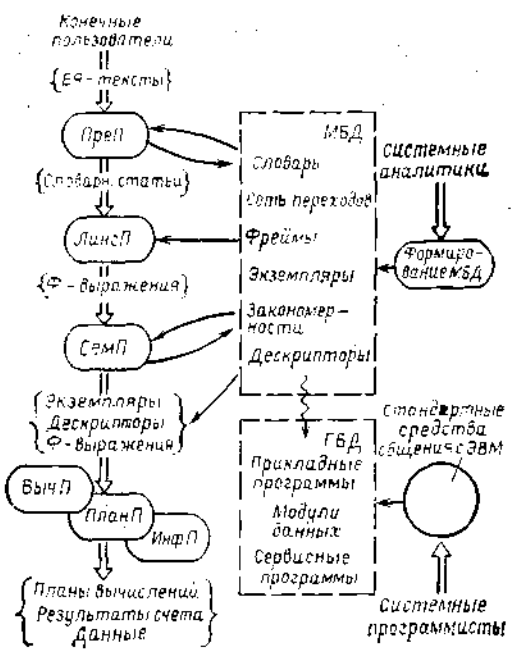


Рис. 6.1

* Система ДИЛОС представляет собой совокупность нескольких взаимодействующих программ, называемых процессорами.

лингвистический процессор выполняет перевод этого текста в *Ф-выражения* (рис. 6.1). Далее вступает в работу семантический процессор (СемП), который анализирует Ф-выражения с учетом имеющейся в МБД информации о данной ПО и выполняет соответствующие действия. В ходе семантической интерпретации могут происходить модификация и появление новых Ф-объектов, выдача пользователю различных значений, формирование планов вычислений и, наконец, запуск прикладных программ, а также обращения к стандартным базам данных для получения требуемых результатов. В этих операциях участвуют исполнительные процессоры системы — планирующий (ПланП), вычислительный (ВычП), информационный (ИнфП) (рис. 6.1).

6.2. Фреймы и Ф-объекты

В основе описываемого подхода к разработке языка представления знаний лежат идеи теории фреймов, которые получили в системе ДИЛОС реальное воплощение благодаря созданию формального аппарата Ф-языка и его реализации. Для каждой ПО вся релевантная информация, накапливаемая в МБД, может быть разделена между объектами следующих категорий:

фреймы (представляют собой специальные структурные описания используемых понятий и семантических отношений);

экземпляры (являются конкретизациями фреймов, отображающими индивидуальные объекты и факты реального мира, затрагиваемые в ходе человеко-машинного диалога);

закономерности (представляют собой специальные разновидности объектов, которые содержат процедурные знания, ассоциируемые с некоторыми понятиями и отношениями);

дескрипторы (служат для хранения информации о схемах вычислений, о прикладных программах и модулях данных, активизируемых во время решения практических задач).

Центральное место в модели любой ПО, представляемой в системе ДИЛОС, занимает совокупность фреймов, отображающих используемые понятия и отношения. Фрейм — это структурное описание, указывающее на участников соответствующего действия или события, на свойства и признаки класса объектов или понятия, на время, место и другие уместные обстоятельства описываемого фрагмента действительности. Во фреймах системы ДИЛОС

объединяются, таким образом, свойства ролевых фреймов и фреймов минимальных описаний, рассмотренных в гл. 2.

Имя фрейма *n* является уникальным мнемоническим кодом (обозначением) соответствующего понятия.

Видовая ссылка представляется специальным свойством с индикатором IS и служит указателем на *суперфрейм*, относящийся к тому же классу, что и фрейм *n*.

Базовые свойства являются описателями характерных черт соответствующего понятия. Индикаторы базовых свойств, которые будем обозначать $p_1, p_2, \ldots, p_k$, сопровождаются спецификаторами, которые указывают на имена других фреймов, а также на условия их конкретизации при формировании экземпляров данного фрейма. Эти условия выражаются с помощью так называемых *управляющих фильтров*, которые будут рассматриваться ниже.

Специальные свойства служат для указания специфических условий, влияющих на формирование экземпляров данного фрейма и на их связи с экземплярами других фреймов. Специальные свойства отличаются от базовых тем, что имеют стандартные (резервные) индикаторы. Обозначим их символами $s_1, \ldots, s_m$. Значениями специальных свойств могут быть имена закономерностей, запускаемых при обработке фрейма, индикаторы некоторых базовых свойств и др.

Общее число специальных свойств не регламентируется; в общем случае во время работы системы допускается приписывание (удаление) любых специальных свойств.

Итак, фрейм представляется в БЗ объектом вида:

[<имя-фрейма>:

IS <суперфрейм>
p_1 <спецификатор-1>
. . . . . .
p_k <спецификатор-K>
s_1 <значение спец-свойства-1>
s_m <значение спец-свойства-m>]

Ниже описываются способы оформления спецификаторов базовых свойств и управляющих фильтров.

1. Спецификаторы базовых свойств фрейма.

Спецификаторы базовых свойств (СпСв) могут служить нескольким целям:

указывать допустимые значения величин, конкретизирующих значения свойств при формировании экземпляров данного фрейма;

регламентировать условия и порядок заполнения (извлечения/удаления) значений свойств в экземплярах;

ассоциировать с соответствующими свойствами произвольные закономерности, которые могут обеспечить вторичные действия над содержимым базы знаний, а также реализовывать связи со стандартным программным обеспечением ЭВМ.

Синтаксически СпСв оформляются одним из двух способов:

в виде простых спецификаторов значений;

в виде составных спецификаторов, состоящих из управляющих фильтров и задающих более широкую характеристику каждого свойства.

Рассмотрим указанные варианты подробнее.

Спецификатор значения (СпЗн) характеризует допустимые варианты величин-значений данного свойства в экземплярах фрейма. СпЗн оформляется в виде одной из следующих конструкций.

а. Произвольное Ф-выражение, полный синтаксис которого будет рассмотрен ниже, указывает, что значением свойства может быть величина, вырабатываемая в результате семантической интерпретации данного Ф-выражения. Все остальные рассматриваемые ниже варианты СпЗн фактически являются частными случаями Ф-выражения общего вида или же могут быть приведены к нему простыми преобразованиями.

б. Ф-выражение вида (* *fn*), где *fn* — имя фрейма, указывает, что значением свойства может быть любой экземпляр фрейма *fn*. В частности, если *fn* — имя стандартного фрейма, например NUMber или STRing, то экземпляром такого фрейма должно быть число или строка.

в. Список вида (SETOF a), где a — произвольное Ф-выражение, указывает, что значением свойства может быть множество величин, вырабатываемых в результате интерпретации a.

г. Список вида (ANYOF $a_1 \ldots a_k$) указывает, что значением свойства может быть любая (одна) из величин, вырабатываемых в результате интерпретации Ф-выражений $a_1 \ldots a_k$.

д. Список вида (: $n_1 \ldots n_2$), где n_1, n_2 — целые числа, указывает, что значением свойства может быть любое число N, находящееся в диапазоне $n_1 \leqslant N \leqslant n_2$.

е. Список вида (, $v_1 \ldots v_k$), где $v_1, \ldots, v_k$ — произвольные константы (числа или строки), указывает, что значе-

нием свойства может быть любое подмножество $\{v_i\} \subset \{v_1, \ldots, v_k\}$.

ж. Список вида $(+(f\, a_1 \ldots a_k))$, где f — имя некоторой функции, $a_1 \ldots a_k$ — фактические значения ее аргументов, указывает, что значением свойства должна быть величина, получаемая в результате вычисления $f(a_1, \ldots, a_k)$;

з. Пустой список вида ( ) или NIL означает, что значение данного свойства не специфицировано, т. е. может быть произвольным.

Таким образом, СпЗн позволяют описывать величины, потенциально допустимые как значения свойств в экземплярах фреймов. Они используются как на этапе лингвистического преобразования ЕЯ-текстов в Ф-выражения, так и в ходе семантической интерпретации Ф-выражений.

Составной спецификатор (ССп) может включать в себя СпЗн, но, кроме того, позволяет вводить дополнительные указания об условиях и порядке формирования и манипулирования значениями данного свойства. Эти указания названы управляющими фильтрами свойств (УФСв). Составной спецификатор оформляется в виде списка

$$(=(s_1 w_1 \ldots s_k w_k)),$$

где $s_1 \ldots s_k$ — специальные индикаторы (имена управляющих фильтров), а $w_1 \ldots w_k$ — значения фильтров. Каждый фильтр $(s_i w_i)$ служит особой цели и подвергается соответствующей обработке при интерпретации Ф-выражений. Основные виды УФСв:

$VAL <спецификатор-значения>;
$DFL <спецификатор-значения-по-умолчанию>;
$FILL <спецификатор-значения-по-принуждению>;
$ASK <спецификатор-значения-по-запросу>;
$GETF <список-GET-закономерностей>;
$ADDF <список-ADD-закономерностей>.

Под индикатором $VAL задается СпЗн, рассмотренный выше. Он указывает, как обычно, какие величины могут претендовать на заполнение значения данного свойства при формировании экземпляров.

Под индикатором $DFL задается спецификатор значения по умолчанию (СпУм). Это могут быть любое Ф-выражение или константа (число или строка), вырабатывающие некоторое значение $\tilde{v}$. Вычисление СпУм производится в случае, когда в искомом экземпляре под индикатором данного свойства не обнаруживается никакого значения (значение равно NIL). Тогда при наличии во фрейме СпУм производится его вычисление и полученная величина $\tilde{v}$ трактуется как значение данного свойства в экземпляре.

Под индикатором $FILL задается спецификатор значения «по принуждению» (СпПр). Оформляется он точно так же, как СпУм, но вычисляется в тех случаях, когда при формировании нового экземпляра значение данного свойства не поступило из внешнего мира (из входного сообщения). Вычисленная по СпПр величина принудительно заносится в экземпляр как значение данного свойства.

Под индикатором $ ASK задается спецификатор значения «по запросу» (СпЗп). Его вычисление производится в тех же случаях, что и СпПр (фильтры $FILL и $ ASK взаимно исключают друг друга). Оформляется СпЗн как список пар $((m_1 txt_1) \ldots (m_k txt_k))$, где $m_1, \ldots, m_k$ — семафоры, управляющие режимами работы системы; $txt_1 \ldots txt_k$ — тексты вопросов, задаваемых пользователю. При вычислении СпЗн просматриваются все семафоры $m_1, \ldots, m_k$, и первый же из них $m_i \neq$ NIL вызывает печать текста вопроса txt_i на терминале. Ответ пользователя на заданный вопрос трактуется как величина, подставляемая в экземпляр в качестве значения свойства.

Под индикатором $GETF задается список имен «GET закономерностей», которые необходимо активизировать при попытке извлечения значения данного свойства из некоторого экземпляра (фактически это происходит до начала поиска соответствующих экземпляров). В процессе активизации закономерности могут происходить любые вторичные действия, оговоренные в теле этой закономерности, в том числе прерывание процесса поиска и извлечения свойства.

Под индикатором $ADDF указывается список имен «ADD-закономерностей», активизируемых при занесении нового значения данного свойства. Обработка ADD-закономерностей может иметь такой же побочный эффект, как и в случае GET-закономерностей.

Таким образом, УфСв обеспечивают не только контроль значений свойств, но и различные побочные действия, расширяющие возможности Ф-представления.

Пример 6.1. Рассмотрим в качестве примера несколько простых фреймов, соответствующих понятиям *изделие, изготавливать, автомобиль, двигатель* (рис. 6.2).

[Изделие: IS ENT
Изгот (* Завод)
Стоим (* NUM)]

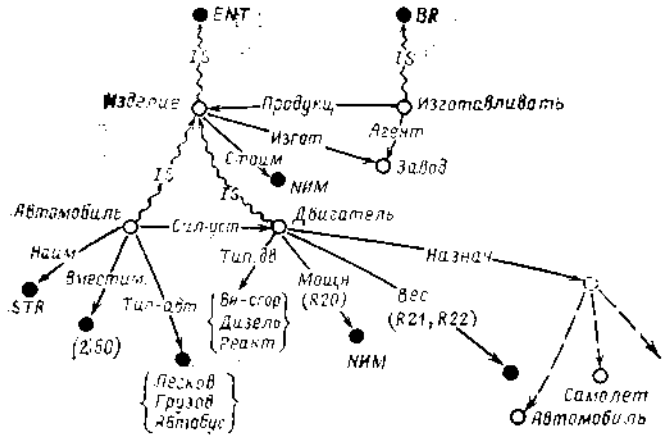


Рис. 6.2

```
[Изготавливать: IS BR
                Агент (* Завод)
                Продукц (* Изделие)]
 [Автомобиль: IS Изделие
                Наим (* STR)
                Сил-уст (* Двигатель)
                Тип-авт (, Легков Грузов Автобус)
                Вместим (: 2 50)]
  [Двигатель: IS Изделие
                Тип-дв (, Вн-сгор Дизель Реакт)
                Мощн (=( $VAL (* NUM)
                          $ GETF (R20)))
                Вес (=($GETF (R 21) $ ADDF (R22)))
                Назнач (ANYOF (* Автомобиль)
                              (* Самолет) ...)]
```

Рисунок 6.2 иллюстрирует фрагмент концептуальной семантической сети (КСС), соответствующий этим фреймам. Базовые узлы ENT (entity) и BR (basic relation), используемые в качестве суперпонятий, указывают на верхние уровни родо-видовой иерархии. Базовые узлы NUM (number) и STR (string) являются, наоборот, терминальными, характеризуя значения соответствующих свойств у фреймов *Изделие, Автомобиль, Двигатель*. Терминальными являются также узлы, помеченные спецификаторами вида «д» и «е». Заметим, что СпЗн вида «г», так же как и спецификации вида «в», порождают фиктивные узлы, промежуточные между некоторыми одиночными узлами (*Двигатель*) и наборами других узлов (*Автомобиль, Самолет...*). Управляющие фильтры, ссылающиеся на закономерности R20, R21, R22, «подвешиваются» к дугам *Мощн* и *Вес*, указывая, что получить значения этих

ойств можно лишь посредством закономерностей (вызывающих, возможно, счет по прикладным программам или обращения к стандартным базам данных).

2. Специальные свойства фреймов

Специальные свойства играют такую же роль по отношению к фрейму в целом, как УфСв по отношению к отдельным свойствам, т. е. они позволяют присоединять к фрейму специфическую информацию, оказывающую влияние на формирование и обработку его экземпляров. Охарактеризуем некоторые из специальных свойств.

1) $GETF <список GET-закономерностей>.

Это свойство аналогично GET-фильтру, используемому в ССп. Активизация GET-закономерностей происходит в данном случае при обработке любого Ф-выражения, предписывающего поиск экземпляров данного фрейма.

2) $ADDF <список ADD-закономерностей>.

С помощью этого свойства к фрейму присоединяются ADD-закономерности, активизируемые при формировании экземпляров данного фрейма.

3) $ DF <индикаторы-дискриминирующих-свойств>.

Значением свойства $DF является список индикаторов некоторых базовых свойств, комбинация значений которых является уникальной по отношению к другим экземплярам данного фрейма и, следовательно, позволяет устанавливать идентичность (или разницу) экземпляров. Например, у фрейма *Город* дискриминирующими могут быть свойства *Название* и *Страна*.

4) ATOMS <список имен экземпляров>.

Это свойство служит главным образом системным целям, указывая список имен всех экземпляров данного фрейма. В число специальных могут быть включены также свойства, указывающие на возможные причинно-следственные и временные отношения между экземплярами данного и других фреймов, на отношения часть-целое и др. Обработка всех таких свойств должна осуществляться специальными функциями в составе Ф-интерпретатора. В сущности набор специальных свойств, которые можно назвать управляющими фильтрами фреймов (УфФр), должен быть открытым, допуская возможность добавления новых фильтров с одновременным расширением обрабатывающих их функций Ф-интерпретатора.

6.3. Экземпляры фреймов

Ф-объект, представляющий экземпляр фрейма, имеет общий вид

$$[<\text{имя-экземпляра}> : p_1 v_1 \ \ldots \ p_k v_k],$$

где $p_1, \ldots, p_k$ — индикаторы; $v_1, \ldots, v_k$ — значения базовых свойств. Все экземпляры фрейма с именем n хранятся в Ф-разделе с именем n, что позволяет отличать их от экземпляров других фреймов.

Имя каждого экземпляра представляет собой мнемонический код (обозначение) некоторого физического объекта, факта, события и т. п. Этот код либо поступает из внешнего мира (из сообщения пользователя), либо генерируется самой системой.

Пример 6.2

Экземпляры фрейма „Автомобиль":

[А00300	:	Наим	ГАЗ-24
		Сил-уст	Д00010
		Тип-авт	Легков
		Вместим	5]
[А00320		Наим	ИКАРУС
		Сил-уст	Д00120
		Тип-авт	Автобус
		Вместим	35]

Экземпляры фрейма „Двигатель"

[Д00010	:	Тип-дв	Вн-сгор
		Мощн	95
		Назнач	АСС300]
[Д00120	:	Тип-дв	Дизель
		Назнач	А00320]

Здесь имена экземпляров представлены внутренними системными кодами. Значения свойств согласуются со спецификаторами, указанными в примере 6.1. Заметим, что отдельные свойства в экземплярах могут отсутствовать; это означает, что соответствующая информация еще не поступила из внешнего мира.

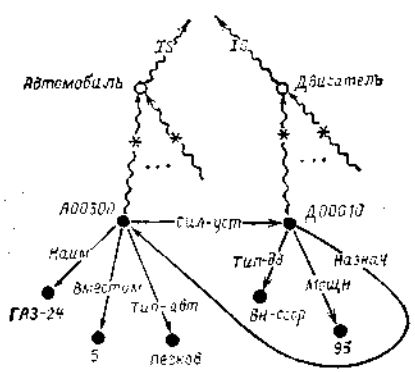


Рис. 6.3

Следует обратить внимание, что значениями многих свойств являются имена экземпляров других фреймов. Именно благодаря этому множество экземпляров отображает определенный связный компонент терминальной семантической сети (ТСС), представляющей собой конкретизации понятий КСС (рис. 6.3).

6.4. Закономерности

Возможность представления абстрактной схемы внешнего мира в виде КСС и его реальных фрагментов в виде ТСС позволяет описывать статические отношения между объектами и терминальными величинами. Однако этого недостаточно, если требуется описывать динамические аспекты, в частности условия формирования новых фрагментов ТСС, взаимосвязи между ТСС и содержимым стандартных баз данных, а также условия и порядок запуска прикладных программ, относящихся к данной ПО.

Как указывалось, к фреймам с помощью управляющих фильтров (УфСв и УфФр) могут присоединяться различные закономерности. ADD- и GET-фильтры управляют моментами активизации соответствующих закономерностей, а в них самих могут производиться анализ текущего состояния БЗ и выполнение различных побочных действий. Общий вид Ф-объекта, представляющего закономерность:

[<имя-закономерности>: * TH
PATT <образец>
FUN <программа>
BODY <список-альтернатив>]

При активизации закономерности с заданным именем прежде всего производится сопоставление ее образца с текущим содержанием МБД.

Сопоставление заключается в том, что заданное в образце Ф-выражение вычисляется и при этом вырабатывается некоторое значение v. Если v=NIL, то сопоставление считается неудачным и обработка закономерности прекращается с результатом NIL (неудача). Если $v \neq$ NIL, то сопоставление считается удачным и начинается обработка свойств {FUN <программа>} и {BODY <список альтернатив>}.

Элемент <программа> может содержать произвольную ЛИСП-программу, исполнение которой приводит к некоторому побочному эффекту.

Элемент <список-альтернатив> соответствует группе поименованных элементарных правил, а элементарное правило составляется в общем случае из трех компонентов:

CND <условие>
CNS <следствие>
ACT <задержанное-действие>

В условии обычно проверяется полное или частичное совпадение полученных при сопоставлении значений шаб-

лонных переменных с какими-либо величинами (константами). Часто условие опускается, и тогда по умолчанию считается, что оно удовлетворено. Следствие и задержанное действие обрабатываются лишь при удовлетворении условия.

В следствии могут быть заданы Ф-функции, которые с помощью Ф-выражений могут вызывать вторичные обращения к МБД, поиск, изменение и удаление любых Ф-объектов (в том числе фреймов и закономерностей). Таким образом, описанный механизм позволяет возложить на саму систему создание и модификацию ее базы знаний.

Задержанное действие также может содержать обращения к Ф-функциям, однако они не вычисляются, а запоминаются. Их вычисление откладывается до окончания всего процесса интерпретации исходных Ф-выражений. Выполнение всех задержанных и собранных вместе действий осуществляется лишь по специальной команде пользователя. Следовательно, задержанное действие создает побочный эффект особого рода — его результат может сказаться лишь по окончании всего процесса при условии, что пользователь даст соответствующую команду. Именно с помощью задержанных действий рекомендуется осуществлять связь со стандартными базами данных и пакетами прикладных программ.

Закономерности дают возможность выражать процедурные знания о ПО, связывая их с декларативными знаниями, выраженными во фреймах. С помощью закономерностей отображаются правила поведения, действия, зависящие от различных условий, транзитивные и рефлексивные операции над объектами МБД и др.

6.5. Ф-выражения

В «интеллектуальной» диалоговой системе решаются по крайней мере две задачи, моделирующие умственную деятельность человека:

1) отражение действительности в пределах некоторой ПО;

2) понимание входных сообщений на ЕЯ и их семантическая интерпретация.

Этому разделению соответствует представление информации в виде Ф-объектов, описывающих ПО, и Ф-выражений, интерпретируемых системой в целях создания, поиска и модификации Ф-объектов (в первую очередь экземпляров фреймов).

При этом совокупность фреймов отражает понятийно-категориальную структуру данного фрагмента действительности («что бывает»), а множество экземпляров, являющихся «конкретизациями» фреймов, соответствует конкретным ситуациям, наблюдаемым в рамках этой структуры («что есть на самом деле»).

Ф-выражения служат для представления смысла входных ЕЯ-сообщений, который распознается на лингвистическом этапе анализа сообщений и интерпретируется на постлингвистических стадиях обработки. Фреймы, экземпляры и Ф-выражения находятся в тесной взаимосвязи. При этом фреймы:

служат для описания экземпляров — всякий экземпляр может возникнуть только при наличии соответствующего фрейма;

определяют множество правильных Ф-выражений, т. е. множество «смыслов, допустимых в системе».

Таким образом, фреймы призваны играть активную роль как в процессе распознавания смысла, так и в процессе его интерпретации. В первом случае они означают «размерности в пространстве смыслов», так что входное сообщение на ЕЯ получает координаты вдоль этих размерностей; во втором случае фреймы способствуют разложению Ф-выражений, которое приводит, в частности, к тем или иным изменениям в мире экземпляров.

Фреймы — самая стабильная, постоянная часть системы; число их более или менее фиксировано и сравнительно невелико. Число экземпляров (реальностей действительности, находящихся в рассмотрении) может быть на порядок или несколько порядков выше. Ф-выражения наиболее недолговечны: они выявляются из входных сообщений, интерпретируются и исчезают. Их число потенциально бесконечно.

1. Синтаксис. Ф-выражения (ФВ) в зависимости от назначения и способа употребления можно разделить на три типа, которые будем обозначать символами α, β, γ. Возможность альтернативного употребления типов ФВ будем обозначать сочетаниями этих символов, например $<\alpha\beta>$ или $<\alpha\gamma>$. В рассматриваемых далее синтаксических правилах нетерминальные символы обозначаются греческими буквами (возможно, с индексами) или словами и словосочетаниями, заключенными в угловые скобки, а многоточие означает возможность повторения предыдущей конструкции.

$<$Ф-выражение$>::=\alpha|\beta|\gamma$

α::=(<указательный-сегмент> ... :<дескрипторный-сегмент> ...)
β::=(THE ρ O Fα) | (THE ρ OF SELF)
γ::=(NOTα) | (OR<αγ>...) | (AND<αγ>...)
<указательный-сегмент>::=<спецуказатель>
<спецификация>
<дескрипторный-сегмент>::=ρ<спецификация>
<спецуказатель>::=*|$|DET|QUANT
ρ::=<индикатор-базового-свойства>
<спецификация>::=<константа>|(+<обращение-к-функции>)|<шаблонная-переменная>|<Ф-выражение>
<константа>::=<число>|<идентификатор>|<строка>
<шаблонная-переменная>::=<свободная шабл. переменная>|<связанная шабл. переменная>
<свободная-шабл.-переменная>::=—Θ
<связанная-шабл.-переменная>::=+Θ
Θ::=<идентификатор>

2. Семантика. Вычисление Ф-выражений. Основной конструкцией Ф-языка является выражение α ($ФВ_\alpha$), которое входит так или иначе в выражения других типов — $ФВ_\beta$ и $ФВ_\gamma$. Всякое $ФВ_\alpha$ строится вокруг некоторого опорного фрейма f_0, и при интерпретации («вычислении») такого выражения в качестве значения вырабатывается имя (или список имен) экземпляров фрейма f_0. Полученные экземпляры будем называть в дальнейшем референтами данного $ФВ_\alpha$.

Значение $ФВ_\beta$ вырабатывается в результате вычисления, состоящего как бы из двух этапов. Сначала интерпретируется выражение, заданное в качестве четвертого элемента $ФВ_\beta$ (после служебного символа *OF*). В нем выделяется опорный фрейм f_0 и вырабатывается список соответствующих референтов. Этот список становится исходным для второго этапа вычисления, на котором происходит извлечение (формирование) значений свойств с индикатором ρ. Следовательно, результатом вычисления $ФВ_\beta$ является некоторое множество величин $\{a\}$, конкретизирующих значение свойства ρ у всех референтов данного $ФВ_\beta$. Фактически процесс интерпретации $ФВ_\beta$ может выполняться не в два этапа, а за один просмотр соответствующего Ф-раздела МБД.

Значением $ФВ_\gamma$ является список референтов, который получается вычислением Ф-выражений, входящих в $ФВ_\gamma$ с учетом логических указателей NOT, OR и AND. Указатель NOT обозначает, что полученные референты должны быть исключены из последующего рассмотрения. Указатель OR обозначает, что в последующем рассмотрении могут участвовать любые референты выражений $<\alpha\gamma_1>$ ... ... $<\alpha\gamma_t>$. Указатель AND обозначает необходимость обязательной обработки всех входящих в ФВ выражений $<\alpha\gamma_1>$ ... $<\alpha\gamma_t>$, прежде чем будут выполняться дальнейшие операции; значение в этом случае не существенно.

В качестве простейшего примера рассмотрим обработку следующего Ф-выражения, выражающего смысл фразы «Население Ленинграда составляет 4 млн. жителей»:

(* NUM $ (THE НАСЕЛ OF
(* ГОРОД : НАЗВ ЛЕНИНГРАД)) :
VAL 4.0)

Обработка этого выражения начинается с самого внутреннего простого выражения $ФВ_\alpha$:

(* ГОРОД : НАЗВ ЛЕНИНГРАД)

Сначала делается попытка найти экземпляр опорного фрейма ГОРОД, обладающий свойством {НАЗВ ЛЕНИНГРАД}, а если такого экземпляра не обнаруживается, то он формируется заново. В любом случае в качестве значения выдается системное имя этого экземпляра, например ГОР277, после чего начинает обрабатываться следующий уровень — выражение $ФВ_\beta$:

(THE НАСЕЛ OF ГОР277)

Если у экземпляра ГОР277 есть свойство НАСЕЛ, то обработка данного выражения выдаст значение этого свойства. Однако в нашем примере это значение игнорируется, так как на следующем уровне должно обрабатываться выражение

(* NUM $ (THE ...) : VAL 4.0)

При обработке указанного выражения со стандартным фреймом NUM-фактически вычисляется выражение вида:

(* ГОРОД $ ГОР277 : НАСЕЛ 4.0)

Вычисление такого Ф-выражения приводит к занесению в экземпляр ГОР277 свойства (НАСЕЛ 4.0), что и подразумевается в исходном запросе.

Таким образом, происходит рекурсивная раскрутка составного Ф-выражения с попеременным обращением к МБД в целях поиска или занесения (модификации) соответствующих экземпляров.

3. Указательные сегменты. Разделение на указательные и дескрипторные сегменты соответствует лингвистическому делению на тему и рему высказывания.

Указательные сегменты характеризуют опорный фрейм и статус рассматриваемых референтов (квантификация, новизна и т. п.), в то время как дескрипторные сегменты апеллируют к собственным характеристикам референтов, ссылаясь на отдельные базовые свойства и их значения. Для удобства чтения и записи указательные и дескрипторные сегменты отделяются друг от друга символом «:». Все сегменты составляются из пар:

<маркер> <спецификация>

При этом в указательных сегментах в качестве маркеров используются спецуказатели {*, $, DET, QUANT}, которым соответствуют специальные обрабатывающие процедуры. Рассмотрим основные виды указательных сегментов:

1) Сегмент с маркером * имеет в качестве спецификации имя опорного фрейма. Такой сегмент является обязательным элементом всякого $ФВ_\alpha$. Например, в простейшем выражении (* ГОРОД) символ ГОРОД является именем опорного фрейма.

2. Сегмент с маркером $ обычно имеет в качестве спецификации выражение типа β или γ, которое в этом случае называется описателем вхождения. Описатель вхождения указывает, что референты основного ФВ получаются в результате вычисления соответствующего $ФВ_\beta$ или $ФВ_\gamma$. Например, в выражении

(* ГОРОД $ (THE СТОЛИЦА OF (* ГОСУДАРСТВО)))

опорным фреймом является ГОРОД, а референтами — все экземпляры, имена которых получаются как значения свойства СТОЛИЦА у экземпляров фрейма ГОСУДАРСТВО. Описатель вхождения может быть задан буквальным указанием имени или списка имен экземпляров — референтов, однако такой способ задания может использоваться лишь в закономерностях.

3. Сегмент с маркером DET имеет в качестве спецификации один из специальных символов {QUE, NEW, OLD}, который является *детерминативом*, характеризующим статус референтов данного ФВ. Символ QUE указывает, что данное ФВ отображает определенный вопрос пользователя, касающийся референтов. Символ NEW указывает, что данное ФВ описывает новые, до сих пор не известные системе референты. Символ OLD указывает, что речь идет о старых, только что упоминавшихся референтах. Например, выражение (* ЧЕЛОВЕК DET OLD) может соответствовать фрагменту фразы «... этот человек ...» или «... он ...». В систему могут быть введены дополнительные

спецсимволы, имеющие соответствующую трактовку, вместе с интерпретирующими их процедурами.

4. Сегмент с маркером QUANT имеет в качестве спецификации один из специальных символов {SOME, ALL}, который является *квантификатором* данного ФВ. Символ SOME указывает, что речь идет лишь о некоторых референтах (возможно, об одном единственном), а символ ALL указывает, что речь идет обо всех референтах данного ФВ. Например, фразе *все заводы этого министерства* может соответствовать Ф-выражение:

(* ЗАВОД QUANT ALL
$(THE ПРЕДПРИЯТИЕ OF
(* МИНИСТЕРСТВО DET OLD)))

4. Дескрипторные сегменты. Дескрипторные сегменты в отличие от указательных имеют в качестве маркеров символы, являющиеся индикаторами базовых свойств опорного фрейма. Соответствующие им спецификации ссылаются на значения этих свойств у референтов, причем существуют две принципиальные возможности:

дескрипторный сегмент указывает на необходимость извлечения значения свойства с заданным индикатором ρ;

дескрипторный сегмент задает ограничения на значение базового свойства с индикатором ρ.

Первая возможность, используемая сравнительно редко, реализуется с помощью задания спецификации в виде свободной шаблонной переменной (СвбШп), представленной произвольным идентификатором θ с префиксом «=». Извлекаемая величина становится после этого значением одноименной связанной шаблонной переменной (СвзШп), которая представляется тем же идентификатором θ с префиксом «+». Например, при необходимости извлечь числа, указывающие население всех государств, можно применить выражение

(* ГОСУДАРСТВО QUANT ALL: НАСЕЛЕНИЕ $=X$)

Здесь использована СвбШп $=X$, которая после завершения операции превратится в одноименную СвзШп $+X$. Значением последней будет список чисел, являющихся значениями свойства НАСЕЛЕНИЕ у экземпляров фрейма ГОСУДАРСТВО. Позднее $+X$ можно использовать в любых выражениях, имея в виду, что вместо нее будет подставляться указанный список чисел.

Наложение ограничений на значения базовых свойств референтов реализуется тем, что при вычислении спецификаций вырабатываются величины или множества величин, которые сопоставляются со значениями базовых

свойств референтов. При этом возможны следующие варианты задания спецификаций:

1) спецификация вида <константа> непосредственно задает терминальную величину — число, идентификатор или строку, являющуюся значением свойства. Например, выражение (* ГОРОД : НАЗВ ЛЕНИНГРАД) может означать *город Ленинград.*

2) спецификация вида (+ <обращение к функции>) вызывает сначала вычисление указанной функции, после чего полученная величина трактуется как значение обрабатываемого свойства. Синтаксис обращения к функции определяется, вообще говоря, инструментальным языком программирования. В случае языка ЛИСП обращение к функции представляется списком $(f a_1 \ldots a_k)$, где f — имя функции, $a_1, \ldots, a_k$ — фактические аргументы.

3) спецификация вида <связанная шаблонная переменная> вызывает сначала взятие (вычисление) значения указанной СвзШп, после чего полученная величина трактуется как значение свойства.

4) спецификация в виде $ФВ_\alpha$ указывает, что значение обрабатываемого свойства совпадает[1] с именем одного из референтов этого $ФВ_\alpha$. Например, в выражении (* ФИРМА : ПРЕЗИДЕНТ (*ЧЕЛОВЕК ...)...) подразумевается, что в экземпляре опорного фрейма ФИРМА свойство ПРЕЗИДЕНТ должно иметь в качестве значения имя одного из экземпляров фрейма ЧЕЛОВЕК. Подобное ФВ может соответствовать фрагменту фразы «*фирма, президент которой ...*».

5) спецификация в виде $ФВ_\beta$ указывает, что значение обрабатываемого свойства совпадает с величиной, полученной в результате вычисления данного $ФВ_\beta$. Например, фраза *Цепь A соединяет выход блока B с входом блока C* может быть отображена в виде ФВ:

(* Соединение: Цепь *A*
Начало (THE ВЫХОД OF
(* БЛОК : НАЗВ *B*))
Конец (THE ВХОД OF (* БЛОК : НАЗВ *C*)))

6) спецификация в виде $ФВ_\gamma$ указывает, что значение обрабатываемого свойства должно соответствовать результату вычисления $ФВ_\gamma$. Наиболее естественна спецификация вида (НОТ α), исключающая из рассмотрения те экземпляры, у которых значение обрабатываемого свойства совпадает с результатом вычисления α-выражения. Пример: (* Автомобиль : Сил-уст (NOT (* ДВИГАТЕЛЬ : ТИП-ДВ РЕАКТ)))

У автомобилей не бывает реактивных двигателей.

Итак, общий вид $ФВ_\alpha$:

(* <имя-опорного-фрейма>
$ <описатель-вхождения>
DET <детерминатив>
QUANT <квантификатор> :
p_1 <спецификация-1> ...
p_k <спецификация-k>)

Правильные Ф-выражения должны подчиняться определенным соотношениям между опорными фреймами, которые выражают наследование свойств по оси родо-видовой подчиненности фреймов.

Основные правила формирования Ф-выражений следующие. Пусть f — опорный фрейм некоторого $ФВ_\alpha$. Тогда маркер p всякого дескрипторного сегмента в данном $ФВ_\alpha$ есть индикатор базового свойства фрейма f или какого-либо из его суперфреймов F_1 (рис. 6.4).

Пусть теперь p — такой маркер, f_p — опорный фрейм данного сегмента, а F_p — фрейм, специфицирующий свойство p в F_1. Тогда F_p должен быть суперфреймом для f_p.

Пусть далее исходное $ФВ_\alpha$ содержит описатель вхождения $ФВ_\beta$ вида (THE q OF $ФВ'_\alpha$). Тогда q есть индикатор базового свойства фрейма f_q или одного из его суперфреймов F_q, где f_q—опорный фрейм $ФВ'_\alpha$. Пусть F_2— фрейм, специфицирующий свойство q в F_q. Тогда F_2 должен быть суперфреймом для исходного фрейма f.

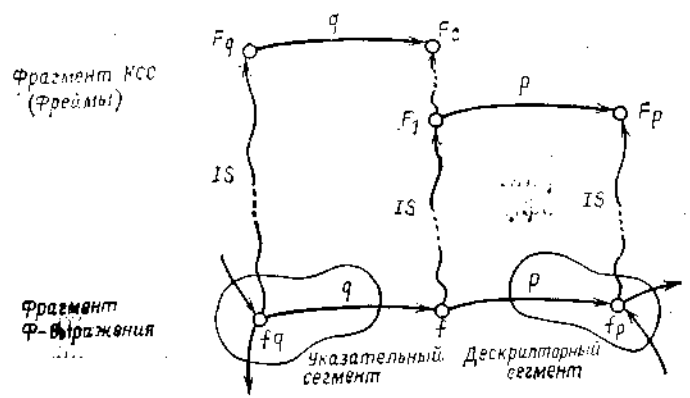


Рис. 6.4

[1] Во всех случаях, когда мы говорим о совпадении некоторой величины v со значением свойства, подразумевается либо полное совпадение, либо совпадение подмножеств (если v и/или значение свойства представляют собой множества).

Не следует, разумеется, рассматривать предлагаемый язык в качестве универсального языка смыслового представления. Совершенно очевидной является его неполнота в различных отношениях: многие «смыслы» не находят в нем удобного представления или же непредставимы вообще. Данный язык развивается главным образом благодаря обобщению опыта по представлению знаний в различных предметных областях. Важно при этом помнить, что в системе, которая подобно ДИЛОС призвана обслуживать реальных пользователей, «смыслы» должны не только «представляться», но и интерпретироваться. Вводя в Ф-выражения те или иные компоненты, нужно представлять себе, каким образом они могут быть обработаны, поняты системой. Сказанное означает, что данный подход к смысловым представлениям скорее прагматический, чем чисто теоретический. Ф-язык удобен для диалога о свойствах объектов и отношений, в то время как ряд других явлений выражается на Ф-языке с трудом.

Некоторые категории и понятия системы представления знаний занимают особое место и находят широкое отражение в естественных языках. К таковым нужно отнести категории «единичности-множественности», «противопоставления-сравнения», «пространства-времени» и др. По всей видимости, они должны играть специальную роль в системе представления знаний. Возможно, удастся использовать для их описания в целом тот же формальный аппарат, который предлагается здесь, дополнив его специальными средствами интерпретации.

Существенной особенностью предложенного языка является его зависимость от предварительно описанной системы фреймов. Это позволяет сосредоточиваться на описании фреймов как первичной задаче, а смысловое представление рассматривать как нечто производное. Столь же «фрейм-зависимым» является и процесс распознавания смысла входных ЕЯ-сообщений.

Эта ориентация на фреймы представляется весьма ценной для практических приложений, так как позволяет, в частности, эффективно настраивать систему на новую предметную область.

6.6. Лингвистический процессор и этапы анализа

В системе ДИЛОС целью работы лингвистического процессора (ЛингП) является перевод предложений с ЕЯ на внутренний язык представления — Ф-язык. Однако структуру ЛингП можно рассматривать и с точки зрения опыта создания универсального лингвистического процессора для преобразования текстов на ЕЯ в диалоговых системах различного типа. Это определяется тем, что при создании ЛингП системы ДИЛОС были реализованы следующие принципы:

универсальность должна обеспечить функционирование ЛингП в составе диалоговой системы общего назначения, а также потенциальное функционирование в различных системах;

минимальность должна привести к простому функционированию в простых случаях, при этом обработка ЕЯ-текстов может производиться с различной степенью сложности в зависимости от характера задачи;

прагматическая ориентация означает, что решающими для понимания ЕЯ-текстов являются знания не столько о языке как таковом, сколько об исследуемых фрагментах действительности.

Из этого, в частности, следует, что получение Ф-интерпретации входных ЕЯ-выражений определяется необходимостью извлечения лишь того смысла, который можно выразить на Ф-языке. Привлечение синтаксических процедур и других традиционных лингвистических методов анализа может происходить лишь по мере необходимости, а во многих (семантически очевидных) ситуациях такие процедуры вообще можно не привлекать к анализу.

Связано это с тем, что для понимания фразы человеку очень часто достаточно тех знаний о мире, которые он хранит в своей модели мира.

Функционирование ЛингП как универсальной подсистемы уместно характеризовать внешними параметрами. В первую очередь следует иметь в виду различные входные языки, которые определяются:

а) терминологией предметной области (T);

б) словарным запасом конкретного пользователя (L);

в) грамматическими конструкциями, характерными для данного языка (G);

г) формализмом представления знаний, т. е. синтаксисом Ф-языка (Φ);

д) семантикой (S), т. е. набором фреймов, участвующих в описании данной ПО.

Следовательно, совокупность внешних параметров

$$\{T, L, G, \Phi, S\}$$

определяет функционирование ЛингП.

ЛингП состоит из трех основных частей, обладающих разной подвижностью при настройке на разные внешние окружения:

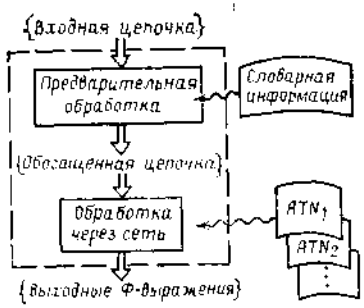


Рис. 6.5

ведущая программа, или Монитор;

словарь;

набор процедур (программ), структурированный в виде расширенной сети переходов и называемый в дальнейшем ATN-механизмом.

Назначение каждой из этих частей следующее.

Монитор является резидентной частью и управляет словарем и ATN-механизмом, которые являются сменными частями. В мониторе воплощены базовые механизмы процессора, не зависящие от внешних параметров ЛингП.

Словарь — наиболее подвижная часть ЛингП. Он может полностью изменяться; так, в случае смены языка общения (например, русского на английский) меняется весь его состав, а при настройке на новую ПО меняется его терминологическая часть, наконец, при фиксированных ЕЯ и ПО лексикон может варьироваться от пользователя к пользователю.

В ATN-механизме сосредоточены Ф-зависимые блоки ЛингП. Это позволяет настраивать ЛингП на синтаксически различные выходные языки.

На рис. 6.5 показана общая схема лингвистического этапа анализа. Настройка ЛингП на данную предметную область и тот или иной фиксированный ЕЯ происходит путем заполнения словаря и ATN-механизма.

Опишем теперь устройство словаря. Используемая лексика разделяется на два основных класса: содержательная (или терминологическая) и функциональная. Основное различие между ними следующее: слова-термины, входящие в содержательную лексику, имеют свои аналоги в виде понятий в модели мира и, следовательно, привязаны к данной ПО, в то время как функциональные слова таких аналогов не имеют и играют организующую роль в построении выходных Ф-выражений.

Каждое понятие исследуемой модели отражается некоторым словом или словосочетанием в языке пользователя и, как правило, выражается не более чем одним семом, т. е. элементом модели мира системы (называемым ниже *s*-кодом). Это означает, что слова с составными значениями содержат несколько семов и могут быть описаны через небольшое число простых значений.

Это позволяет описывать семантику содержательной лексемы в терминах свойств сопоставленного ей *s*-кода, а сами *s*-коды являются не чем иным, как элементами фреймов и экземпляров этих фреймов.

Словарь разбит на четыре взаимозависимых раздела: SEMVOC, MORPH, WORDS и CWVOC. Происхождение и характер хранимой информации в каждом из этих разделов обусловлены этапами и процедурами лингвистической обработки.

Раздел SEMVOC формируется в результате заблаговременного применения специальной процедуры преобразования ко всем фреймам, относящимся к данной ПО. Преобразование некоторого фрейма *F* в объекты SEMVOC заключается в том, что каждый атом (*s*-код), фигурирующий в F, снабжается набором специальных свойств — семантических атрибутов. Семантические атрибуты указывают на исходный фрейм F, семантическую роль (тип) данного кода в этом фрейме и т. п. Семантические атрибуты представлены для описания смысла слов; в частности, атрибуты объекта SEMVOC могут быть приписаны содержательному слову посредством ссылки на этот объект.

Формально семантические атрибуты отражаются в виде пар:

FR ⟨имя-исходного-фрейма⟩
TP ⟨семантический-тип⟩
CD ⟨внутренний-код⟩
SL ⟨имя-подчиненного-фрейма⟩
APP ⟨дополнительные-указания⟩
PH ⟨фрагмент-Ф-выражения⟩

Поясним назначение некоторых атрибутов. Элемент ⟨семантический-тип⟩ может приобретать одно из фиксированных значений, указывающих, что данный *s*-код (и соответствующее слово) занимает во фрейме F определенную позицию. Возможные типы и соответствующие им функции слов:

C — имя фрейма;
V — значение свойства;
I — наименование свойства;
R — наименование свойства-«отношения»;
Q — вопросительное слово;
LAST — ограничитель предложения;
X — игнорируемое слово.

Фактически из фреймов могут порождаться слова лишь первых четырех типов: C, V, I, R. Остальные типы (число их может изменяться по мере модификации системы) характеризуют функциональную лексику и не появ-

ляются в объектах SEMVOC. Функциональные слова вызывают некоторые действия внутри анализатора, влияющие на интерпретацию содержательной лексики. Они могут, например, менять предсказания, возникающие в ходе анализа, вызывать пересмотр уже построенных структур и т. д.

Элемент ⟨внутренний-код⟩ совпадает с обозначением имени фрейма, наименования или значения свойства. В разделе SEMVOC внутренние коды всегда совпадают с именами объектов. Однако, как будет показано ниже, при переносе семантических атрибутов из SEMVOC в другие разделы словаря возникает расхождение внутренних кодов и внешних начертаний слов. Тем самым устанавливается соответствие между внешним «миром слов» и внутренним «миром кодов».

Элемент ⟨имя-подчиненного-фрейма⟩ имеет смысл для *s*-кодов, обладающих типом R и играющих роль связки (бинарного отношения) между экземплярами исходного фрейма F и подчиненного ему фрейма $\tilde{F}$, который является спецификатором свойства в F. Например, *s*-код СИЛ-УСТ играет такую роль по отношению к фреймам АВТОМОБИЛЬ и ДВИГАТЕЛЬ (см. § 6.3). Семантические атрибуты этого слова будут такими:

FR АВТОМОБИЛЬ TP R CD СИЛ-УСТ SL ДВИГАТЕЛЬ

При практической реализации значения атрибутов FR и SL сворачиваются в одну точечную пару, так что в окончательном виде данный объект SEMOC будет иметь вид

СИЛ-УСТ (FR(ДВИГАТЕЛЬ. АВТОМОБИЛЬ) TP R CD СИЛ-УСТ))

Элемент ⟨дополнительные-указания⟩ чаще всего сопровождает слова типа V. Для таких слов с целью ускорения лингвистического анализа и построения правильных Ф-выражений указываются наименования (индикаторы) свойств, значениями которых могут быть данные слова. Так, *s*-код ДИЗЕЛЬ в выщерассмотренных примерах фигурирует как значение свойства ТИП-ДВ во фрейме ДВИГАТЕЛЬ, поэтому семантические атрибуты этого *s*-кода и соответствующего ему слова будут следующими:

FR АВТОМОБИЛЬ TP V CD ДИЗЕЛЬ APP (I ТИП-ДВ)

Элемент ⟨фрагмент-Ф-выражения⟩ содержит дополнения к формируемому Ф-выражению, обусловленные появлением во входной фразе данного слова. Этот атрибут в отличие от остальных никогда не фигурирует в объектах SEMVOC, но может появляться в других разделах словаря при определении новых слов. Например, слово

[illegible] может быть определено с помощью следующих **[illegible]**ических атрибутов:

FR ЧЕЛОВЕК PHI (ПОЛ МУЖ)

Таким образом, с помощью семантических атрибутов задается информация, сопутствующая каждому слову и помогающая строить правильные Ф-выражения. Общий вид словарной статьи SEMVOC: $(w(sa_1\ sv_1\ \ldots\ sa_k\ sv_k))$ (*), где w—*s*-код; $\{sa_1\ sv_1\ \ldots\ sa_k\ sv_k\}$ — набор семантических атрибутов, задаваемых именами sa_i и значениями sv_i.

В некоторых случаях один и тот же *s*-код может фигурировать в разных полях в разных фреймах F_1, F_2 ... Тогда формируется многозначная статья SEMVOC, имеющая вид

$$(w(=(sw_1)\,(sw_2)\ \ldots\ ))$$

Здесь sw_i — набор атрибутов, соответствующий фрейму F_i. Иногда удается слить наборы sw_1, sw_2, ..., и тогда словарная статья приобретает вид (*).

Раздел MORPH служит для хранения основ всех изменяемых слов. С каждой основой в MORPH связываются морфологическая модель (класс словоизменения) и набор семантических атрибутов. Объекты MORPH формируются в процессе определения пользователем слов, не известных системе.

Раздел WORDS содержит все известные системе словоформы. Существуют два вида словарных статей WORDS. Для неизменяемых слов словарная статья содержит специальный маркер «неизменяемости» и набор семантических атрибутов. Словоформы изменяемых слов представлены в WORDS статьями, содержащими основу, соответствующую словоформе, и альтернативные наборы грамматических характеристик, присущих данной словоформе. (При этом в MORPH должна существовать словарная статья, соответствующая основе. Предполагается, что каждой словоформе может соответствовать лишь одна основа).

Объекты раздела WORDS образуются в результате морфологического анализа новых словоформ, употребленных во входной фразе. Если выделенная при анализе основа $\tilde{w}$ обнаруживается в MORPH, то в словарную статью WORDS заносится ссылка на $\tilde{w}$ и полученные грамматические характеристики. При отсутствии $\tilde{w}$ в MORPH предварительно происходит формирование объекта MORPH.

Разделение на MORPH и WORDS в отдельных случаях замедляет процедуру просмотра словарей, а также

увеличивает их объем, однако позволяет для каждой словоформы производить морфологический анализ 1 раз — при ее первом употреблении пользователем.

Раздел CWVOC содержит специальные системные коды, сопоставляемые стереотипным группам слов (словосочетаниям). Словосочетания определяются в результате обращения к специальной процедуре (вне рабочих пользовательских сеансов). Эта процедура, во-первых, помечает в разделах WORDS и/или MORPH все слова, участвующие в словосочетании; при этом каждое участвующее в сочетании слово снабжается атрибутом вида CW ⟨код-словосочетания⟩. Во-вторых, в раздел CWVOC заносится объект вида

(⟨код-словосочетания⟩ (LIST *lw* SEM *sw*)),

где *lw* — список слов-компонентов словосочетания; *sw* — набор семантических атрибутов, приписываемых данному словосочетанию точно так же, как и одиночным словам.

Помимо элементарного значения, выражаемого набором семантических атрибутов, слову или словосочетанию в словаре может быть сопоставлено составное значение, выражаемое последовательностью элементарных значений (например, «старше»=«возраст»+«больше»), а также набор альтернативных значений для слов-омонимов. В мониторе обеспечиваются последовательная обработка составного значения и выбора одного из нескольких альтернативных значений.

Чтобы зафиксировать функциональную зависимость ЛингП от Ф-языка, все Ф-зависимые блоки выделяются в отдельный (сменный) модуль. Этот модуль построен по принципу расширенной сети переходов (ATN).

В отличие от ранних методов использования ATN в качестве синтаксических анализаторов ЕЯ в данном случае ЛингП ориентирован на построение Ф-выражений, а не синтаксических структур. ATN в системе ДИЛОС представляет собой набор отдельных блоков (состояний). Состояние описывается парами (⟨тип⟩, ⟨*prog*⟩), где ⟨тип⟩ — один из *s*- или *f*-типов, а ⟨*prog*⟩ — программа на языке ЛИСП. Каждая такая пара описывает переход из данного состояния в некоторое другое (в частности, снова в прежнее) состояние.

Работа программы связана с проверкой условий (значений переменных) и выполнением некоторых действий, как правило, изменяющих значения переменных, а также генерирующих фрагменты выходных Ф-выражений. Элемент ⟨тип⟩, с которым связывается каждый ⟨*prog*⟩, служит для проверки условия ⟨тип текущего слова⟩ ?=⟨тип⟩, что и приводит к выбору определенного ⟨*prog*⟩.

Легко заметить, что благодаря такой структуре ATN с каждым словом из входной цепочки может быть связана (через один из его типов) определенная программа. Движение вдоль входного предложения вызывает вычисление соответствующей последовательности программ и приводит к построению результирующего выражения Ф-языка.

Одна из переменных монитора — STATE хранит имя текущего состояния. Для переходов в новое состояние программа должна содержать действие, изменяющее значение этой переменной.

Управление сетью осуществляет монитор. Он подключает к обработке сети базовые процедуры. При этом сеть сообщает монитору информацию, которая влияет на его функционирование. В частности, каждый ⟨*prog*⟩ сети вырабатывает так называемую точку возврата, которая указывает монитору на особенности трактовки текущего слова. В реализованной версии ЛингП существуют четыре возможные точки возврата: JUMP (взять следующее слово), MOVE (изменить состояние сети, не меняя слова), FINIS («окончить обработку») и UPSET (вернуться к обработке альтернативной ветви или закончить работу).

Каждая сеть содержит собственное описание, куда входит информация, необходимая для активизации ее монитором, а именно:

имя начального состояния;

набор используемых переменных;

набор допустимых *типов*, необходимых для организации иерархии сетей;

описание используемых в сети процедур, т. е. определяющих выражений ЛИСП-функций; для этого фактически задается имя файла, где хранится список определяющих выражений функций.

Введение в ATN возможности описывать переменные и процедуры обеспечивает обычные программистские средства для эффективного конструирования сетей.

Дадим формальное определение ATN в ЛингП:

ATN::=(⟨описание⟩ ⟨последовательность-состояний⟩)

⟨описание⟩::=(DSCR(INIT⟨имя-начального-состояния⟩
PROG ⟨список-переменных⟩
TYPES ⟨список допустимых типов⟩
AUXFNS ⟨имя файла с описаниями функций⟩))

⟨состояние⟩::=⟨имя состояния⟩ ⟨список переходов⟩

$$\langle\text{переход}\rangle ::= \left\{\begin{array}{l} s\text{-тип} \\ f\text{-тип} \\ \text{test} \end{array}\right\} \langle\text{тело перехода}\rangle$$

$$\langle\text{тело перехода}\rangle ::= \left\{\begin{array}{l} \langle\text{точка возврата}\rangle \\ (+\langle s\text{-выражение}\rangle) \end{array}\right\}$$

⟨точка возврата⟩::=JUMP|MOVE|UPSET|FINIS

Отметим, что в ЛингП имеется возможность рекурсивного вызова сетей.

При восприятии очередной входной фразы лингвистический процессор прежде всего устанавливает, является ли она обычной фразой рабочего сеанса или предназначена для запуска некоторого формирующего цикла (см. § 6.8).

Рабочая фраза вначале обрабатывается препроцессором (ПреП), который в результате процедур морфологического анализа и свертки словосочетаний заменяет все слова фразы соответствующими словарными статьями. Если при этом некоторые (или даже все) слова отсутствуют в WORDS и их не удается сопоставить известным основам или словосочетаниям, то происходит автоматический выход на формирующий цикл для определения каждого из незнакомых слов (см. § 6.8).

Выходным результатом этапа предварительной обработки является цепочка слов входного предложения, обогащенных словарной информацией. Свернутые словосочетания представлены в этой цепочке наравне с автономными словами, и цепочка однородна в том смысле, что всем ее элементам приписана однотипная информация. В простейшем случае (при отсутствии морфологического анализа и словосочетаний во входной фразе) предварительная обработка сводится к извлечению информации из словаря.

Базовые процедуры на этапе основного лингвистического анализа следующие:

а) активизация одной из ATN-сетей;
б) выбор очередного слова из входной цепочки;
в) анализ его значения, предусматривающий:
  в1) сопоставление значения атрибута *FR* с текущим фреймом, установленным предшествующими словами предложения;
  в2) обработку омонимов;
  в3) обработку составных значений;
  в4) обработку незнакомых слов;
  в5) проверку окончания анализа;
г) проведение выбранного значения через сеть;
д) выполнение действий, соответствующих полученной точке возврата;
е) передача выработанного в сети значения в точку, откуда произошел вызов сети.

Процедура в1, следящая за семантической связностью входного текста, является одной из основных в ЛингП и учитывает, в частности, наследование свойств в иерархии фреймов, а также взаимосвязи в сети фреймов, представляющей модель текущей ПО. Это дает основание называть лингвистический анализ фрейм-управляемым. Результат работы ЛингП — правильные Ф-выражения — поступает на вход семантического процессора для интерпретации.

Правильные Ф-выражения являются входной информацией для семантического процессора (СемП), который манипулирует Ф-разделами, осуществляет поиск референтов Ф-выражения, обеспечивает активизацию закономерностей и все другие действия, связанные с семантической интерпретацией (вычислением) Ф-выражения. Работа СемП отображается изменениями в мире экземпляров, выдачей необходимой информации на терминал пользователя, формированием задержанных действий, подлежащих дальнейшему исполнению.

Результаты текущей работы пользователя с системой (множество Ф-разделов с фреймами, экземплярами и закономерностями) накапливаются в специальной области МБД, названной TOPIC. Существование TOPIC, с одной стороны, помогает анализировать связные тексты из многих предложений, с другой стороны, обеспечивает хранение промежуточных результатов с целью их повторного анализа, отладки, «возвратов назад» и др.

6.7. Другие процессоры системы

Во многих случаях несколько следующих подряд циклов общения пользователя с системой могут вовлекать в работу лишь три указанных процессора: ПреП, ЛингП, СемП (см. рис. 6.1). В то же время многие прикладные задачи требуют обращения к стандартному программному обеспечению, сосредоточенному в ГБД. Прямое обращение к содержащимся там прикладным программам и базам данных, по-видимому, возможно лишь через общую операционную систему, что неудобно для пользователя систе-

мы ДИЛОС (требует от него изучения языка управления заданиями, знания мест расположения разных объектов и др.). В связи с этим в ДИЛОС вводится открытый набор исполнительных процессоров, обеспечивающих автоматизацию связи между МБД и ГБД.

Один из таких процессоров назван планирующим (ПланП). Он служит для формирования планов вычислений, в которые входят обращения к элементам ГБД. Обращение к ПланП может происходить из ПреП или СемП. В процессе работы ПланП может использовать экземпляры и задержанные действия, накопленные в TOPIC, и, кроме того, поддерживает активный диалог с пользователем, составляющим конкретный план вычислений.

Другой исполнительный процессор назван вычислительным (ВычП), и его роль состоит в реализации (исполнении) планов вычислений, составленных с помощью ПланП. Оба процессора — ПланП и ВычП — активно используют специальные виды Ф-объектов, называемые дескрипторами. Существуют дескрипторы прикладных программ (ДПП) и дескрипторы модулей данных (ДМД), указывающие с помощью своих свойств местоположение программ и данных в ГБД, условиях их запуска и т. п. Поскольку дескрипторы представлены как обычные Ф-объекты, их можно создавать и модифицировать через Преп-ЛингП-СемП, т. е. путем общения с системой на ЕЯ.

Наконец, еще один исполнительный процессор назван информационным (ИнфП). Его назначение — эффективная связь со стандартными базами данных или со специально организованной системой управления данными. При этом входной информацией для ИнфП является упрощенное Ф-выражение типа α (без вложенных ФВ). Такое выражение может передаваться в ИнфП от СемП, когда последний выясняет (по специальному свойству в опорном фрейме), что искомые объекты находятся не в МБД, а в ГБД.

Таким образом, ПланП, ВычП и ИнфП тесно взаимодействуют с тремя основными процессорами, воспринимая информацию от них в виде специальных управляющих команд, Ф-объектов и Ф-выражений.

Адаптация системы ДИЛОС к конкретной ПО требует довольно кропотливой работы по созданию и вводу в МБД необходимых фреймов, закономерностей, экземпляров, словарных статей и дескрипторов. Эту работу обычно осуществляют системные аналитики, которые должны хорошо знать и исследуемую ПО, и особенности системы

ДИЛОС. Чтобы облегчить и частично автоматизировать их работу, в систему введен формирующий процессор (ФормП), который в режиме диалога с системным аналитиком формирует необходимые Ф-объекты. При этом часть информации система запрашивает у человека с довольно большими подробностями (например, перечень и спецификации базовых свойств фреймов), а другая часть формируется автоматически (например, словарные статьи). В любом случае с человека снимается необходимость соблюдения строгих соглашений по оформлению Ф-объектов, Ф-разделов и т. п.

По мнению разработчиков системы ДИЛОС, этап формирования начального состояния МБД имеет большое значение для успешного использования системы в практической ПО. Поэтому намечено дальнейшее развитие ФормП и расширение его функций.

6.8. Работа пользователя с системой

Сеанс связи пользователя с системой начинается с идентификации пользователя, запроса имени его области данных и режимов работы (если это первый вход в систему данного пользователя) и настройки на соответствующую область и режимы. Основные режимы:

CARE — печать промежуточных Ф-выражений;
PRTIME — печать времени в основных точках анализа;
DMNSTR — блокировка всех промежуточных выдач.

Во время сеанса связи пользователь может выполнять несколько видов операций, каждому из которых соответствует характерный цикл. Основные виды циклов:

А) опрос и модификация модели предметной области;
Б) запросы, связанные с запуском прикладных программ;
В) запросы, связанные с обращением к стандартным базам данных;
Г) формирование понятий;
Д) формирование словарей;
Е) формирование закономерностей;
Ж) формирование схем вычислений.

Циклы А—В являются рабочими, так как они соответствуют обычным рабочим потребностям конечных пользователей.

Циклы Г—Ж будем называть формирующими, так как они служат для заполнения и модификации различных разделов модельной базы данных, причем осуществляют эту работу системные аналитики.

Рабочий цикл, как правило, состоит из следующих этапов:

восприятие набранного пользователем текста и анализ каждого слова в соответствии с содержимым словарей;

лингвистический анализ и формирование Ф-выражений, отображающих смысл входной фразы;

семантическая интерпретация полученных Ф-выражений и обработка связанных с ними закономеркостей;

исполнение отдельных прикладных программ или обращение к стандартным базам данных (внешним по отношению к системе ДИЛОС).

В циклах вида А последний этап, как правило, отсутствует. При безошибочной работе система сигнализирует о прохождении каждого этапа печатью на терминале специального сигнала. Кроме того, в режиме CARE система запрашивает у пользователя подтверждение на переход к следующему этапу. При утвердительных ответах пользователя ДА, YES, T работа продолжается. Отрицательные ответы НЕТ, NO заставляют систему завершить данный цикл и начать следующий.

Формирующий цикл запускается в результате того, что пользователь вводит фразу, начинающуюся с одного из ключевых слов:

NEWCONCEPT — формирование нового понятия и связанных с ним закономерностей;

СОЧЕТАНИЕ — формирование словосочетания;

FORGET — вычеркивание указанных слов со всеми словоформами из словарей;

MODIFY — изменение понятия или закономерности;

NEWSCHEMA — формирование новой схемы вычислений.

Каждый формирующий цикл состоит из характерных этапов, и переход от одного этапа к другому происходит под влиянием специального диалога системы с пользователем, в котором ведущую роль играет система.

1. *Формирование новых понятий и закономерностей.* Цикл формирования новых понятий и закономерностей начинается, когда пользователь вводит выражение вида

NEWCONCEPT <понятие>.

При этом система запрашивает у пользователя необходимую информацию для формирования Ф-объекта, представляющего фрейм: имя суперпонятия, имена и спецификаторы базовых свойств, а также дополнительную информацию (управляющие фильтры и др.). При вводе спецификаторов свойств пользователь употребляет стандартные имена (NUM, STR, BOOL, SETOF, ANYOF), имена понятий, уже известных системе или подлежащих определению. Имена упомянутых понятий и закономерностей запоминаются системой, и впоследствии происходит уведомление пользователя о необходимости ввода недостающих определений. Введенные фреймы и закономерности запоминаются в соответствующих файлах модельной базы данных.

2. *Формирование словарей.* Для работы лингвистического процессора в системе поддерживаются четыре словаря, представленных в МБД файлами SEMVOC, WORDS, MORPH, CWVOC. Пользователям удобно формировать их изнутри системы ДИЛОС, освобождаясь от необходимости соблюдения точного синтаксиса, знания различных тонкостей и т. п.

Файл SEMVOC формируется автоматически при вводе новых понятий. Он содержит основную информацию о содержательной лексике, т. е. мнемонические коды всех имен понятий, свойств и их значений, а также указания о роли каждого кода при переходе от фраз естественного языка к выражениям Ф-языка. Объекты SEMVOC служат в первую очередь для определения всех прочих слов, употребляемых пользователем.

Цикл формирования словарей WORDS и MORPH начинается каждый раз, когда во входной фразе, переданной пользователем, встречаются не известные системе слова (в принципе вся фраза может состоять из неизвестных слов). Цикл состоит из двух главных этапов: 1) попытки «понять» новое слово после его морфологического анализа и сопоставления выделенной основы с имеющимися основами; 2) определения нового слова путем специального диалога с пользователем.

При удачном завершении 1-го этапа происходит нормальное продолжение работы; при неудаче начинается 2-й этап, который предусматривает ответы пользователя на вопросы о морфологической основе парадигме (типе словоизменения) данного слова. При этом человеку приходится апеллировать к собственным знаниям русского языка. Ошибки не страшны и в худшем случае ведут к появлению избыточной информации в словарях и «непониманию» других слов с теми же основами.

Семантика слова определяется при ответе на вопрос о синониме или мнемоническом коде. Здесь возможны несколько вариантов ответов:

а) *X* — указание, что данное слово следует игнорировать при анализе входных фраз;

б) <мнем-код> — слово из SEMVOC, имеющее тот же смысл, что и данное слово;

в) <слово> — любое слово из файла WORDS, играющее при анализе ту же роль, что и данное слово;

г) <понятие> <имя-экземпляра> — указание о том, что данное слово выступает в качестве значения свойства NAME в одном из экземпляров указанного <понятия>;

д) <понятие> WITH <свойство> <значение> ... — указание о том, что данное слово соответствует таким экземплярам <понятия>, у которых заданные <свойства> имеют заданные <значения>;

е) CD <мнем-ход> TP <тип> FR <фрейм> SL <спецификатор> — таким способом вводятся специфические указания о <мнем-коде> и <типе> данного слова, а также о том, в формировании какого <фрейма> это слово может участвовать и каков <спецификатор> свойства, характеризуемого данным словом;

ж) НЕТ или NO — отказ от определения слова ввиду отсутствия у пользователя достаточной уверенности.

Файл CWVOC хранит словосочетания. Каждое словосочетание может быть образовано из двух или более слов, входящих в файл WORDS, и снабжено семантикой так же, как и отдельное слово.

Цикл определения словосочетаний начинается, если пользователь вместо входной фразы введет выражение

СОЧЕТАНИЕ <слово 1> ... <слово k>.

Используемые в сочетании слова могут уже содержаться в словарях или подлежать определению с помощью описанной выше процедуры. Завершается данный цикл запросом синонима или мнемонического кода для всего словосочетания. Варианты ответов пользователя рассмотрены выше.

3. Формирование схемы вычислений. Работа планирующего процессора, осуществляющего формирование схемы вычислительного процесса (СВП), начинается с запроса у пользователя имен файлов, содержащих дескрипторы программ и модулей данных, участвующих в процессе. Затем начинается собственно планирование, в ходе которого строятся фрагменты СВП. При этом система задает вопросы, касающиеся отдельных операторов СВП, и, получая ответы пользователя, строит соответствующие выражения. Построение очередной СВП завершается занесением в нее оператора RETURN. Построенный план вычислений предъявляется пользователю, а затем записывается в рабочую область под именем, указанным пользователем.

Операторы, из которых формируются СВП, можно разбить на пять основных групп:

1) обмен с терминалом (PRT, ASK);

2) обмен с базами данных (STORE, ASSIGN);

3) запуск прикладных программ (RUN, RUN+, MACRO);

4) управление ходом вычислений (IF, WHILE, GOMPUTE, STOP, RETURN);

5) формирование значений переменных (SET, PLUS).

Из указанных операторов можно формировать произвольно структурированные СВП, при использовании которых происходит обмен информацией между пользовательским терминалом, базами данных и прикладными программами.

Сформированные СВП накапливаются в МБД и в дальнейшем используются вычислительным процессором для фактического управления вычислениями.

КОММЕНТАРИЙ И БИБЛИОГРАФИЯ

Ниже для каждой главы книги укажем литературу, которая либо была использована при написании данной книги, либо освещает затронутые в ней проблемы с иной стороны. Библиография сопровождается краткими комментариями, цель которых помочь читателям сориентироваться в имеющейся литературе в области диалоговых систем, базирующихся на естественном языке. Другой целью комментариев является указание на трудные, спорные и нерешенные вопросы, связанные с рассматриваемой областью. В библиографии приведены только те книги и статьи, которые опубликованы на русском языке. Оправданием этого может служить наше глубокое убеждение, что в книге, рассчитанной на первое знакомство специалистов с новой для них областью, отечественных публикаций хватает с избытком. А для тех, кто хочет познакомиться с иностранными источниками по диалоговым системам, базирующимся на естественном языке, можем рекомендовать воспользоваться библиографией, приведенной в указанных отечественных книгах. Текущий уровень успехов в этой области можно прослеживать, знакомясь с трудами международных конференций по искусственному интеллекту, регулярно проходящих раз в 2 года.

Глава первая

При работе на ЭВМ в диалоговом режиме на языке, близком к естественному, возникают проблемы двух видов. Во-первых, это проблемы собственно языковые, связанные с изучением самого языка, его грамматики и семантики, способов передачи смысла за счет синтаксических конструкций языка, его лексики и морфологии. В этих исследованиях сам язык выступает как объект нашего анализа. Во-вторых, это проблемы, связанные с отображением мира действительности в памяти ЭВМ и соотнесением языковых текстов с этими моделями действительности. Последний процесс по своей сути есть процесс понимания текста ЭВМ. Создание диалоговых систем, которое на первом этапе казалось лингвистической проблемой, на самом деле есть проблема более широкая — семиотическая, если под семиотическими моделями понимать любые структурно-организованные модели действительности. В работе [1] эта концепция разделения исследований на два типа — модельно-языковые собственно лингвистические и исследования в области создания действующих диалоговых систем обоснована весьма убедительно. А опыт создания первых диалоговых систем, базирующихся на проблемно-ориентированных подмножествах естественного языка, отраженный, в частности, и в этой книге, подтверждает наличие такого разделения исследований. В работе [2] исследуется феномен, связанный с появлением диалоговых систем, ориентированных не на весь естественный язык в целом, а на его проблемно-ориентированные подмножества. В этой работе, пожалуй, впервые четко сформулированы основные концепции, на которые явно или неявно всегда опирается специалист, работающий в области создания диалоговых систем.

Первый этап работ в области машинного перевода отражен в ра-

ботах [3, 4]. Современное понимание этой проблемы хорошо иллюстрируется работой [5], из которой заимствованы многие идеи и примеры, изложенные в § 1.2. Анализ перевода басни Лафонтена Крыловым и еще несколько примеров влияния синтаксиса на семантику содержатся в монографии [6]. Рассуждения о требованиях к языку смыслов сформировались у авторов на основании высказываний создателей лингвистической модели «Смысл — Текст» [7—9]. Пример с превращением фразы: «Идея яростно спит» в осмысленную заимствован из [10]. В работе [11] высказаны некоторые соображения, оказавшие влияние на авторскую интерпретацию термина *понимание*. В работах [12—14] обсуждаются те трудности, которые возникают при попытках выявить семантическую структуру текстов на естественных языках, а в работах [15—19] перечисляется еще ряд трудностей, оказывающих существенное влияние на сложность процедур универсального лингвистического процессора. Исчерпывающий анализ влияния синтаксиса абстрактных фраз на понимание их людьми дан в цикле экспериментов, проведенных В. А. Шороховым. Им, в частности, показано, что для фразы Щербы: «Глокая куздра штеко бодланула бокра и курдячит бокренка» приписываемая этой фразе семантика, по-видимому, кроется в аналогии слова «бодланула» со словом «боднула». Именно это заставляет людей видеть в этой абстрактной фразе синтаксическую конструкцию типа: «Злая корова сильно боднула быка и гонит теленка», а, скажем, не конструкцию типа: «Слушая брата, крепко психанула сестра и молчит девчонка». Системы, в основе которых лежит идея спецификационных списков, весьма многочисленны. По-видимому, впервые эта идея была реализована в системе, описанной в [20]. Дескрипторные и тезаурусные системы описывались многократно [21—24]. Некоторые другие интересные аспекты, связанные с материалом гл. 1, читатели могут найти в публикациях [25—31].

1. **Нариньяни А. С.** Формальная модель языка: общая схема и выбор адекватных средств. — Препринт ВЦ СО АН СССР, вып. 1, № 107, Новосибирск, 1978. — 29 с.

2. **Ершов А. П.** К методологии построения диалоговых систем: феномен деловой прозы. — Препринт ВЦ СО АН СССР, № 156, Новосибирск, 1979. — 24 с.

3. **Теоретические** основы машинного перевода на русский язык/ О. С. Кулагина, А. А. Ляпунов, И. А. Мельчук, Т. М. Молошная. — В кн.: Исследования по славянскому языкознанию. — М.: 1961, с. 374—382.

4. **Машинный перевод.** — Сборник статей. — М.: Изд-во иностр. лит., 1957 — 314 с.

5. **Апресян Ю. Д.** Лингвистическое обеспечение в системе автоматического перевода третьего поколения. — Изд. Научного Совета по комплексной проблеме «Кибернетика» при Президиуме АН СССР, 1978. — 47 с.

6. **Выготский Л. С.** Мышление и речь. — М. — Л.: Соцэкгиз, 1934. — 324 с.

7. **Апресян Ю. Д.** Экспериментальное исследование семантики русского глагола. — М.: Наука, 1967. — 251 с.

8. **Мельчук И. А.** Опыт теории лингвистических моделей «Смысл — Текст». — М.: Наука, 1974. — 314 с.

9. **Апресян Ю. Д.** Лексическая семантика. Синонимические средства языка. — М.: Наука, 1974. — 366 с.

10. **Ревзин И. И.** Некоторые трудности при построении семантических моделей для естественных языков. — В кн.: Симпозиум по структурному изучению знаковых систем. — М.: Изд-во АН СССР, 1962, с. 17—24.

11. **Мартемьянов Ю. С.** Заметки о строении ситуации и форме ее описания. — В кн.: Машинный перевод и прикладная лингвистика. Вып. 8, М.: МГПИЯ, 1964, с. 125—148.

12. **Севбо И. П.** Структура связного текста и автоматизация реферирования. — М.: Наука, 1969. — 135 с.

13. **Берзон В. Е., Зубов А. В.** О семантической классификации параметров связности текста. — В кн.: Романское и германское языкознание. Вып. 1. — Минск: Минский государственный педагогический ин-т иностранных языков, 1977, с. 185—197.

14. **Пробст М. А.** О методах исследования текста, не расчлененного на слова. — Научно-техническая информация, сер. 2, изд. ВИНИТИ АН СССР, № 11, 1965.

15. **Корельская Т. Д.** О формальном описании синтаксической синонимии. — М.: Наука, 1975. — 251 с.

16. **Ревзин И. И.** Современная структурная лингвистика: Проблемы и методы. — М.: Наука, 1977. — 263 с.

17. **Пиотровский Р. Г.** Машинный перевод: Некоторые итоги и перспективы. — В кн.: Проблемы структурной лингвистики 1971, М.: Наука, 1972, с. 521—533.

18. **Сова Л. З.** Формальные связи в русском языке. — В кн.: Проблемы структурной лингвистики 1972. — М.: Наука, 1973, с. 527—542.

19. **Гак В. Г.** Высказывание и ситуация. — В кн.: Проблемы структурной лингвистики 1972. — М.: Наука, 1973, с. 349—372.

20. **Грин Б.** «Бейсбол» — программа, автоматически отвечающая на вопросы. — В кн.: Вычислительные машины и мышление. — М.: Мир, 1967, с. 221—233.

21. **Белоногов Г. Г., Богатырев В. И.** Автоматизированные информационные системы. — М.: Советское радио, 1973. — 328 с.

22. **Белоногов Г. Г., Котов Р. Г.** Автоматизированные информационно-поисковые системы. — М.: Советское радио. 1968. — 182 с.

23. **Пиотровский Р. Г.** Текст, машина, человек. — Л.: Наука, 1975. — 329 с.

24. **Денисов П. Н.** Принципы моделирования языка. — М.: МГУ, 1965. — 205 с.

25. **Шаумян С. К.** Структурная лингвистика. — М.: Наука, 1965. — 395 с.

26. **Шаумян С. К., Соболева П. А.** Основания порождающей грамматики русского языка. — М.: Наука, 1968. — 373 с.

27. **Сова Л. З.** Аналитическая лингвистика. — М.: Наука, 1970. — 254 с.

28. **Бидер И. Г., Большаков И. А., Еськова Н. А.** Формальная модель русской морфологии. — М.: Ин-т руского языка АН СССР, ч. 1, 1978. — 48 с., ч. II, 1978, 59 с.

29. **Смысловое** восприятие речевого сообщения. — М.: Наука, 1976. — 263 с.

30. **Слейгл Дж.** Искусственный интеллект. — М.: Мир, 1973. — 319 с.

31. **Хант Э.** Искусственный интеллект. — М.: Мир, 1978. — 558 с.

Глава вторая

Представление информации в виде предикатных выражений использовалось в основном на первом этапе развития диалоговых систем [30—31]. Однако и в настоящее время подобные описания продолжают развиваться из-за их удобства при решении задачи планирования решений [32—36]. В частности, в [36] описана получившая широкую известность система STRIPS, послужившая образцом для подражания при создании многочисленных планирующих систем для организации целесообразного поведения интегральных роботов. Пример по-

строения предикатного языка для планиметрии, приведенный в § 2.2, без изменений заимствован из работы [37]. Система E и примеры конкретных трансформационных правил для русского языка, приведенные в этом же параграфе, взяты из монографии [15]. В этой монографии, пожалуй, впервые дано формальное описание всех синтаксических трансформационных правил для русского языка. Ряд ценных соображений о природе подобных правил имеется в работах [25, 38].

Реляционные языки различного типа хорошо описаны в отечественной литературе. Табличные языки Кодда описаны, например, в [39, 40]. Во второй из этих работ, посвященной некоторым философско-методологическим проблемам построения реляционных баз данных, приведена и формальная алгебраическая модель для реляционных алгебр. Различные формализации реляционных баз данных приведены также в работах [41—43].

Языки типа RX-кодов были предложены впервые Э. Ф. Скороходько. В [44—46] даны примеры описаний в этих языках и показано применение подобных языков в информационных и управляющих системах.

Из работы [45] заимствован пример 2.6 настоящей главы.

Язык синтагматических цепей и опирающийся на него язык ситуационного управления широко используются при построении систем ситуационного управления. В работах [47—49] этот язык описан достаточно подробно. Универсальный семантический код, о котором говорится в последнем параграфе главы, предложен В. В. Мартыновым [50, 51]. Другие ролевые языки описаны в работах [52—54]. В [54] описаны общие контуры языков KRL и FRL, а также более простая система фреймовых представлений GUS.

32. **Рафаэл Б.** Думающий компьютер. — М.: Мир, 1979. — 407 с.

33. **Ефимов Е. И., Поспелов Д. А.** Семиотические модели в задачах планирования для систем искусственного интеллекта. — Изв. АН СССР. Техническая кибернетика, 1977, № 5, с. 60—68.

34. **Ефимов Е. И.** Автоматическое обучение решателя задач. — Проблемы управления и теории информации, 1979, № 3, с. 239—256.

35. **Чумаков Б.** Логическая концептуализация поведения. — Проблемы моделирования языковой интеракции. — Труды по искусственному интеллекту. Ученые записки Тартуского государственного университета, Тарту, 1978, вып. 472, с. 38—60.

36. **Файкс Р., Нильсон Н.** Система STRIPS — новый подход к применению методов доказательства теорем при решении задач. — В кн.: Интегральные роботы. — М.: Мир, 1973, с. 382—403.

37. **Падучева Е. В.** Семантический анализ естественного языка при переводе на языки математической логики. — В кн.: Информационно-поисковые системы и автоматизированная обработка научно-технической информации. Т. 2. — М.: ВИНИТИ АН СССР, 1967, с. 156—169.

38. **Ревзин И. И.** Модели языка. — М.: Изд-во АН СССР, 1962. — 188 с.

39. **Цаленко М. Ш.** Реляционные модели баз данных. — В кн.: Алгоритмы и организация решения экономических задач. — М.: Статистика, 1977, Ч. 1, вып. 9, с. 18—36, Ч. 2, 1977, вып. 10, с. 16—29.

40. **Цаленко М. Ш.** Философия и математика моделирования процессов обработки информации (на примере реляционных моделей баз данных). — В кн.: Семиотика и информатика. Вып. 13. — М.: ВИНИТИ АН СССР, 1979, с. 150—183.

41. **Бенаминов Е. М.** Алгебраический подход к моделям баз данных реляционного типа. — В кн.: Семиотика и информатика. Вып. 12. — М.: ВИНИТИ АН СССР, 1979, с. 154—160.

42. **Вольфенгаген В. Э., Мясников А. В.** Алгебраическая интерпретация базы данных. — В кн.: Банки информации для принятия решений. — М.: Московский дом научно-технической пропаганды, 1976, с. 49—55.

43. **Белецкий М. И., Кривец Т. П.** Об информационно-алгоритмическом языке для проектирования АСУ городом. — В кн.: Проблемы создания территориальных звеньев РАС УССР. — Киев: Институт кибернетики АН УССР, 1977, с. 39—46.

44. **Пшеничная Л. Э., Скороходько Э. Ф.** Единицы информационного языка в сопоставлении с единицами естественного языка. — В кн.: Информационно-поисковые системы и автоматизированная обработка научно-технической информации. Т. 2. — М.: ВИНИТИ АН СССР, 1967, с. 187—191.

45. **Пшеничная Л. Э., Скороходько Э. Ф.** Синтез осмысленных предложений на ЭЦВМ. — Проблемы кибернетики. М.: 1963, вып. 10, с. 261—273.

46. **Клыков Ю. И.** Модельный язык. — В кн.: Информационно-поисковые системы и автоматизированная обработка научно-технической информации. Т. 2. — М.: ВИНИТИ АН СССР, 1967, с. 360—371.

47. **Поспелов Д. А., Пушкин В. Н.** Мышление и автоматы. — М.: Советское радио, 1972. — 222 с.

48. **Клыков Ю. И.** Ситуационное управление большими системами. — М.: Энергия, 1974. — 134 с.

49. **Поспелов Д. А.** Логико-лингвистические модели в управлении. — М.: Энергия, 1980. — 238 с.

50. **Мартынов В. В.** Семиологические основы информатики. — Минск: Наука и техника, 1974. — 191 с.

51. **Мартынов В. В.** УСК-3 как язык описания и исчисления смыслов. — Минск: Наука и техника, 1977. — 34 с.

52. **Вольфенгаген В. Э., Кузин Л. Т., Саркисян В. И.** Реляционные методы проектирования банков данных. — Киев: Вища школа, 1979. — 192 с.

53. **Ыйм Х., Салувеэр М.** Фреймы и понимание языка. — Проблемы моделирования языковой интеракции. Ученые записки Тартуского государственного ун-та, Тарту, 1978, вып. 472, с. 101—113.

54. **Системы** представления понятийных знаний с использованием фреймов/ Г. В. Рыбина, Н. А. Строганова, М. И. Фардзинова, А. А. Хромов. — В кн.: Интеллектуальные банки данных. — М.: Советское радио, 1979, с. 25—48.

Глава третья

Диспетчерские информационные системы жесткого типа хорошо описаны в литературе. Укажем, например, на публикации [55—58]. Данные об объеме словарей профессиональных диспетчеров в энергосистемах получены на основании исследований, проведенных в ЦДУ ЕЭС СССР. Эти исследования опирались на анализ записей разговоров диспетчеров. Результаты этого анализа приведены в [59].

Семантические сети — аппарат хорошо известный в теории искусственного интеллекта. Общее представление о них можно получить, например, из монографий [60, 61]. Активные семантические сети, называемые иногда M-сети, были предложены в Институте кибернетики АН УССР. В монографии [62] они описаны достаточно подробно. Основной механизм этих сетей — Система Усиления-Торможения (СУТ), определяющий характер их функционирования, в § 3.3 назван *Активатором*.

В алгоритме *Наблюдение* используются понятия лингвистической переменной и расплывчатого множества, заимствованные из теории Л. Заде [63].

Кроме монографии [62] методы работы с семантическими сетями для случая диспетчерского управления рассмотрены также в [64, 65]. В [66, 67] содержатся некоторые дополнительные сведения о системе МИМИР (Информационная Модель Мира), которая в настоящей монографии названа ДВОЭ.

55. **Лугинский Я. Н., Семенов В. А.** Информационно-вычислительные системы в диспетчерском управлении. — М.: Энергия, 1975. — 159 с.

56. **Лугинский Я. Н., Семенов В. А.** Современное состояние работ по АСДУ в СССР и за рубежом. — М.: Информэнерго, 1976. — 50 с.

57. **Карпина О. Я., Кемельмахер Г. Л., Эпельман Л. Е.** Некоторые вопросы проектирования математического обеспечения систем сбора, обработки и отображения оперативной диспетчерской информации. — Программирование, 1977, № 4, с. 64—71.

58. **Карпина О. Я., Кемельмахер Г. Л., Любарский Ю. Я.** Система СОТ для сбора, обработки и отображения телемеханической информации в оперативном диспетчерском управлении энергосистемами. — В кн.: Технические средства телеобработки информации в АСУ в реальном масштабе времени. — М.: Московский дом научно-технической пропаганды, 1976, с. 109—113.

59. **Дегтярев А. Р.** Использование профессиональных языков в качестве информационно-поисковых языков. — В кн.: Технические средства телеобработки информации в АСУ в реальном масштабе времени. — М.: Московский дом научно-технической пропаганды, 1976, с. 52—56.

60. **Линдсей П., Норман Д.** Переработка информации у человека. — М.: Мир, 1974, с. 355—420.

61. **Попов Э. В., Фирдман Г. Р.** Алгоритмические основы интеллектуальных роботов и искусственного интеллекта. — М.: Наука, 1976, 455 с.

62. **Автоматы** и разумное поведение/ Н. М. Амосов, А. М. Касаткин, Л. М. Касаткина, С. А. Талаев. — Киев: Наукова думка, 1973. — 373 с.

63. **Заде Л.** Понятие лингвистической переменной и его применение к принятию приближенных решений. — М.: Мир, 1976. — 163 с.

64. **Любарский Ю. Я.** Алгоритмы адаптивного поведения в автоматизированном диспетчерском управлении энергосистемами. — В кн.: Эвристические модели в психологии и социологии. — Киев: Институт кибернетики АН УССР, 1976, с. 64—77.

65. **Любарский Ю. Я.** Возможности применения М-автомата в информационных системах оперативного диспетчерского управления энергосистемами. — Электричество, 1977, № 11, с. 24—27.

66. **Любарский Ю. Я.** Представление знаний об объекте управления в диспетчерских информационных системах. — Программирование, 1978, № 1, с. 41—50.

67. **Купершмидт Ю. Я., Любарский Ю. Я.** Формирование понятия текущей ситуации в объекте управления для диспетчерских информационных систем. — В кн.: Средства и системы управления в энергетике, № 2. — М.: Информэнерго, 1978, с. 8—12.

Глава четвертая

Некоторые другие аспекты, связанные с принципами построения диалоговых систем, выдвигаемыми авторами системы ДИСПУТ, может

быть, и не найдешими лока воплощения в существующей системе, освещаются в работах [68—70].

68. **Микулич Л. И., Червоненкис А. Я.** Об использовании формальных исчислений в диалоговых системах. — Труды IV объединенной международной конференции по искусственному интеллекту. — М.: Научный совет по комплексной проблеме «Кибернетика» при Президиуме АН СССР, 1975, вып. 12, с. 176—188.

69. **Микулич Л. И., Червоненкис А. Я.** Лингвистический процессор системы ДИСПУТ. — В кн.: Информационно-программное обеспечение систем искусственного интеллекта. — М.: Московский дом научно-технической пропаганды, 1978, с. 51—56.

70. **Микулич Л. И.** Диалог в автоматизированных системах планирования. — В кн.: Проблемы планирования в транспортных системах, вып. 11. — М.: ИПУ, 1976, с. 93—103.

Глава пятая

Библиотечная система поддержания разработки (БСПР МО) была предложена для ЭВМ БЭСМ-6 А. Г. Красовским и описана в [71]. Смешанные системы для языковых модулей, онирающиеся на принципы гетерархического программирования, впервые были реализованы в рамках программного обеспечения диалоговой системы SHEDLU и описаны в монографии [72]. Принципы гетерархического программирования изложены, например, в работах [73] и [74]. Сетевые грамматики, используемые в системах МИВОС и ДИЛОС (о последней будет идти речь в следующей главе), предложены В. Вудсом. Основная работа Вудса нереведена на русский язык [75].

Диалоговая система ПОЭТ, о которой упоминается в § 5.2, подобно системам МИМИР и ДИСПУТ ориентирована на определенный прикладной модуль. Ее основное назначение — диалог с пользователем в области экономики и материально-технического снабжения. Принципы построения системы ПОЭТ изложены в работе [76].

Язык программирования, онирающийся на расширенные сети переходов, в первоначальной версии был описан еще В. Вудсом в той же работе [75]. Создатели системы МИВОС существенно расширили эту первоначальную версию. Первой нубликацией по языку ATNL была их работа [77].

Как было отмечено в § 5.5, ATNL-интерпретатор реализован с использованием языка РЕФАЛ/2. Описание этой программной системы содержится в работе [78].

71. **Красовский А. Г.** Поддержка разработки МО для СЦВМ. Тезисы докладов Первой всесоюзной конференции «Технология программирования». — Киев: ИК АН УССР, с. 41—42.

72. **Виноград Т.** Программа, понимающая естественный язык. — М.: Мир, 1976. — 292 с.

73. **Лозовский В. С.** Ситуационная и дефинитоная семантика системы нредставления знаний. — Кибернетика, 1979, № 2, с. 98—101.

74. **Лозовский В. С.** Гетерархическое программирование. — В кн.: Всесоюзная конференция «Семиотические модели при управлении большими системами». — М.: Научный совет по комплексной проблеме «Кибернетика» при Президиуме АН СССР, 1979, с. 5—7.

75. **Вудс В. А.** Сетевые грамматики для анализа естественных языков. — Кибернетический сборник. Новая сер., вып. 13. — М.: Мир, 1976, с. 120—158.

76. **Бакланов В. М., Попов Э. В.** «Понимание» фраз ограниченного русского языка. — Изв. АН СССР. Сер. Техническая кибернетика, 1978, № 4, с. 66—70.

77. **Преображенский А. Б., Рыбина Г. В., Хорошевский В. Ф.** МИВОС — многоцелевая естественноязыковая система. — Изв. АН СССР. Сер. Техническая кибернетика, 1979, № 6, с. 142—151.

78. **РЕФАЛ** в мониторной системе «Дубна» БЭСМ-6. Входной язык компилятора и запуск программ/ Ан. В. Климов, Л. В. Проворов, С. А. Романенко, Е. В. Травкина. Препринт № 8 Института прикладной математики АН СССР. М.: ИПМ АН СССР, 1975. — 86 с.

Глава шестая

Система ДИЛОС в различных ее модификациях описана в работах [79—80]. Некоторые другие аспекты, связанные с представлением данных для системы ДИЛОС и организацией ее работы в ВЦ АН СССР, можно найти в работах [81—84].

79. **Брябрин В. М., Поспелов Д. А.** Диалоговая система для информационного поиска, вычислений и логического вывода. — Тр. конференции по вопросно-ответным системам (Вена, Июнь 1975). CP-76-6, IIASA, Austria, 1976, с. 11—19.

80. **Брябрин В. М., Сенин Г. В.** Руководство к системе ДИЛОС — БЭСМ-6. Сер. Математическое обеспечение БЭСМ-6. — М.: ВЦ АН СССР, 1977. — 46 с.

81. **Брябрин В. М.** Универсальная семантическая память в системе ЛОРД. — К кн.: Обработка символьной информации. Вып. 2, — М.: ВЦ АН СССР, 1975, с. 5—21.

82. **Система** ПУЛЬТ-78. Руководство к пользованию/ В. М. Брябрин и др. — В кн.: Сообщения по программному обеспечению ЭВМ. — М.: ВЦ АН СССР, 1978. — 99 с.

83. **Брябрин В. М.** Система управления интеллектуальной базой данных на ЛИСПе. — В кн.: Обработка символьной информации. Вып. 4. — М.: ВЦ АН СССР, 1978, с. 75—83.

84. **Брябрин В. М.** Структурные описания как основа семантической интерпретации естественноязыковых текстов. — В кн.: Взаимодействие с ЭВМ на естественном языке/ Под ред. А. С. Нариньяни. — Новосибирск: ВЦ СО АН СССР, 1978, с. 166—181.

ОГЛАВЛЕНИЕ

Виктор Михайлович Брябрин, Юрий Яковлевич Любарский, Леонид Ильич Микулич, Елена Яковлевна Найденова, Дмитрий Александрович Поспелов, Александр Борисович Преображенский, Владимир Федорович Хорошевский, Алексей Яковлевич Черевоненкис

Диалоговые системы в АСУ

Редактор Е. Т. Семенова
Редактор издательства Л. Д. Никулина
Технический редактор Л. В. Порчахева
Корректор Л. С. Тимохова
ИБ № 2233

Сдано в набор 16.03.83 Подписано в печать 11.05.83 Т-11328
Формат 84×108$^{1}/_{32}$ Бумага типографская № 1 Гарнитура литературная
Печать высокая Усл. печ. л. 10,92 Усл. кр.-отт. 11,13 Уч.-изд. л. 12,46
Тираж 8000 экз. Заказ 3096 Цена 65 к.

Энергоатомиздат, 113114, Москва, М-114, Шлюзовая наб., 10

Ордена Октябрьской Революции и ордена Трудового Красного Знамени Первая Образцовая типография имени А. А. Жданова Союзполиграфпрома при Государственном комитете СССР по делам издательств, полиграфии и книжной торговли. Москва, М-54, Валовая, 28

ЭНЕРГОАТОМИЗДАТ

ГОТОВИТСЯ К ИЗДАНИЮ

Горбатов В. А., Павлов П. Г., Четвериков В. Н. Логическое управление информационными процессами/ Под ред. В. А. Горбатова. — М.: Энергоатомиздат.

Изложены основы логического управления процессами организации, хранения, обновления, поиска и порождения информации, протекающими в ЭВМ и сетях ЭВМ и образующими современный информационный процесс. Особое внимание уделено использованию характеризационных принципов при логическом проектировании баз данных и их физической организации.

Для инженерно-технических и научных работников в области информационного обеспечения ЭВМ и сетей ЭВМ.