ФОРМАЛИЗОВАННОЕ ПРЕДСТАВЛЕНИЕ ЗНАНИЙ О СИСТЕМАХ

ПРЕДСТАВЛЕНИЕ ЗНАНИЙ. ОПЫТ СИСТЕМНОГО АНАЛИЗА

Д. А. ПОСПЕЛОВ

Введение. Проблема представления знаний в области искусственного интеллекта является центральной. От ее успешного решения зависит успех в создании эффективных систем планирования целесообразной деятельности интеллектуальных систем, и в частности роботов. Столь же тесно связана с проблемой представления знаний и проблема организации эффективного диалога между человеком и системой искусственного интеллекта. Наконец, система обработки внешней информации и восприятия во многом опирается на развитую систему представления знаний.

В этой работе мы рассмотрим основные задачи, возникающие в области представления знаний не с точки зрения методов их решения, а с позиции системного анализа. Другими словами, мы будем исследовать и описывать систему представления знаний именно как систему. Поэтому нашей важнейшей задачей будет построение классификации систем представления знаний, выявление генезиса развития этих систем и выявление тех функциональных отношений, которые связывают систему представления знаний с другими системами интеллектуальных устройств.

План настоящей работы следующий. В начале мы рассмотрим тот путь, который привел к самому понятию «система представления знаний» [1] (СПЗ). Затем мы дадим несколько классификаций подобных систем и проанализируем их структуры. Наконец, после описания места СПЗ в системе организации целенаправленного поведения интеллектуальных устройств, мы в заключительной части перечислим основные проблемы в этой области, которые еще необходимо решить.

[1] Соответствующий английский термин — Knowledge Representation System.

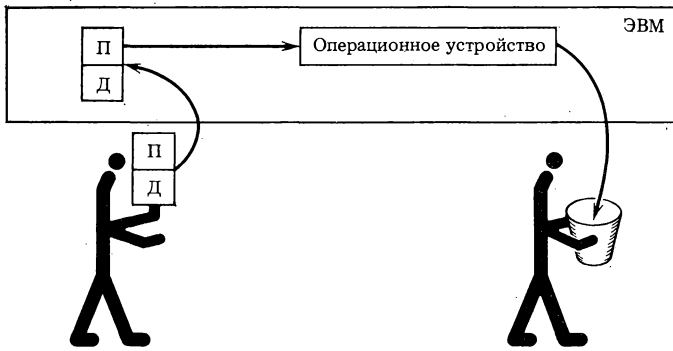

Рис. 1

От данных к знаниям. Проследим, как генетически возникли СПЗ. На начальном этапе развития ЭВМ доминирующую роль играла программа, реализуемая на машине. Именно в программе был овеществлен процесс решения той или иной задачи. Программист, переводя свой замысел в написанную программу, использовал семантику самой задачи, учитывал смысл исходных данных и алгоритма ее решения. Он также соотносил получаемый после реализации программы результат с некоторыми целевыми результатами для решаемой задачи. На рис. 1 показан схематически этот этап в использовании ЭВМ. Здесь ЭВМ выступает как формальная система, на вход которой поступает стандартизированное описание (программа) и на выходе которой также получается некоторая стандартизированная запись. Вся интерпретация производится только самим программистом. Такой взгляд на ЭВМ был впервые четко высказан в работах фон Неймана [9]. Фон Нейман показал, что функционирование ЭВМ можно уподобить работе машины Тьюринга. Однако — более точно — ЭВМ аналогична не машине Тьюринга, а Линейной Ограниченной Машине [7]. Это связано с тем, что память ЭВМ не бесконечна, а лишь потенциально бесконечна. При этом множество порождаемых ею языков есть то множество, которое порождается Линейно Ограниченными Машинами, т. е. множество не нулевого, а первого уровня в классификации, принятой в теории формальных грамматик [5]. Замечу, что в [17] была дана несколько иная трактовка ЭВМ как формальной системы, но для нашего исследования это не существенно. Важно лишь то, что в схеме, показанной на рис. 1, ЭВМ есть аналог некоторой формальной системы.

Именно то, что ЭВМ на начальном этапе своего существования использовалась в соответствии со схемой рис. 1, обусловило «однообразие» работы ЭВМ при решении любых мыслимых задач. Какую бы задачу ни решала ЭВМ (более точно, какую бы программу она ни выполняла), она всегда выполняет лишь определенную последовательность из тех команд, которые входят в ее систему команд. Если человек, не знающий, какую задачу решает ЭВМ, будет наблюдать за каждым ее шагом (например, анализировать распечатку всех действий по шагам), то он может лишь констатировать, что на некотором шаге произошло умножение содержимого определенных ячеек, на другом шаге — пересылка содержимого из одной ячейки в другую и т. д. А каков истинный смысл операций, совершаемых ЭВМ, ему восстановить без программиста невозможно, ибо только программист знает, что именно делает введенная им программа: решает систему дифференциальных уравнений, сочиняет музыку или играет в шахматы.

Обратим внимание на небольшой «довесок» к программе, показанный на рис. 1. Это исходные данные, с которыми работает программа. Как они были организованы на рассматриваемом нами этапе использования ЭВМ? Организованы они были по весьма жесткому и однозначному принципу. Каждое данное представляло собой с точки зрения ЭВМ машинное слово фиксированной длины (как правило, это была длина машинной ячейки, изредка двух таких ячеек). Массив данных не был никак структурирован. Каждый его элемент никак не зависел от других элементов, либо зависел от них только «пространственно» (при вводе в ЭВМ таких математических объектов, как вектор или матрица). В последнем случае при отображении данных в памяти ЭВМ приходилось располагать пространственно связанные элементы в виде адресно связанных ячеек. Причем эта связь должна была быть выражена в явной для человека и программы форме. Например, элементы, соответствующие координатам вектора, располагались в ячейках памяти с соседними номерами, матрицы «вытягивались» в строку и рассматривались в виде векторов особой природы и т. п. Каждое данное было тесно связано с программой и не мыслилось вне ее. В программе должна была найтись хотя бы одна команда, которая могла бы использовать данное в качестве своего операнда.

Дальнейшее развитие программирования пошло по двум направлениям. С одной стороны, происходило укрупнение тех операторов, которые программист мог бы использовать в своей программе. ЭВМ автоматически за счет структуры

может выполнять только вполне определенный набор команд. Но для программиста эти команды слишком «мелки». Программы получаются громоздкими, труднообозримыми, содержащими много ошибок. Укрупнение команд в операторы тесно связано с семантикой решаемых задач. Если человек имеет дело, например, с матричной алгеброй, то ему удобно давать указания ЭВМ прямо в виде операций над матрицами (например, «умножь матрицу A на матрицу B»). Если же программист занят решением на ЭВМ задач теории вероятностей, то для него желательно иметь операторы типа: «Найди математическое ожидание X» или «Вычисли дисперсию $У$». Стремление к введению таких обобщенных операторов в каждой из проблемно-ориентированных областей человеческой деятельности очень быстро привело к эффекту Вавилонской башни. Появилось несколько сот языков программирования для ЭВМ, каждый из которых содержал такой набор стандартных операторов, который был удобен для выбранной предметной области. Кроме того, каждый язык снабжался специальными средствами для перевода записей программ в выбранной системе операторов в запись той же программы в системе команд, доступных ЭВМ. Трудность удалось преодолеть. Были созданы языки программирования (к примеру, АЛГОЛ-68 или ПЛ-1), в которых предусматривались специальные средства для определения и введения в язык новых небольших операторов. С этого момента расширение операторных возможностей для описания процедур достигло такого уровня, который вполне удовлетворяет пользователей ЭВМ.

На практике введение новых обобщенных операторов приводит к схеме, показанной на рис. 2. В отличие от предшествующей схемы, кроме ввода программы и данных в ЭВМ, человек должен еще (если он хочет работать с обобщенными операторами) описать эти операторы и ввести их в специальную программу «Переводчик» [2]. В процессе выполнения необходимой программы «Переводчик» все необходимые операторы расшифровывает согласно введенным описаниям в соответствующие стандартные программы. Эти подпрограммы постоянно хранятся в памяти ЭВМ. На рис. 2 показано n таких подпрограмм.

Отметим два важных для нас момента. Во-первых, программа «Переводчик» и стандартные подпрограммы как бы отрываются от основной программы. Если необходимо решать

---

[2] Мы называем эту программу «Переводчик», хотя более точно ее надо было бы назвать «Интерпретатор». Но этот термин в программировании означает нечто другое.

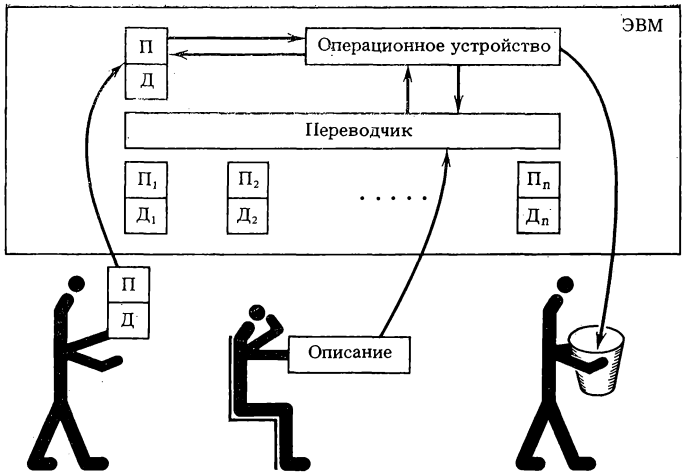

Рис. 2

не одну задачу из данной проблемной области, а много, то «Переводчик» и стандартные подпрограммы могут оставаться в памяти ЭВМ на все время решения основных задач. При смене проблемных областей можно поступить следующим образом. Библиотеку накопленных стандартных подпрограмм можно назвать именем той проблемной области, для которой она использовалась (например, «Матричная алгебра», или «Планиметрия»), и убрать из операционной памяти ЭВМ в долговременную. Так постепенно в долговременной памяти ЭВМ можно накопить богатый набор стандартных подпрограмм из разных проблемных областей. То, что эти подпрограммы хранятся в памяти ЭВМ и тогда, когда они не нужны непосредственно для решения очередной задачи, превращает их в знания ЭВМ. Эти знания, овеществленные в программах, мы будем называть процедурными. При этом программа «Переводчик» сама по себе связана лишь с особенностями языка программирования, используемого для решения задач, и никак не зависит от выбранной пользователем проблемной области. Ее функционирование есть некоторая внутренняя метапроцедура.

Во-вторых, данные в схеме, соответствующей рис. 2, как и ранее, жестко привязаны к тем или иным программам и подпрограммам. (Заметим, что термины «программа» и «подпрограмма» описывают одинаковые объекты. Различие в них возникает только при установлении между этими объектами

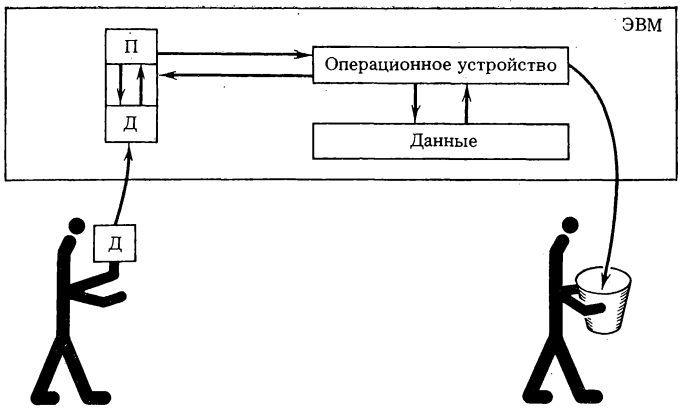

Рис. 3

отношения «часть—целое». В остальном же подпрограммы для ЭВМ абсолютно ничем не отличаются от программ.)

Мы рассмотрели первую ветвь эволюции языков программирования. Она привела нас к появлению процедурных знаний. Но параллельно с этой ветвью, правда несколько запаздывая, вырастала и другая ветвь. Ее развитие началось тогда, когда ЭВМ стали использоваться для решения не только вычислительных задач, но и информационных, в частности, использоваться как элемент в автоматизированных информационно-поисковых системах. В таких системах в памяти ЭВМ постоянно хранится определенный массив информации, а входная программа представляет собой некоторый запрос на поиск информации из хранящегося массива (рис. 3). Практически во всех системах запросы имеют стандартную форму. Поэтому человек вводит в систему подобного типа не программу с данными, а некоторый набор данных, характеризующих запрос. Метапрограмма поиска связана только с формой представления данных, принятой в ЭВМ, и не должна быть связана с той проблемной областью, к которой относится массив хранимой в памяти информации. Другими словами, рис. 3 легко преобразуется в рис. 4, на котором показана структура, пригодная для хранения в памяти ЭВМ информации о многих проблемных областях. Для того чтобы запрос попал туда, куда это нужно пользователю, необходимо указывать в нем имя той проблемной области, которую нужно вызвать в оперативную память ЭВМ для поиска ответа на запрос.

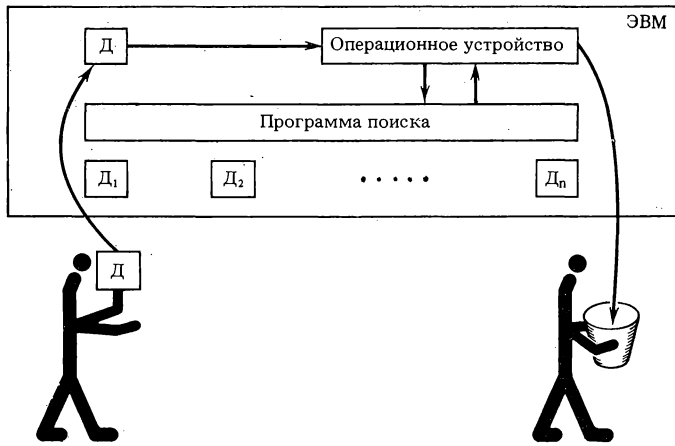

Рис. 4

Эволюция систем подобного типа шла по пути усложнения структур данных, допустимых в ЭВМ. Появились возможности работы с массивами информации и файлами. Файлы — это массивы, структура которых может быть задана программистом. Файлом, например, является всякая форма документа. Вслед за файлами появились списочные структуры, реализованные в известном среди специалистов языке ЛИСП, затем более сложные структуры, о которых мы поговорим чуть позже. Наконец, появилась возможность описывать желаемые для пользователя типы данных и вводить эти описания в программу, которая расшифровывает эти описания и осуществляет работу со структурами данных, удобных для пользователя. На рис. 5 приведена соответствующая схема. Интересно сравнить ее со схемой, показанной на рис. 2. Эти схемы похожи. Только теперь расшифровываются не только операторы, входящие в программу, а данные из этой программы.

Знания ЭВМ, овеществленные в базах данных (на рис. 5*n* баз данных для *n* различных проблемных областей), мы будем называть декларативными. Программа «Переводчик данных» реализует внутреннюю метапроцедуру.

Итак, знания в ЭВМ появляются тогда, когда данные отторгаются от конкретной задачи, решаемой на ЭВМ в настоящий момент, но могут быть вызваны для использования при желании пользователя. Наличие специальных метапроцедур, никак не определяемых решением конкретных задач, является

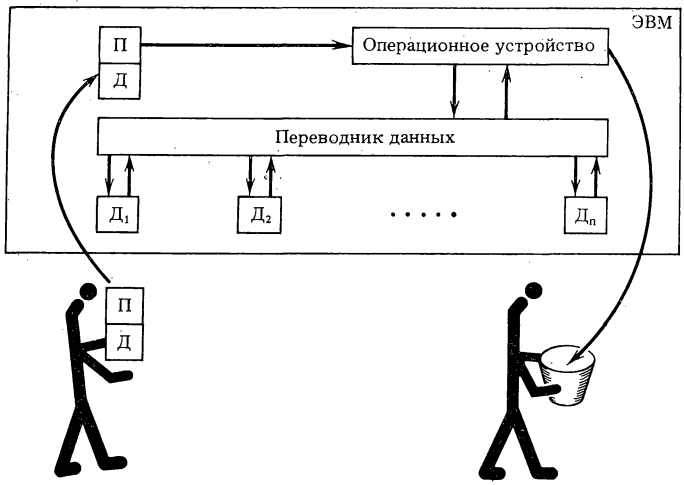

Рис. 5

необходимым условием наличия знаний. Система представления знаний (СПЗ) в самой простейшей форме есть совокупность процедурных и декларативных (в смешанных системах и тех и других) знаний и соответствующих им метапроцедур. Переход от данных к знаниям носит эволюционный характер. Мы пока отметили лишь две особенности, присущие знаниям: отсутствие ориентации на некоторую конкретную программу и наличие специальных метапроцедур, необходимость в которых порождена самими знаниями. Отметим еще ряд особенностей, которыми обладают знания.

Первой и важнейшей из них является возможность внутренней интерпретации знаний. Поясним это на следующем примере. Пусть в память системы введена некоторая таблица, отражающая анкетные данные сотрудников определенного учреждения. Она может, например, иметь следующий вид:

Фамилия	Год рождения	Пол	Специальность
Иванов	1952	м	Сварщик
Дудко	1941	ж	Повар
Логинов	1934	м	Сварщик
Логинов	1955	м	Монтажник

Если в память ЭВМ ввести лишь ту часть таблицы, которая находится ниже шапки, то в памяти образуется файл, с которым может работать программа, выполняющая различные операции, необходимые, например, для отдела кадров или бухгалтерии. Однако, как уже говорилось выше, сама ЭВВ не будет при этом иметь никакой информации о том, что именно хранится в данном файле. Если в нее ввести запрос: «Что тебе известно об Иванове?» или «Перечисли всех женщин, работающих на данном предприятии», то она не сможет ответить ничего вразумительного. Если же в память ЭВМ ввести всю таблицу целиком и сообщить ЭВМ, что верхняя строка (шапка) есть перечисление имен (в этом случае их чаще называют атрибутами) того, что стоит в соответствующих столбцах, то машина станет способной к самостоятельной интерпретации записанных в нее данных. Данные получат собственные имена и станут обладать некоторой семантикой.

Вторым шагом на пути к знаниям является введение отношений над единицами информации, хранимыми в памяти. В качестве примера введения отношений рассмотрим вновь нашу кадровую таблицу. Можно считать, что каждая строка этой таблицы образует четырехместное отношение, схема которого задана шапкой таблицы. Это отношение связывает между собой все анкетные данные, относящиеся к одному индивиду. Кроме отношений такого типа, можно задавать отношения и других типов. Например, родовидовые, временные, каузальные, пространственные и т. п. Списки наиболее интересных для СПЗ отношений проводились неоднократно. В ситуационном управлении [12] этому уделялось особое внимание. Другим примером может служить список отношений, приведенный в Приложении 3 к книге [10].

Как известно, именно специфика вводимых отношений над элементами сложного объекта определяет особенности того или иного объекта или, точнее, знаний об этом объекте [1]. В существующих сейчас реально СПЗ именно отношения и взаимные связи через ссылки в информационных единицах определяют организацию СПЗ, ее целостность в духе того, как это трактуется в [2]. С другой стороны, именно система отношений, вводимых в знания, позволяет получать алгебраические и логические модели представления знаний.

Еще одной особенностью того, что принято называть знаниями, является шкалированность. Сущность шкалирования состоит в том, что знания о некоторой предметной области как бы размещены в специальном семантическом пространстве, шкалами которого являются некоторые обобщенные характеристики. Идея шкалирования и семантического про-

странства восходит к Ч. Осгуду, предложившему в конце 20-х годов схему эксперимента по выявлению структуры этого пространства. В эксперименте Осгуда на начальной стадии в качестве шкал использовались шкалы, образованные словами антонимами типа: добрый-злой, высокий-низкий, острый-тупой и т. п. Эти слова располагались по концам шкалы, а роль нуля играло нейтральное высказывание: ни добрый, ни злой, ни высокий, ни низкий и т. п. Остальные деления шкалы словами могли и не обозначаться. Испытуемые должны были располагать на этих шкалах слова, характеризующие понятия из списка, задаваемого экспериментатором. Число исходных шкал у Осгуда было около 400. Результаты экспериментов обрабатывались каким-либо способом, позволявшим выделить существенные компоненты. Чаще других для этого использовались методы факторного или кластерного анализа. После обработки выделялись обобщенные компоненты, которые и образуют оси семантического пространства. Еще Осгуд показал, что почти для всех понятий можно построить трехмерное семантическое пространство, оси которого можно интерпретировать как шкалу оценок, шкалу силы и шкалу активности. Метрика этого пространства в его работах совпадала с обычной евклидовой метрикой. Близость расположения в нем слов, характеризующих определенные понятия, оценивала их ситуативную близость.

Идея такого шкалирования и сейчас является в психологии одной из основных при описании когнитивных структур, хранящих знания о мире в голове человека.

Кроме шкал осгудовского типа, для организации знаний характерны еще родовидовые связи, отражающие иерархичность знаний, и шкалы, связанные с частотой или типичностью тех или иных знаний. Другими словами, кроме семантических шкал, отражающих ситуативную классификацию знаний, и шкал, отражающих иерархическую структуру знаний, возникающую в процессе обобщения информации о внешнем мире, существуют еще шкалы, отражающие частоту встречаемости тех или иных явлений и фактов во внешнем мире [4].

В настоящее время уже созданы СПЗ, в которых реализованы свойство внешней интерпретации и возможность установления родовидовых связей или вероятностных шкал. Однако семантические шкалы пока еще не нашли применения в существующих СПЗ и в специализированных языках для работы со знаниями.

Отметим еще одну важную особенность того, что в отличие от данных можно назвать знаниями. Использовавшиеся

на начальном этапе в ЭВМ данные не допускали членения на более мелкие содержательные единицы и с большими трудностями могли быть объединены (через программу) в более сложные структуры типа векторов или матриц. Знания же обычно обладают такой формой представления, которая допускает рекурсивное вложение одних информационных единиц в другие. В следующем разделе статьи мы будем подробно говорить о различных формах представления знаний. Здесь мы заметим следующее. Наличие имен для единиц информации и отношений между ними всегда позволяет задавать знания в виде следующей типовой структуры:

$$(i;\ \langle j_1,\ v_1\rangle,\ \langle j_2,\ v_2\rangle,\ \ldots,\ \langle j_n,\ v_n\rangle).$$

Здесь i — имя информационной единицы, j_k — имена ее частей, которые чаще всего называют слотами, а v_k — значения слотов. В качестве значения некоторого слота может стоять слот второго уровня, что приводит к структуре вида

$$(i;\ \langle(j_1;\ \langle j_1^1,\ v_1^1\rangle,\ \ldots,\ \langle j_1^l,\ v_1^l\rangle)\rangle,\ \ldots,\ \langle j_n,\ v_n\rangle),$$

в которой слот с именем j_1 сам образует систему слотов. Этот процесс развертывания слота в систему новых слотов может продолжаться неограниченно.

Кроме того, в качестве значения слота любого уровня вложенности может стоять имя другой информационной единицы. Появление такого значения есть сигнал о переходе к единице информации с указанным именем. Такие переходы задают отношения между информационными единицами, т. е. структуру на множестве информационных единиц. Наконец, в качестве значения слота может стоять имя некоторой процедуры, что соответствует вызову этой процедуры для ее выполнения. Это позволяет соединять в одно целое декларативные и процедурные знания.

Классификация СПЗ. Прежде всего отметим, что задание любой СПЗ требует задания языков двух типов: языка описания знаний (ЯОЗ) и языка манипулирования знаниями (ЯМЗ). Язык первого типа служит для задания самих знаний в памяти ЭВМ. Язык же второго типа, по существу, описывает те метапроцедуры, которые будут работать со знаниями. На рис. 6 показана общая схема СПЗ.

Метапроцедуры $M_1, M_2, \ldots, M_к$ оперируют с процедурными знаниями. Эти процедурные знания овеществлены в программах $П_1, П_2, \ldots, П_l$). В этих программах нет никаких конкретных исходных данных. В них есть лишь специальные описатели $O_1, O_2, \ldots, O_l$, в которых указано, какие исходные данные и в какой форме должны быть заданы для ра-

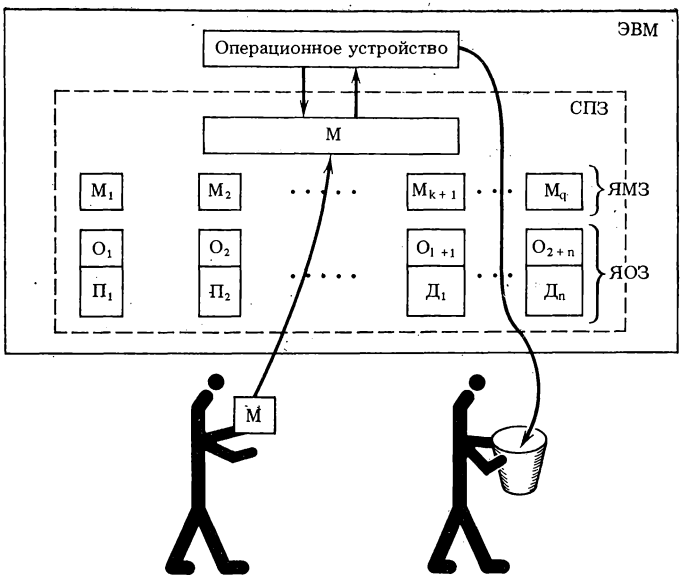

Рис. 6

боты соответствующей программы. Смысл метапроцедур для процедурных знаний может быть самым различным. С их помощью, например, происходит поиск нужной программы, наполнение ее конкретными данными, передача данных от одной программы к другой, расшифровка операторов и т. д.

Метапроцедуры $М_{к+1}$, $М_{к+2}$, . . ., $М_q$ оперируют с декларативными знаниями. Они овеществлены в базах данных $Д_1$, $Д_2$, . . ., $Д_n$. В этих базах нет никаких процедурных элементов. Специальные описатели $О_{l+1}$, $О_{l+2}$, . . ., $О_{l+n}$ служат для указания на те структуры данных, которые используются в данной базе данных. Метапроцедуры, работающие с декларативными знаниями, могут выполнять самые разнообразные задачи: находить нужную базу данных, осуществлять поиск необходимой информации в базе данных, переводить данные из одной формы представления в другую и т. д.

Набор метапроцедур, обозначенный на рис. 6 буквой М,— это набор метапроцедур над метапроцедурами. Другими словами, это набор метапроцедур второго порядка. Их задача — осуществление связи между решаемой на ЭВМ задачей и СПЗ. С их помощью выбираются нужные метапроцедуры первого уровня. На рис. 6 отмечены уровни, соответствующие ЯОЗ и ЯМЗ.

Классификацию СПЗ мы начнем с классификации ЯОЗ. Рассмотрим сначала процедурные знания. Несмотря на большое разнообразие языков программирования, используемых для описания процедурных знаний, их можно разделить на два типа: языки, в явной форме задающие процедуру, и языки, задающие процедуру в неявной форме. Большинство языков программирования относится к первому типу. В той или иной форме в основе этих языков лежит модель продукций, хорошо известная в логике. Эта модель служит основой большинства формальных уточнений понятия алгоритма. Элементарный шаг в таких языках связан с выполнением подстановки вида: $A \Rightarrow B$, где A и B некоторые выражения языка РЕФАЛ. В языках типа АЛГОЛ, ФОРТРАН, КОБОЛ и подобных им продукционная модель используется в несколько модифицированной форме. Однако эта модификация не меняет сути дела. В любом случае в языках подобного типа пользователь при описании процедурных знаний должен в явной форме задать алгоритм, реализующий эту процедуру. Расшифровка алгоритма в ЭВМ при этом предполагается однозначной (хотя реализация может быть и неоднозначной, если, например, в записи процедуры имеется оператор случайного выбора, опирающийся на анализ значения некоторой случайной величины, вырабатываемой на ЭВМ). Другими словами, ЯОЗ есть язык алгоритмического типа. При манипулировании знаниями, описанными на ЯОЗ, расшифровка этих знаний является однозначной.

Однако среди ЯОЗ есть и такие, которые не опираются в явном виде на продукционную модель. Наиболее известным из них является язык ЛИСП, в основе которого лежит λ-исчисление Чёрча. Отметим, что еще весьма давно языки программирования такого типа предлагались в нашей стране. Однако это предложение не было тогда доведено до уровня работающего языка.

Для декларативных знаний в системах искусственного интеллекта и других системах, базирующихся на знаниях, сейчас используются несколько типов ЯОЗ. Среди них наибольшей популярностью пользуются логические, реляционные, фреймовые и сценарные ЯОЗ. В качестве логических ЯОЗ в подавляющем числе случаев используются языки исчисления предикатов первого порядка. В подобных языках все описания представляют собой правильно построенные формулы используемого исчисления. В этой форме описываются как факты, так и закономерности. Преимуществом таких языков являются достаточно эффективные процедуры преобразования знаний. Для этого могут быть использова-

ны такие универсальные процедуры, как обратный метод Маслова или метод резолюций с его многочисленными модификациями. Сведение преобразования знаний к процедурам логического вывода позволяет точно ставить и решать многие задачи, связанные с противоречиями в знаниях или с их неполнотой. Существенным недостатком, присущим всем ЯОЗ логического типа, является громоздкость описаний и их ненаглядность. Кроме того, в СПЗ, основанных на логических языках, практически невозможно проводить композицию и декомпозицию отдельных единиц знаний. Это не позволяет в явной форме отразить принципиальное свойство любого сложного явления или факта, имеющего системный характер, описываемого некоторыми структурами над более мелкими явлениями или фактами. По этой же причине в логических языках плохо реализуются процедуры обобщения и организация родовидовых классификаций. Эти языки рассчитаны лишь на один фиксированный уровень описания знаний. И там, где этого единственного уровня достаточно, они могут применяться вполне успешно.

Неоднократно делались попытки увеличить выразительную силу языков логического типа. Эти попытки шли по пути усложнения исчислений, приближения их к требованиям СПЗ. Например, совершался переход от исчислений с одним типом термов и формул к многосортным исчислениям, вводились богатые наборы модальных операторов, в явной форме делались попытки введения времени и меток ситуаций в формулы исчисления и т. п. Однако на сегодняшний день эти попытки не привели к сколько-нибудь убедительным результатам. И хотя логические языки по-прежнему привлекают внимание исследователей (особенно после появления языка программирования ПРОЛОГ, моделирующего, по сути, процедуры вывода в некотором логическом исчислении), но при необходимости работать со знаниями, отражающими особенности достаточно сложной проблемной области, приоритет принадлежит языкам иного типа.

Одним из типов языков такого рода являются реляционные языки. Именно они получили в 70-х годах широкое распространение в базах данных (реляционные базы данных). Особенностью реляционных языков является наличие средств для явного задания именованных отношений между отдельными единицами знаний.

Существует довольно много реляционных языков, отличающихся друг от друга способами задания отношений. Наиболее просты табличные языки. Именно на примере языка такого типа мы иллюстрировали выше свойство внутренней

интерпретируемости знаний. Как показал первый исследователь табличных реляционных языков—Кодд, математической моделью их является исчисление предикатов первого порядка. И следовательно, по своим возможностям табличные языки эквивалентны тем, которыми обладают классические предикативные исчисления.

Каждая строка таблицы задает n — арное отношение на атрибутах, которые соответствуют столбцам таблицы. Такое задание отношений не всегда удобно и вызывает массу технических и принципиальных трудностей. Это породило многочисленную семантическую критику баз данных и баз знаний, основанных на табличных представлениях.

Реляционными языками иного типа являются RX-коды и язык синтагматических цепей, использовавшиеся в ситуационном управлении [12]. В этих языках отношения входят в формулы языка таким образом, что легко реализуются иерархические записи. Это дает возможность строить эффективные процедуры декомпозиции и композиции знаний, а также проводить обобщение информации различного типа [12]. Язык синтагматических цепей не требует перехода к бинарным отношениям, так как он содержит средства, позволяющие использовать отношения любой размерности.

Моделями реляционных языков являются семантические сети. В таких сетях вершинам соответствуют единицы знаний (допускается наличие многосортных знаний, что может отражаться в различной разметке вершин), а именованным дугам — отношения между этими информационными единицами. На семантических сетях обычно определяются классические теоретико-множественные операции, а кроме того, специальные операции вставки на место любой вершины фрагмента и стягивания некоторого ее фрагмента в обобщенную вершину. Это позволяет строить иерархические семантические сети и отображать на них операции декомпозиции и композиции.

Таким образом, реляционные языки хорошо отражают такие свойства знаний, которые связаны с наличием у них внутренней структуры и сети отношений, связывающих отдельные единицы знаний в ситуативные или родовидовые классифицирующие образования. Более сложной проблемой для языков реляционного типа является проблема выводимости одних знаний из других. Для табличных языков положение несколько проще, чем для языков типа языка синтагматических цепей. В настоящее время делаются многочисленные попытки построить эффективные процедуры поиска информации в базах, использующих реляционные языки.

Этому несколько мешает та свобода, которая в нетабличных языках позволяет строить практически любую семантическую структуру. Такая свобода затрудняет построение формальных процедур. И это явилось одним из источников попыток както упорядочить саму структуру представления знаний, сохранив все те возможности, которые им должны быть присущи.

Так, появились языки фреймового типа. Сейчас они считаются наиболее перспективным классом языков для представления знаний. Общая структура фрейма имеет следующий вид:

{Имя фрейма ⟨Имя слота⟩ ⟨Значение слота⟩ ⟨Имя слота⟩ ⟨Значение слота⟩ . . . ⟨Имя слота⟩ ⟨Значение слота⟩ . . .}.

Число слотов фрейма может быть фиксированным или может наращиваться в процессе заполнения и формирования базы знаний. Каждый слот фрейма может содержать как некоторую конечную информацию, так и делиться на ряд слотов более низкого уровня. Другими словами, слоты можно вкладывать друг в друга, как деревянные матрешки. Это позволяет в полной мере реализовать во фреймах операцию декомпозиции знаний. Однако возможно и другое положение, когда значением некоторого слота данного фрейма окажется имя слота более высокого уровня в данном фрейме. Такие отсылки позволяют организовать в рамках фрейма деревья родовидовых классификаций. Наконец, значением некоторого слота может оказаться имя другого фрейма с указанием имени отношения, связывающего данный слот (и, следовательно, данный фрейм) с этим новым фреймом. Так возникает сеть фреймов, которая очень напоминает семантическую сеть. В вершинах сети находятся фреймы или слоты, а связи между ними преобразуются в именованные отношения. Конечно, нет никакой принципиальной разницы между слотами и фреймами. Деление это чисто условно, что связано с выделением в системах тех или иных составляющих их компонентов или с выделением системы из некоторой среды [2]. Сказанное о фреймах тесно связано с тем, что говорилось о структурированности знаний ранее.

Во фреймовых языках весьма просто осуществляются вставки и стягивания, о которых мы говорили в связи с семантическими сетями. Это означает, что при таком способе представления знаний весьма нетрудно организовать процедуры, связанные с введением обобщенных понятий. Наконец, фреймовые языки позволяют реализовать еще одну особенность знаний, о которой мы не говорили. Это так называемая активность знаний. Исторически сложилось так, что при решении задач на ЭВМ роль активного начала всегда играла

программа. Она инициировала процессы, протекающие в ЭВМ, искала и находила нужные для себя данные в памяти машины и размещала полученные результаты в этой памяти. Данные же играли пассивную роль. Сами по себе они не инициировали никаких процессов. Это положение было связано со всей технологией использования ЭВМ при решении задач, когда лишь программист определял необходимость решения задачи и время начала процесса, связанного с ее решением. При включении ЭВМ в контур управления некоторым объектом инициация начала процесса решения задачи могла осуществляться не программистом, а некоторыми данными, приходящими в виде сигналов от объекта управления. Это был первый шаг на пути активизации данных.

Активизация знаний — это следующий шаг в данном направлении. Источником выполнения программ становятся состояния базы знаний. Примерами таких активизирующих состояний могут служить состояния, связанные с обнаружением неполноты в знаниях или того, что знания частично противоречивы. Столь же активно используются знания в задачах синтеза необходимых процедур [6]. И это сближает знания в современных ЭВМ с когнитивными структурами, присущими животным и человеку. Ибо у живых существ именно знания являются тем активизирующим началом, которое вызывает к исполнению необходимые процедуры или формирует нужные процедуры из готовых блоков.

Способность к активности фреймовых представлений обеспечивает то, что в качестве значений слотов фреймов могут выступать имена некоторых стандартных процедур, программы выполнения которых хранятся либо в самом этом слоте, либо где-то в специальном хранилище программ. Для организации связи с процедурами используется обычно стандартная продукционная запись вида $A \Rightarrow B$, где A есть некоторое проверяемое при использовании продукции условие, а B — указание на то, что надо сделать, если условие A выполнено. В адаптирующихся системах продукции могут иметь вид $A \Rightarrow B, C$, где C описывает условия изменения левых частей продукций в данной продукции и, возможно, в других продукциях, расположенных в иных слотах данного фрейма или других фреймов. В блоке C, конечно, содержится и информация об адресах всех модифицируемых продукций.

Появление фреймовых языков вызвало ряд исследований в области экспериментальной психологии. Были проведены эксперименты по проверке гипотезы о том, что когнитивные структуры человека устроены аналогично фреймовым структурам. К сожалению, эти эксперименты дали отрица-

тельный результат. Когнитивные структуры человека устроены иначе, чем фреймовые представления [3].

Особым типом фреймовых представлений являются так называемые сценарии [20]. По сути своей сценарии — это процедурные знания, записанные в виде семантической сети с вложениями, которые можно представить в виде вложенных структур слотов. Дуги этой сети отождествляются с элементарными действиями, а пути — с цепочками этих действий. Вершины сценария описывают состояния, в которых имеется альтернативный выбор для дальнейшего движения к целевому состоянию. Но, как и для фреймовых структур, попытки найти аналогии между сценариями и процедурными знаниями в когнитивных структурах человека оказались тщетными [3].

Знания и поведение. В этом разделе мы коснемся тех вопросов, которые связаны с использованием знаний при построении планов целесообразного поведения. Формально для искусственных систем типа роботов эти вопросы связываются с задачами планирования целесообразного поведения [6]. Общая схема этой взаимосвязи обсуждалась еще в работе [14], в которой была введена структура гиромата — устройства, операциональная структура которого меняется в зависимости от решаемой задачи и имеющихся в этот момент времени знаний. В работе [15] показывалось, как меняется поведение системы при введении в нее знаний о ней самой. А в работе [8] отмечалось, что без всякой ситуативной динамики в системе предпочтений на знаниях невозможно осуществить целесообразный выбор в альтернативных ситуациях.

Резюмируя все сказанное, можно сформулировать следующее общее утверждение: СПЗ играют центральную роль при решении всех основных задач, связанных с созданием интеллектуальных систем. Ни планирование их целесообразной деятельности, ни осуществление коммуникативной функции между ними и человеком, ни организация коллективного взаимодействия интеллектуальных искусственных систем между собой невозможны без развитых СПЗ и языков обработки знаний.

Нерешенные задачи. В области представления знаний имеется большое количество нерешенных задач. Перечислим основные из них.

1. Задача создания активных СПЗ и организации на их основе процедурных знаний, необходимых для данной ситуации. Во втором разделе статьи уже говорилось о том, что для человека характерен примат декларативных знаний над процедурными, чего пока нет в искусственных системах.

Неоднократно в когнитивной психологии делались попытки построить общую модель активности в знаниях, вызывающей возникновение тех или иных процедур [3], но пока эти результаты еще не имеют достаточной объяснительной силы.

2. В моделях знаний необходимо отражать информацию о мотивах, целях, поступках и поведении, которая имеет моральную, этическую и социальную оценочную компоненты. Первоначальные попытки сделать это [21] показывают, что тут можно надеяться получить ряд новых по качеству результатов.

3. Логика рассуждения над когнитивными структурами, которой пользуются люди, во многом непохожа на привычные схемы человеческих рассуждений. Это требует разработки специальных логических систем, в рамках которых эти рассуждения могли бы быть смоделированы [13, 19]. Сюда же относятся и проблемы, связанные с индуктивными схемами выводов, один из видов которых получил недавно свою формализацию [18].

4. Необходима разработка разнообразных систем правдоподобных рассуждений, рассуждений по ассоциации и рассуждений по аналогии.

5. Необходимо учитывать в знаниях различную неполноту, связанную с качественным характером знаний, недетерминированностью их и т. п.

ЛИТЕРАТУРА

1. *Блауберг И. В., Садовский В. Н., Юдин Э. Г.* Системный подход: предпосылки, проблемы, трудности. М.: Знание, 1969. 48 с.
2. *Блауберг И. В., Юдин Э. Г.* Становление и сущность системного подхода. М.: Наука, 1973. 270 с.
3. *Величковский Б. М.* Современная когнитивная психология. М.: Изд-во МГУ, 1982. 336 с.
4. Вероятностное прогнозирование в деятельности человека. М.: Наука, 1977. 391 с.
5. *Гладкий А. В.* Лекции по математической лингвистике для студентов НГУ. Новосибирск: НГУ, 1966. 189 с.
6. *Ефимов Е. И.* Решатели интеллектуальных задач. М.: Наука, 1982. 316 с.
7. *Минский М.* Вычисления и автоматы. М.: Мир, 1970. 364 с.
8. *Наппельбаум Э. Л., Поспелов Д. А.* Субъективное структурирование ситуации в задачах коллективного принятия решений.— В кн. Нормативные и дескриптивные модели принятия решений. М.: Наука, 1981, с. 191—205.
9. *Фон Нейман Дж.* Вычислительная машина и мозг.— В кн.: Кибернетический сборник. М.: Иностр. лит. 1960, вып. 1, с. 11—60.
10. *Одрин В. М., Картавов С. С.* Морфологический анализ систем. Киев: Наук. думка, 1977. 147 с.
11. *Петренко В. Ф.* Введение в экспериментальную психосемантику:

Исследование форм репрезентации в обыденном сознании. М.: Изд-во МГУ, 1983. 175 с.
12. *Поспелов Д. А.* Логико-лингвистические модели в системах управления. М.: Энергоатомиздат, 1981, 231 с.
13. *Поспелов Д. А.* О «человеческих» рассуждениях в интеллектуальных системах.— В кн.: Вопросы кибернетики: Логика рассуждений и ее моделирование. М.: Науч. совет по комплекс. пробл. «Кибернетика» при Президиуме АН СССР, 1983, с. 5—37.
14. *Поспелов Д. А.* Системный подход к моделированию мыслительной деятельности.— В кн.: Проблемы методологии системного исследования. М.: Мысль, 1970, с. 333—368.
15. *Поспелов Д. А.* «Сознание», «самосознание» и вычислительные машины. — В кн.: Системные исследования. Ежегодник. 1969. М.: Наука, 1969, с. 178—184.
16. *Садовский В. Н.* Основания общей теории систем. М.: Наука, 1974. 277 с.
17. *Успенский В. А.* Машина Поста. М.: Наука, 1979. 95 с.
18. *Финн В. К.* Информационные системы и проблемы их интеллектуализации.— Научно-техническая информация, 1984, серия 2, № 1, с. 1—14.
19. *Чесноков С. В.* Детерминационный анализ социально-экономических данных. М.: Наука, 1982. 168 с.
20. *Шенк Р.* Обработка концептуальной информации. М.: Энергия, 1980. 360 с.
21. *Шустер В. А.* Субъективные оценки словесных и фреймовых описаний поступков.— В кн.: Вопросы кибернетики: Логика рассуждений и ее моделирование. М.: Науч. совет по комплекс. пробл. «Кибернетика» при Президиуме АН СССР, 1983, с. 103—136.

Поспелов Д. А. Представление знаний. Опыт системного анализа // Системные исследования. Методологические проблемы. Ежегодник / Под ред. Д. М. Гвишиани, В. Н. Садовского. — № 17. 1985. М.: Наука, 1986. — С. 83–102. — SR17_Pospelov_1985.pdf

```
@InBook{SR17_Pospelov_1985,
  title =         "Представление знаний.
                   Опыт системного анализа",
  author =        "Поспелов, Д. А.",
  booktitle =     "Системные исследования.
                   Методологические проблемы.
                   Ежегодник",
  publisher =     "Наука",
  editor =        "Гвишиани, Д. М. and Садовского, В. Н.",
  number =        "17. 1985",
  year =          "1986",
  address =       "М.",
  language =      "russian",
  pages =         "83--102",
  numpages =      "360",
  note =          "\url{SR17_Pospelov_1985.pdf}"
}
```