МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
МИНИСТЕРСТВО ОБРАЗОВАНИЯ СТАВРОПОЛЬСКОГО КРАЯ
ГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
СТАВРОПОЛЬСКИЙ ГОСУДАРСТВЕННЫЙ
ПЕДАГОГИЧЕСКИЙ ИНСТИТУТ

**В.С. Тоискин**

# ИНТЕЛЛЕКТУАЛЬНЫЕ ИНФОРМАЦИОННЫЕ СИСТЕМЫ

## Учебное пособие

## Часть 2

**Ставрополь, 2010**

УДК 681.518(075.8)
ББК 32.813я73
Т 50

Печатается по решению
редакционно-издательского совета
Ставропольского государственного
педагогического института

**Рецензенты:**

доктор технических наук, профессор, заслуженный работник высшего образования РФ ***В.П. Пашинцев*** (ГОУ ВПО Сев-КавГТУ)

кандидат технических наук, доцент ***А.М. Еремин*** (ГОУ ВПО СГПИ)

**Тоискин В.С.**

Т 50 **Интеллектуальные информационные системы**: Учебное пособие. – Часть 2. – Ставрополь: Изд-во СГПИ, 2010. – 188 с.

Рассматриваются модели представления знаний, методы поиска решений в интеллектуальных информационных системах. Раскрывается сущность алгоритмов, положенных в основу обучения систем, основанных на искусственных нейронных сетях. Приведена общая характеристика инструментальных средств проектирования интеллектуальных систем

Материал учебного пособия соответствует требованиям Государственного образовательного стандарта высшего профессионального образования к одноименной дисциплине специальности «Прикладная информатика (в экономике)».

УДК 612.518(075.8)
ББК 32.813я73

© Тоискин В.С., 2010
© Ставропольский государственный педагогический институт, 2010

# ВВЕДЕНИЕ

Эволюция информационных технологий и систем все в большей степени определяется их интеллектуализацией. Интеллектуальные информационные системы – одна из наиболее перспективных и быстро развивающихся научных и прикладных областей информатики. Она оказывает существенное влияние на все научные и технологические направления, связанные с использованием компьютеров, и уже сегодня дает обществу то, что оно ждет от науки – практически значимые результаты, многие из которых способствуют кардинальным изменениям в сферах их применения.

Целями интеллектуальных информационных систем являются, во-первых, расширение круга задач, решаемых с помощью компьютеров, особенно в слабоструктурированных предметных областях, и во-вторых, повышение уровня интеллектуальной информационной поддержки современного специалиста.

Ключевым компонентом научного фундамента интеллектуальных информационных технологий является искусственный интеллект (ИИ). Для создания и развития ИИ как научного направления за рубежом много сделали Н. Винер, У. Маккаллох, У. Питтс, Д. Маккарти, Ф. Розенблат, Н. Хомский, Т. Виноград, М. Куиллиан, Н. Нильсон, П. Уинстон, Л. Заде, Д. Ленат, Дж. Хинтон, Дж. Андерсон, Ж.-Л. Лорьер и многие другие. В СССР, а затем в России со становлением и развитием ИИ связывают имена А.А. Ляпунова, А.И. Берга, Г.С. Поспелова, М.Л. Цетлина, М.М. Бонгарда, М.А. Гаврилова, А.П. Ершова, В.Н. Пушкина, Л.Т. Кузина, А.С. Нариньяни, А.И. Половинкина, В.В. Чавчанидзе, В.К. Финна, Э.В. Попова, Э.Х. Тыугу, Н.Н. Непейводы, И.П. Кузнецова, О.И. Ларичева, А.И. Галушкина, А.Н. Горбаня, А.В. Чечкина и многих других. Следует отметить исключительную роль Д.А. Поспелова и его научной школы: В.Н. Вагина, Т.А. Гавриловой, А.П. Еремеева, Г.С. Осипова, В.Ф. Хорошевского и др.

Название учебного пособия соответствует названию учебной дисциплины федерального компонента, предусмотренной Государственным образовательным стандартом для студентов, обучающихся по специальности «Прикладная информатика (в экономике)». Пособие является продолжением одноименного издания (2009 г).

Цель написания данного учебного пособия – дать студентам знания из той области информатики, которая связана с созданием и применением программно-аппаратных комплексов, способных автоматически или при посредстве человека решать задачи анализа и синтеза информации, еще недавно считавшиеся доступными только интеллекту человека.

Главный акцент в изложении содержания учебного пособия делается на раскрытие сущности различных технологий, применяемых при построении интеллектуальных информационных сетей. Уделено внимание моделям и методам принятия решений, применяемым в экспертных системах, методам интеллектуальной поддержки управленческих. Системы, основанные на искусственных нейронных сетях, освещаются в основном с раскрытием сущности алгоритмов, положенных в основу обучения таких систем, и особенностей различных схем их реализации.

**Завершается учебное пособие изложением состояния и перспектив развития ИИС в связи с развитием информационных технологий.**

# 1. МОДЕЛИ ПРЕДСТАВЛЕНИЯ ЗНАНИЙ, ОСНОВАННЫЕ НА ПРАВИЛАХ

## 1.1. Основные понятия инженерии знания

В настоящее время требуются информационные системы, которые, с одной стороны, могут использовать большое количество знаний, передаваемых специалистами, а с другой – способны вступать в диалог и объяснять свои собственные выводы. Это предполагает наличие эффективного управления большой по объему и хорошо структурированной базой знаний, строгое разграничение между различными уровнями знаний, наличие множества удобных представлений для правил, схем предикатов или прототипов и четко определенный процесс обмена информацией между различными источниками. Представлением и обработкой знаний в компьютерных системах занимаются исследователи в области инженерии знаний, введенное в 1977 г. Э.Фейгенбаумом.

Постановка и решение любой задачи связаны с определенной предметной областью. Языки, предназначенные для описания предметных областей, называются языками представления знаний. Универсальным языком представления знаний является естественный язык. Однако использование естественного языка в системах машинного представления знаний наталкивается на ряд препятствий, главным из которых является отсутствие формальной семантики естественного языка. Кроме того, необходимо определить, что такое «знания» и «данные».

Напомним, что данными называют информацию фактического характера, описывающую объекты, процессы и явления предметной области, а также их свойства. В процессах компьютерной обработки данные проходят следующие этапы преобразований:

исходная форма существования данных (результаты наблюдений и измерений, таблицы, справочники, диаграммы, графики и т.д.);

представление на специальных языках описания данных, предназначенных для ввода и обработки исходных данных в ЭВМ;

базы данных на машинных носителях информации.

Для того чтобы поместить знания в информационную систему, их необходимо представить определенными структурами данных, соответствующих выбранной среде разработки интеллектуальной системы. Поэтому при разработке информационной системы сначала осуществляются накопление и представление знаний, причем на этом этапе обязательно участие человека, а затем знания представляются определенными структурами данных, удобными для хранения и обработки в ЭВМ. Знания в ИС существуют в следующих формах:

исходные знания (правила, выведенные на основе практического опыта, математические и эмпирические зависимости, отражающие взаимные связи между фактами; закономерности и тенденции, описывающие изменение фактов с течением времени; функции, диаграммы, графы и т. д.);

описание исходных знаний средствами выбранной модели представления знаний (множество логических формул или продукционных правил, семантическая сеть, иерархии фреймов и т. п.);

представление знаний структурами данных, которые предназначены для хранения и обработки в ЭВМ;

базы знаний на машинных носителях информации.

Ученые предлагают различные определения понятия «знания».

Знания – это формализованная информация, на которую ссылаются или используют в процессе решения задачи.

Знания – это закономерности предметной области (принципы, связи, законы), полученные в результате практической деятельности и профессионального опыта, позволяющие специалистам ставить и решать задачи в этой области.

Знания – это хорошо структурированные данные, или данные о данных, или метаданные.

Знание о предметной области – это описание объектов, их окружения, необходимых явлений, фактов, а также отношений между ними.

Знания в области искусственного интеллекта – это комплексные отношения объектов, представленные посредством интенсионального описания объектов (интенсиональное значение). Исследования в области баз данных обеспечивают эффективный доступ к большим массивам экземпляров типа объекта (экстенсиональное значение).

Общение с компьютерной системой на уровне знания предопределяет возможность ввода и использования ею некоторой совокупности взаимосвязанной информации. Сложность понятия «знание» заключена в множественности его носителя и неразрывности с понятием «данные». Выделяют несколько уровней формализации знания о предметной области: знания в памяти человека; знания в форме языковой модели предметной области, используемые человеком и зафиксированные на физических носителях с использованием контекстно-зависимых языков, графических образов и т.п.; знания, формализованные при использовании в ЭВМ; фактографические сведения или данные.

Исследователи в области искусственного интеллекта различными способами классифицируют знания.

*Фактические и стратегические знания.* Фактические знания – основные закономерности предметной области, позволяющие решать конкретные производственные, научные и другие задачи, то есть факты, понятия, взаимосвязи, оценки, правила, эвристики. Стратегические знания – стратегии принятия решений в предметной области.

*Факты и эвристики.* Факты указывают на хорошо известные в той или иной предметной области обстоятельств. Такие знания еще называют текстовыми, имея в виду достаточную их освещенность в специальной литературе и учебниках. Эвристики основываются на индивидуальном опыте специалиста (эксперта) в предметной области, накопленном в результате многолетней практики. Эта категория знаний нередко играет решающую роль при построении интеллектуальных программ. К эвристикам относятся такие знания, как «способы удаления бесполезных гипотез», способы использования нечеткой информации», «способы разрешения противоречий» и т.п.;

*Факты и правила.* Факты – это знания типа «А – это А», такие знания характерны для баз данных и сетевых моделей. Правила, или продукции, – это знания типа «ЕСЛИ А, ТО В».

*Декларативные и процедурные знания.* Под декларативными знаниями подразумеваются описания фактов, явлений (типа «А это В») и они характерны для баз данных. К процедурным знаниям относятся сведения о способах оперирования или преобразования декларативных знаний.

*Интенсиональные и экстенсиональные знания.* Интенсиональные знания – это знания о связях между атрибутами (признаками) объектов данной предметной области. Они оперируют абстрактными событиями и отношениями. Экстенсиональные знания представляют собой данные, характеризующие конкретные объекты, их состояния, значения параметров в пространстве и времени. В этой классификации оперируют интенсионалом и экстенсионалом понятий. Интенсионал понятия – это определение его через соотнесение с понятием более высокого уровня абстракции указанием специфических свойств. Интенсионалы формулируют знания об объектах. Другой способ определяет понятие через соотнесение с понятиями более низкого уровня абстракции или перечисление фактов, относящихся к определяемому объекту – это определение через данные, или экстенсионал понятия.

*Глубинные и поверхностные знания.* В глубинных знаниях отражается понимание структуры предметной области, назначение и взаимосвязь отдельных понятий (глубинные знания в фундаментальных науках – это законы и теоретические основания). Поверхностные знания обычно касаются внешних эмпирических ассоциаций с каким-либо феноменом предметной области.

*Жесткие и мягкие знания.* Жесткие знания позволяют получать однозначные рекомендации при заданных начальных условиях. Мягкие знания допускают множественные, «размытые» решения и различные варианты рекомендаций.

Форма представления знаний оказывает существенное влияние на характеристики ИС. Базы знаний являются моделями человеческих знаний. Однако все знания, которые привлекает человек в процессе решения сложных задач,

смоделировать невозможно. Поэтому в интеллектуальных системах требуется четко разделить знания на те, которые предназначены для обработки компьютером, и знания, используемые человеком. Очевидно, что для решения сложных задач база знаний должна иметь достаточно большой объем, в связи с чем неизбежно возникают проблемы управления такой базой. Поэтому при выборе модели представления знаний следует учитывать такие факторы, как однородность представления и простота понимания. Однородность представления приводит к упрощению механизма управления знаниями. Простота понимания важна для пользователей интеллектуальных систем и экспертов, чьи знания закладываются в ИС. Если форма представления знаний будет трудна для понимания, то усложняются процессы приобретения и интерпретации знаний. Следует заметить, что одновременно выполнить эти требования довольно сложно, особенно в больших системах, где неизбежным становится структурирование и модульное представление знаний.

Выделают три парадигмы представления знаний: логическая, структурная, процедурная.

*Логическая парадигма* отождествляет знание с теорией, то есть теорией первого порядка.

*Структурная парадигма* уделяет особое внимание организации фактов, составляющих базу знаний. С вычислительной точки зрения, факты получаются из семантических элементов использованием заранее заданных правил. Примеры: семантические сети, фреймы, ER-диаграммы базы данных.

Согласно *процедурной парадигме*, база знаний составлена из активных агентов с определенными образами действия. С вычислительной точки зрения, использование знаний сводится к реакции этих агентов на данную ситуацию согласно процедурным правилам в базе знаний.

Рассмотрим модели представления знаний.

В математике *моделью* (или реляционной системой) называется некоторое множество *M* с заданным на нем набором отношений $\{r_1, r_2, ..., r_n\}$. Как извест-

но, отношение математически представляется в следующей форме: имя_отношения (список_аргументов).

Когда идет речь о модели знаний, то отнюдь необязательно, чтобы множество *M* было однородно (состояло только из объектов одной природы). Например, отношение «быть владельцем (*x*, *y*)» устанавливает, что *x* есть владелец *y*, а экземпляр отношения «быть_владельцем (Иван, книга)» имеет дело с объектами разной природы. Важно подчеркнуть, что в общем случае отношение предполагает некоторую процедуру (называемую разрешающей процедурой), которая устанавливает истинность или ложность отношения. Такая процедура может задаваться алгоритмом, правилами, таблицами и иными способами. При этом всегда стоит вопрос об эффективности (и даже существовании) разрешающей процедуры.

Модели представления знаний в интеллектуальных системах традиционно делятся на логические (формальные) и эвристические (формализованные).

К логическим моделям относятся:

логика высказываний;

логика предикатов первого порядка;

логика Хорна в языке Prolog и ему подобных системах;

логика предикатов высших порядков;

трехзначная логика;

логика возможных миров;

модальные логики;

теория нечетких множеств;

нечеткая логика;

лингвинистические переменные.

К эвристическим моделям представления знаний можно отнести:

деревья решений;

семантические сети;

фреймы и сети фреймов;

онтологии;

объектно-ориентированное программирование;

реляционную алгебру;

правила-продукции;

матрицу весов связей обученной нейронной сети.

Рассмотрим лишь ряд наиболее распространенных моделей представления знаний в информационных системах:

логические модели;

продукционные модели;

сетевые модели;

фреймовые модели;

модели, основанные на теории нечетких множеств и нечеткой логике;

модели на основе нейронных сетей.

## 1.2. Методы представления знаний

### *1.2.1. Логические модели*

Постановка и решение любой задачи всегда связаны с ее "погружением" в подходящую предметную область. Все предметы и события, которые составляют основу общего понимания необходимой для решения задачи информации, называются предметной областью. Мысленно предметная область представляется состоящей из реальных или абстрактных объектов, называемых сущностями.

Сущности предметной области находятся в определенных отношениях друг к другу (ассоциациях), которые также можно рассматривать как сущности и включать в предметную область. Между сущностями наблюдаются различные отношения подобия. Совокупность подобных сущностей составляет класс сущностей, являющийся новой сущностью предметной области.

Отношения между сущностями выражаются с помощью суждений. Суждение – это мысленно возможная ситуация, которая может иметь место для предъявляемых сущностей или не иметь места. В языке (формальном или ес-

тественном) суждениям отвечают предложения. Суждения и предложения также можно рассматривать как сущности и включать в предметную область.

Языки, предназначенные для описания предметных областей, называются языками представления знаний. Универсальным языком представления знаний является естественный язык. Однако использование естественного языка в системах машинного представления знаний наталкивается на большие трудности ввиду присущих ему нерегулярностей, двусмысленностей, пресуппозиций (в словаре специальных терминов пресуппозиция – это предварительное предположение, фундаментальное, глубинное допущение, идея или утверждение) и т.п. Но главное препятствие заключается в отсутствии формальной семантики естественного языка, которая имела бы достаточно эффективную операционную поддержку.

Для представления математического знания в математической логике давно пользуются логическими формализмами – главным образом исчислением предикатов, которое имеет ясную формальную семантику и операционную поддержку в том смысле, что для него разработаны механизмы вывода. Поэтому исчисление предикатов было первым логическим языком, который применили для формального описания предметных областей, связанных с решением прикладных задач.

Описания предметных областей, выполненные в логических языках, называются (формальными) логическими моделями.

В основе логических моделей лежит понятие формальной теории, которая задается четверкой: $S = <T, P, A, F>$,

здесь $T$ – конечное множество базовых элементов (алфавит) системы;

$P$ -множество синтаксических правил, позволяющих из $T$ строить синтаксически правильные выражения (формулы);

$A$ – множество априорно истинных формул (аксиомы);

$F$ – семантические правила или правила вывода, позволяющие получить формулы, которые считаются истинными в данной теории.

При представлении знаний используется исчисление предикатов первого порядка.

*Предика́т* (*n*-местный, или *n*-арный) – это функция с множеством значений {0,1} (или «ложь» и «истина»), определённая на множестве $M = M_1 \times M_2 \times ... \times M_n$. Таким образом предикат характеризует каждый набор элементов множества *M* либо как «истинный», либо как «ложный». Или предикат – это то, что утверждается или отрицается о субъекте суждения.

Алфавит исчисления предикатов первого порядка состоит из следующего набора символов:

знаков пунктуации {(, )};

пропозициональных связок $\{\rightarrow, \wedge, \neg, \vee\}$;

знаков кванторов $\{\forall, \exists\}$;

символов переменных $x_k$, $k = 1,2,...$;

*n*-местных функциональных символов $f_k^n$, $k > 1$, $n > 0$, ($f_k^0$ -константная буква);

*n*-местных предикатных символов $p_k^n$, $k > 1$, $n > 1$.

Предикат называют *тождественно-истинным* и пишут: $P(x_1,...,x_n) \equiv 1$, если на любом наборе аргументов он принимает значение 1.

Предикат называют *тождественно-ложным* и пишут: $P(x_1,...,x_n) \equiv 0$, если на любом наборе аргументов он принимает значение 0.

Предикат называют *выполнимым*, если хотя бы на одном наборе аргументов он принимает значение 1.

Из символов алфавита строят выражения, среди которых выделяют *термы, элементарные формулы и правильно построенные формулы* (или просто *формулы).*

Всякий символ переменной или константной буквы – *терм.* Если $t_1,...,t_n$ – терм ($n > 1$), то $f_k^n(t_1,...t_n)$ – терм.

Если $p_k^n$ – предикатный символ, а $t_1,...,t_n$ – терм, то $p_k^n(t_1,...t_n)$ -*элементарная формула (атом).* Атом – это правильно построенная формула.

Если *A*, *B*- это правильно построенные формулы или просто *формулы,* то $\neg A$, $A \wedge B$, $A \wedge B$, $A \rightarrow B$ -есть *формулы.* Если *A* – формула, а *x* – переменная в *A*, то $\exists x(A)$ и $\forall x(A)$ – формулы.

Так как предикаты принимают только два значения, то к ним применимы все операции булевой алгебры, например: отрицание, импликация, конъюнкция, дизъюнкция и т.д.

*Конъюнкцией двух предикатов* $A(x)$ и $B(x)$ называется новый предикат $A(x) \wedge B(x)$, который принимает значение «истина» при тех и только тех значениях $x \in T$, при которых каждый из предикатов принимает значение «истина», и принимает значение «ложь» во всех остальных случаях. Множеством истинности $T$ предиката $A(x) \wedge B(x)$ является пересечение множеств истинности предикатов $A(x)$ – $T1$ и $B(x)$ – $T2$, т.е. $T = T1 \cap T2$. Например: $A(x)$: «$x$ – четное число», $B(x)$: « $x$ кратно 3». $A(x) \wedge B(x)$ – «$x$ – четное число и $x$ кратно 3». Т.е. предикат «$x$ делится на 6».

*Дизъюнкцией двух предикатов* $A(x)$ и $B(x)$ называется новый предикат $A(x)$ $B(x)$, который принимает значение «ложь» при тех и только тех значениях $x \in T$, при которых каждый из предикатов принимает значение «ложь» и принимает значение «истина» во всех остальных случаях. Областью истинности предиката $A(x) \vee B(x)$ является объединение областей истинности предикатов $A(x)$ и $B(x)$.

*Отрицанием предиката* $A(x)$ называется новый предикат , который принимает значение «истина» при всех значениях $x \in T$, при которых предикат $A(x)$ принимает значение «ложь», и принимает значение «ложь», если $A(x)$ принимает значение «истина». Множеством истинности предиката является дополнение $T'$ к множеству $T$ в множестве $X$.

*Импликацией предикатов* $A(x)$ и $B(x)$ называется новый предикат $Z(x) = A(x) \rightarrow B(x)$, который является ложным при тех и только тех значениях $x \in T$, при которых $A(x)$ принимает значение «истина», а $B(x)$ – значение «ложь» и принимает значение «истина» во всех остальных случаях. Читают: «Если $A(x)$, то $B(x)$». Например. $A(x)$: «Натуральное число $x$ делится на 3». $B(x)$: «Натуральное число $x$ делится на 4», можно составить предикат: «Если натуральное число $x$ делится на 3, то оно делится и на 4». Множеством истинности предиката $A(x)$ $B(x)$ является объединение множества $T2$ – истинности предиката $B(x)$ и дополнения к множеству $T1$ истинности предиката $A(x)$.

Для придания формулам содержания их *интерпретируют* как утверждения относительно рассматриваемой предметной области. Назовем *D областью интерпретации*. Под интерпретацией понимают всякую систему, состоящую из непустого множества *D* и какого-либо соответствия, относящего каждой предикатной букве $p_k^n$ некоторое *n*- местное отношение в *D*; каждой функциональной букве $f_k^n$ некоторую *n*- местную функцию, отображающую $D^n \to D$; каждой константной букве $f_k^o$ некоторый элемент из D. При заданной интерпретации всякой элементарной формуле придается значение ИСТИНА или ЛОЖЬ.

Приписывание значения элементарной формуле $p_k^n(t_1,...,t_n)$ осуществляется по следующему правилу: если термы предикатной буквы соответствуют элементам из *D*, удовлетворяющим отношению, определяемому данной интерпретацией, то значением элементарной формулы будет ИСТИНА, в противном случае – ЛОЖЬ.

Значение неэлементарной формулы вычисляется рекуррентно, исходя из таблиц истинности.

Основной задачей, решаемой в рамках исчисления предикатов, является выяснение истинности или ложности заданной формулы на некоторой области интерпретации. Формулы, являющиеся истинными при любой интерпретации, являются *общезначимыми*, а формулы, ложные при любой интерпретации – *невыполнимыми*.

Справедлива следующая основополагающая *теорема дедукции*:

Пусть даны формула $B_1,\ldots,B_n$ и формула *A*. Формула *A* является логическим следствием $B_1,\ldots,B_n$ тогда и только тогда, когда формула $B_1 \wedge B_2 \wedge \ldots \wedge B_n \supset A$ общезначима. *A* логически следует из $B_1, \ldots, B_n$ тогда и только тогда, когда всякая интерпретация I, удовлетворяющая $B_1 \wedge \ldots \wedge B_n$ удовлетворяет и *A*. Формулы $B_1,\ldots,B_n$ называются *посылками*, *A – заключением* логического следования.

Итак, при использовании данного представления необходимо доказывать общезначимость формулы $B_1 \wedge B_2 \wedge \ldots \wedge B_n \supset A$ или невыполнимость формулы $B_1 \wedge \ldots \wedge B_n \wedge \neg A$.

Для исчисления предикатов первого порядка не существует общего метода установления общезначимости любых формул, т.е. оно неразрешимо. Но если некоторая функция общезначима, то существует процедура для проверки этого факта. Наиболее известными методами доказательства теорем являются методы *резолюции* и *обратный* метод.

Рассмотрим представление знаний с использованием исчисления предикатов первого порядка

**Пример**. Никакой торговец подержанными автомобилями не покупает автомобиль для своей семьи. Некоторые люди, которые покупают авто для своих семей, абсолютно нечестные люди. Доказать, что некоторые абсолютно нечестные люди не являются торговцами подержанных авто.

**Доказательство**. Введем следующие обозначения:

предикат $P(x)$ означает что $x$ – торговец подержанными авто;

предикат $Q(x)$ – $x$ купил подержанное авто для своей семьи;

предикат $R(x)$ – $x$ абсолютно нечестен.

Тогда имеются формулы:

B1: $\{\forall x\}\ (P(x) \rightarrow \neg Q(x))$

B2: $\{\exists x\}\ (Q(x) \wedge R(x))$

Доказать, что A: $\{\exists x\}\ (R(x) \wedge \neg P(x))$ является истинной формулой.

Предположим, что B1 и B2 истинны в интерпретации $I$ с областью $D$.

Так как B2 истинна в $I$, то $\{\exists x\} \in D$ такой, что $x$ = a и формула $Q(a) \wedge R(a)$ истинна в $I$. Следовательно, $Q(a)$ истинна в $I$, т.е. $\neg Q(a)$ ложна в $I$. Формулу B1 можно записать так: $\{\forall x\}\ (\neg P(x) \vee \neg Q(x))$. Так как B1 истинна в $I$, а $\neg Q(x)$ ложна в $I$, то ($\neg P(a)$ должна быть истинна в $I$. Отсюда следует, что формула $R(a) \wedge \neg P(a)$ является истинной в $I$. Таким образом $A$, т.е. формула $\{\exists x\}(R(x) \wedge \neg P(x))$, истинна в $I$. Следовательно формула $A$ является логическим следствием формул B1 и B2.

Основным достоинством использования исчисления предикатов в качестве модели представления знаний является наличие единообразной формальной процедуры доказательства теорем. Однако высокая степень единообразия влечет за собой и основной недостаток данного подхода – сложность использования при доказательстве эвристик, отражающих специфику конкретной проблемной области. Указанный недостаток особенно важен при построении экспертных систем, вычислительная мощность которых определяется знаниями, характеризующими специфику проблемной среды. К другим недостатком формальных систем следует отнести их монотонность, отсутствие средств для структурирования используемых элементов и недопустимость противоречий.

В промышленных интеллектуальных системах логические модели используются редко.

### *1.2.2. Продукционная модель представления знаний*

В модели знаний на основе продукций знания представлены совокупностью правил в формате «ЕСЛИ – ТО». Рассмотрим, например, правила порождения родительного падежа слов, задаваемые таблицей 1.1.

Для того, чтобы получить родительный падеж слова «Знахарь» отыскиваем первую подходящую строку, начиная с верхней, в левой колонке табл. 1.1. Строка будет подходящей, если указываемое в ней окончание совпадает с окончанием слова (в данном случае выбирается строка 5). Нетрудно, однако видеть, что строка 6 также подходит для нашей цели, хотя выдаваемый ею результат (правая колонка табл. 1.1.) не верен. Прежде чем мы рассмотрим более подробно это свойство системы продукций, выясним их природу. Рассматривая структуру продукции, нетрудно видеть, что ее условная часть («ЕСЛИ...») определяет ситуацию, в которой продукция применима. В примере со словом «знахарь» ситуация определяется его окончанием, т.е. либо окончанием «арь», либо «ь».

Если ситуация удовлетворяет продукции, то в результате ее применения может быть получен новый объект (состояние) согласно части «ТО ...» в структуре

продукции. Так, применение продукции с номером 5 в табл. 1.1. к слову «знахарь» порождает слово «знахаря», а применение продукции номер 6 дает слово «знахари». Таким образом, одним из основных вопросов в реализации продукционных систем является стратегия выбора альтернативных правил. В общем случае эта проблема нетривиальна.

***Таблица 1.1***

| № п/п | Слово или его окончание в именительном падеже | Слово или его окончание в родительном падеже |
|---|---|---|
| 1. | кино | -кино |
| 2. | -ча | -чи |
| 3. | -ка | -ки |
| 4. | -а | -ы |
| 5. | -арь | -аря |
| 6. | -ь | -и |
| 7. | -ие | -ия |
| 8. | -мя | -мени |
| 9. | -я | -и |

Как правило, задача, формулируемая для продукционной системы, имеет одну из следующих структур

$$\langle S^0, S^f - ? \rangle, \quad (1.1)$$

$$\langle S^0 - ?, S^f \rangle, \quad (1.2)$$

$$\langle S^0, S^f, A - ? \rangle, \quad (1.3)$$

$$\langle S^0, S^f - ?, A - ? \rangle, \quad (1.4)$$

где $S^0$ – начальная ситуация, $S^f$ – конечная (желаемая, требуемая ситуация), $A$ – алгоритм (последовательность выполняемых продукций), переводящий систему из состояния $S^0$ в состояние $S^f$.

Задача (1.1) связана с определением ситуации (состояния) $S^f$, удовлетворяющей некоторому критерию, которая может быть получена из заданной начальной ситуации.

Задача (1.2) является обратной по отношению к предыдущей.

Задача (1.3) заключается в отыскании алгоритма преобразования начальной ситуации в конечную.

Задача (1 .4) представляет обобщение задач (1 .1) и (1 .3).

Продукции удачно моделируют человеческий способ рассуждений при решении проблем. Поэтому продукции широко используются во многих действующих ЭС.

Продукционные системы впервые изобретены Постом в 1941г. Продукция в системе Поста имеет следующую схему

$$\frac{t_1, t_2, ..., t_n}{t}, \tag{1.5}$$

где $t_1, t_2, ..., t_n$ называются посылками, а $t$ заключением продукции.

Применение схемы (1.5) основывается на подстановке цепочек знаков вместо всех переменных, причем вместо вхождений одной и той же переменной подставляется одна и та же цепочка.

Иерархическое разбиение множества продукций позволяет более эффективно организовать их выполнение, существенно сократив затраты на перебор множества продукций при проверке условий их срабатывания, что определяет дополнительный интерес к продукционным системам.

В рамках этой модели продукция определяется четверкой:

$P = < L, C, N, A >$,

где $L$ – метка; $C$ – условие применимости; $N$ – ядро продукции; $A$ – постдействие.

Следующий пример демонстрирует полную продукцию:

МЕТКА: R26 Использование зонтика

УСЛОВИЕ: ЕСЛИ (идет дождь)

ДЕЙСТВИЕ (ЯДРО ПРОДУКЦИИ): ТО (возьмите зонтик)

ОБЪЯСНЕНИЕ (ПОСТДЕЙСТВИЕ): (зонтик предохраняет от дождя)

Продукция может быть записана также в следующем виде:

$$(i): P; A \Rightarrow B; Q,$$

здесь i – *номер* продукции, с помощью которого данная продукция выделяется из множества продукций. В качестве номера (имени) может выступать некоторая лексема, отражающая суть продукции. $A \Rightarrow B$ – *ядро* продукции. Символ $\Rightarrow$ трактуется как знак логического следования $B$ из истинного $A$. (Если $A$ не истинно, то о $B$ ничего сказать нельзя). $P$ – *условие применимости* ядра продукции (предусловие правила). $Q$ – *постусловие*, которое описывает действия и процедуры, выполняемые после реализации ядра.

Ядро правила продукции интерпретируется по-разному. Обычное прочтение ядра выглядит так: «Если $A$, то $B$» или «Если $A$, то $B$ иначе $C$».

**Пример.**

(1): ЕСЛИ коэффициент рентабельности > 20

ТО рентабельность = «удовлетворительно»

(2) ЕСЛИ задолженность = «нет»

И рентабельность = «удовлетворительно»

ТО финансовое _состояние = «удовлетворительно»

(3): ЕСЛИ финансовое _состояние = «удовлетворительно»

И Репутация = «Удовлетворительно»

ТО надежность предприятия = «удовлетворительно»

Ядра продукции можно классифицировать различными способами. В детерминированных ядрах если выполняется левая часть правила, то правая часть выполняется обязательно. В недетерминированных правилах правая часть правила может не выполняться. В этом случае интерпретация ядра выглядит так: «Если $A$, то возможно $B$». Возможность определяется некоторыми оценками реализации ядра. Например, если задана вероятность выполнения $B$ при актуализации $A$, то продукция может быть такой: «Если $A$, то с вероятностью $P$ реализовать $B$». Оценка может быть и лингвистической, например, «Если $A$, то с большой долей уверенности $B$».

Детерминированные продукции могут быть однозначными и альтернативными. Во втором случае в правой части ядра продукции указываются альтернативные возможности выбора.

**Пример** правил продукции при анализе финансового состояния предприятия.

(1) ЕСЛИ коэффициент ликвидности = «удовлетворительно»

И ликвидность баланса = «удовлетворительно»

ТО платежеспособность = «удовлетворительно»

(2) ЕСЛИ коэффициент ликвидности = «неудовлетворительно»

И ликвидность баланса = «удовлетворительно»

ТО платежеспособность = «удовлетворительно»

(3) ЕСЛИ коэффициент ликвидности = «удовлетворительно»

И ликвидность баланса = «неудовлетворительно»

ТО платежеспособность = «удовлетворительно»

(4) ЕСЛИ коэффициент ликвидности = «неудовлетворительно»

И ликвидность баланса = «неудовлетворительно»

ТО платежеспособность = «неудовлетворительно»

Продукционные системы можно разделить на продукционные системы, управляемые данными (предусловиями правил), и на продукционные системы, управляемые целями (действиями правил). Обычно предусловие (антецедент) задается в виде логической комбинации утверждений о данных рабочей памяти, а действием (консеквентном) является некоторая операция по модификации памяти.

Можно указать следующие достоинства представления знаний в виде продукционных правил:

1. Универсальность метода представления знаний. Это позволяет создавать проблемно-ориентированные экспертные системы.

2. Модульная организации знаний, что обеспечивает независимость правил.

3. Использование предусловий применения правил позволяет использовать наиболее рациональную стратегию вывода, существенно сокращая перебор правил.

Продукционная модель позволяет осуществлять эвристические методы вывода на правилах и может обрабатывать неопределенности в виде условных

вероятностей или коэффициентов уверенности, а также выполнять монотонный или немонотонный вывод. При организации знаний с помощью продукционных моделей в базе знаний содержатся правила продукций, а в базе данных хранится информация о текущем состоянии решаемой задачи. Инициализацию необходимого правила осуществляет интерпретатор.

Основным недостатком данного подхода является его более низкая эффективность по сравнению с методами традиционного программирования.

Продукционное представление знаний соединяет в себе свойства как декларативного, так и процедурного представлений. Эта модель знаний в настоящее время наиболее часть применяется в промышленных экспертных системах. Она привлекает разработчиков своей наглядностью, высокой модульностью, легкостью внесения изменений и дополнений, а также простотой механизма логического вывода.

### *1.2.3. Фреймовая модель представления знаний*

Этот тип моделей представления знаний относится к разновидности объектно-ориентированных. Слово frame на английском языке означает «телосложение», «скелет», «остов», «рама», «каркас». Он был впервые применен Мервином Мински (70-е годы прошлого столетия) в предложенной им системе представления знаний о структуре пространственных (трехмерных) объектов, в качестве абстрактного образа для представления стереотипа объекта, понятия или ситуации. В философии абстракции наделяются свойствами, присущими только определенному классу реальных сущностей. Эти свойства имеют устойчивые названия и прагматику в абстрактных объектах данного класса, но реальным объектам этого класса присущи конкретные значения этих свойств, выраженные количественными или качественными мерами.

Например, абстрактное понятие «мяч» может быть выражено свойствами:

НАЗНАЧЕНИЕ – игровой спортивный снаряд;

ФОРМА В РАБОЧЕМ СОСТОЯНИИ – сфероид вращения;

СОСТОИТ ИЗ – внешней и внутренней оболочек;

РАБОЧЕЕ СОСТОЯНИЕ – заполнен воздухом;

РАБОЧЕЕ ДАВЛЕНИЕ ВО ВНУТРЕННЕЙ ОБОЛОЧКЕ – не более 0.6 атмосфер;

МАТЕРИАЛ ВНЕШНЕЙ ОБОЛОЧКИ – юфть или хром;

МАТЕРИАЛ ВНУТРЕННЕЙ ОБОЛОЧКИ – каучук;

МАССА – не более 1.2 кг.

В этом описании изложены принципиальные свойства спортивного мяча, хотя ясно, что у свойства футбольного мяча будут отличаться от свойств волейбольного и, тем более, – от свойств мяча для игры в регби.

В основе фреймовых моделей представления знаний лежит восприятие фактов через сопоставление полученной информации с конкретными элементами знаний и ограничениями, определенными для каждого конкретного объекта. Структурой, описывающей эти ограничения, и является фрейм. Главное назначение фрейма – быть формой представления концептуальных знаний. Иначе говоря, фрейм – это абстрактный образ для представления, восприятия некоторого стереотипа.

Создание фреймовой модели представления знаний подразумевает предварительную классификацию сущностей предметной области с определением иерархии всех классификационных категорий (концептов). В описаниях концептов указываются только такие их свойства, которые являются общими для всех реальных объектов данной предметной области и характерны для рассматриваемой пространственно-временной ситуации. Такое описание называется концептуальной моделью предметной области и составляет содержание концептуальной компоненты базы знаний интеллектуальной информационной системы. Ситуационная компонента базы знаний содержит факты (уже известные знания) о свойствах реальных объектов предметной области и разного вида бинарных отношениях между ними, соответствующие их состоянию в реальном пространственно-временном измерении.

Фреймом называется структура знаний о свойствах сущности предметной области (концептуального или реального объекта) и ее отношениях к другим

такой же категории сущностям рассматриваемой. При создании фреймовой модели представления знаний применяются два типа фреймов: классификационные фреймы (КФ) и фреймы смысловых связок (ФСС).

Любой фрейм имеет имя, позволяющее однозначно идентифицировать описываемое им понятие (концепт) в пределах базы знаний конкретной ИИС, и содержит описания (имена и текущие значения характеристик) его основных структурных элементов (неделимых частей). Каждое такое описание называется слотом. Имя слота в пределах одного фрейма должно быть уникальным. Слоты могут быть частными (свойственными одному фрейму) и системными, общими для базы знаний конкретной интеллектуальной информационной системы. Имена системных слотов должны быть уникальными в пределах одной базы знаний. Системные слоты предназначаются для выполнения системных процедур редактирования базы знаний и управления выводом. Физически (на машинном носителе) слот состоит из нескольких информационных полей (полей данных), в которых помещаются имена и текущие значения структурных элементов понятия, представленного фреймом. Слоты КФ содержат имена характеристик (признаков) объекта и их значений, а слоты ФСС – названия типов бинарных связей данного объекта (концепта) с другими объектами предметной области и количественные или качественные меры выражения этих связей. Совокупность КФ и ФСС служит основой для построения фрейм-фраз, образующих семантическую модель предметной области. В основе фреймовой модели представления знаний лежит восприятие фактов посредством сопоставления любой информации о новом объекте с конкретными элементами концептуальных знаний (то есть, знаний о свойствах концепта, который по совокупности признаков может быть взят в качестве прототипа). Слот может содержать не только конкретные значения характеристик, выраженные числами или качественными категориями. Вместо этого в нем может быть указана процедура вычисления значения этой характеристики на основе значений других слотов данного фрейма или ссылка

на определенные слоты других фреймов, или ссылка на другой фрейм. Иначе говоря, значением слота может быть любая информация, которая может понадобиться как для описания концептов и реальных объектов, так и для определения (заимствования или вычисления) значений их характеристик. Если в качестве значения слота одного фрейма указано имя другого фрейма, то образуется сеть фреймов, а роль такого слота в модели представления знаний – отобразить отношение между фреймами.

Возможность наследования одними объектами свойств других объектов является одной из главных особенностей фреймовой модели представления знаний. Эта возможность крайне важна в ситуациях, когда для представления новых состояний объекта используются базовые фреймы. Средства наследования свойств позволяют в различных фреймах использовать одинаковые фрагменты базовых фреймов. Для реализации процедуры наследования применяются специальные ссылки – указатели наследования. Эти указатели присутствуют в структуре фрейма лишь тогда, когда в модели представления знаний предполагается использовать отношение «абстракция – конкретность» в определении объекта. Так как между различными концептами существуют некоторые аналогии или ассоциативные связи, то образуется иерархическая структура с классификационными обобщающими свойствами. Эта структура отображает отношения типа «часть – целое». В отношениях этого типа наследование атрибутов (свойств) объектов запрещено. Указатели наследования показывают, какую информацию, составляющую содержание слота-донора (он, как правило, принадлежит фрейму более высокого уровня иерархии или базовому фрейму), наследует одноименный с ним слот данного фрейма. В конкретных системах указатели наследования именуются и процедурно организуются по-разному.

Есть несколько способов получения слотом того или иного значения во фрейме-экземпляре:

по умолчанию – от одноименного слота фрейма-прототипа;

наследованием свойства (значения) от фрейма, указанного в слоте АКО (аббревиатура, применяемая в качестве стандартного имени слота, значением которого является имя фрейма-родителя);

по формуле, указанной в слоте;

через присоединенную процедуру;

через диалог с пользователем;

из внешней, информационно сопряженной базы данных.

Наследование свойств происходит с помощью, так называемых АКО-связей. Слот АКО адресует «свой» фрейм к фрейму более высокого уровня иерархии, откуда неявно переносятся (наследуются) значения одноименных слотов. В общем случае правило наследования свойств определяется специальными кодами, которыми снабжается наследуемое значение:

*U* – код уникального значения, наследование которого запрещено;

*S* – код свободно наследуемого значения;

*R* – код значения, наследуемого в пределах, указанных в одноименном слоте фрейма-родителя;

*O* – дополнительный код, разрешающий наследование уникального значения и запрещающий наследование значения, указанного предельными границами.

Кроме указателей наследования слот содержит ссылку, именуемую указатель типа данных. Она используется для выражения значения слота в виде прямого указания типа данных (вещественные, целые, булевы, тексты, списки, табличные и т.п.) или указывают имя фрейма, в котором есть одноименный слот, тип данных которого следует заимствовать. Значение слота должно соответствовать указанному типу данных или условию наследования.

Специфичен по своему назначению слот, называемый «демон». Он представляет собой процедуру, которая должна выполняться автоматически при обращении к данному слоту. Например, при идентификации какого-либо объекта значение одного из слотов может быть не определено, но есть ссылка на соответствующий слот-демон. Демон типа IF-NEEDED задает процедуру вычисле-

ния отсутствующего значения или отыскания этого значения в слотах других фреймов. Демон типа IF-ADDED запускается при изменении значения слота, а демон типа IF-REMOVED предназначен для удаления значения слота.

В качестве значения слота, определяющего его роль, иногда используется специальная процедура, называемая присоединяемой. Она запускается в соответствии с предписанием, которое может быть получено даже из другого фрейма. В связи с этой особенностью присоединенной процедуры ее можно рассматривать как процедуру управления выводом. Это обстоятельство говорит о возможности использовать фреймы в продукционных моделях.

Обычно при фреймовой структуре представления декларативная компонента базы знаний состоит из двух частей: базовые знания о концептуальных объектах и отношениях между ними, помещаемые во фреймах- прототипах (фреймах-образцах или базовых фреймах), и текущие знания о реальных объектах, помещаемые во фреймах-экземплярах.

Демоны, процедуры наследования и присоединенные процедуры в совокупности образуют процедурную компоненту базы знаний, созданной на основе фреймовой модели представления знаний. Таким образом, фреймовая структура представления знаний представляет декларативные и процедурные знания, не разделяя их. Операции, входящие в состав процедур наследования, демонов и присоединенных процедур, позволяют реализовать любой из известных механизмов логического решения.

При введении в базу знаний нового понятия для него создается фрейм-прототип и, если необходимо – шаблон фрейма-экземпляра. Фрейму-прототипу присваивается имя, называются все его слоты и некоторые из них заполняются значениями, свойственными понятию-концепту. Если в базу знаний необходимо ввести данные о конкретных экземплярах, то заполняются фреймы – экземпляры, в которых будут указаны только свойственные конкретным экземплярам значения слотов, общие для понятия-концепта свойства – наследуются. Сложные понятия представляются иерархической системой фреймов, а простые – одним фреймом.

Фреймовая структура хорошо соответствует представлению всего многообразия отношений, возможных между объектами реального мира. Фрейм эффективно моделирует фрагмент сетевой структуры, состоящей из простых и сложных объектов, представленных узлами (вершинами), и дуг, соединяющих вершины и отображающих бинарные отношения между объектами. Если в качестве значения слота будет выступать имя другого фрейма, то образуется сеть фреймов. Эту сеть можно рассматривать как семантическую сеть с блочной структурой, позволяющей реализовать альтернативные интерпретации предметных областей (представление предметной области в разных прагматических аспектах). Фреймы такой сети, соединяя процедурные и декларативные знания, обеспечат любое преобразование информации как внутри фрейма, так и в его связях с другими фреймами. Наличие бинарных отношений разного характера делают такую сеть достаточно мощной в семантическом отношении.

Модель фрейма является достаточно универсальной, поскольку позволяет отобразить все многообразие знаний о мире, используя:

фреймы-структуры, применяемые для обозначения объектов и понятий (заем, залог, вексель);

фреймы-роли (менеджер, кассир, клиент);

фреймы-сценарии (банкротство, собрание акционеров, празднование именин);

фреймы-ситуации (тревога, авария, рабочий режим) и др.

Итак, фрейм можно представить в виде следующей конструкции:

$$F = \left[< r_1, v_1 >, < r_2, v_2 >, ..., < r_n, v_n >\right],$$

где $F$ – имя фрейма; $r_i$ – имя слота; $v_i$ – значение слота.

Следовательно, структуру фрейма можно представить как список свойств:

(ИМЯ ФРЕЙМА:

(имя 1-го слота: значение 1-го слота),

(имя 2-го слота: значение 2-го слота),

…

(имя N-го слота: значение N-го слота).

Существуют два способа организации фреймовых знаний: целевые структуры и сценарии.

**Целевые структуры** – это ролевые фреймы, в которых в качестве имен слотов выступают вопросительные слова, ответы на которые являются значениями слотов. Например, фрейм деловая поездка=[<кто,*x*>,<куда,*y*>, <когда,*z*>,<цель,*w*>,<с кем,*l*>,<вид транспорта,*r*>]. Среди ролей выделяются *главные роли*, заполнение которых обязательно.

Большое значение имеют стереотипные знания, которые описывают знания о стандартных ситуациях реального мира. Такие знания называют понятием. Система понятий рассматривается как неформальное знание о проблемной области. В противоположность этому фрейм – формализованное знание о стереотипной ситуации. Для планирования действий в таких ситуациях используются фреймы-сценарии.

**Сценарий** это формализованное описание стандартной последовательности взаимосвязанных фактов, определяющих типичную ситуацию в проблемной области. Это может быть последовательность действий или процедур, описывающих способы достижения цели.

При реализации фреймового представления знаний *различают фреймы-образцы* или прототипы, хранящиеся в базе знаний, и *фреймы-экземпляры*, которые создаются для отображения реальных фактических ситуаций на основе поступающих данных. В *прототипах* позиции, соответствующие значениям слотов, не заполнены.

Пример фрейма-прототипа: Имя фрейма: руководитель Является: экономист

Имя:_____

Возраст:_____

Адрес:_____

Слот «*является»* служит для построения иерархии фреймов. Его значение указывает, что фреймом более высокого уровня служит фрейм «экономист» Следовательно, фрейм «руководитель» наследует все свойства фрейма «экономист».

На месте значений слотов *фрейма-прототипа* могут быть указаны условия заполнения слотов.

Пример фрейма-прототипа с условиями заполнения слотов:

Имя фрейма: руководитель

Является: экономист

Имя: агрегат (фамилия, имя)

Возраст: агрегат (годы)

Адрес: адрес.

Значением слота может быть что угодно: числа или математические зависимости, тексты на естественном языке, программы, ссылки на другие

слоты данного фрейма или слоты других фреймов. В качестве значения могут выступать слоты более низкого уровня. Это позволяет организовать иерархическую подчиненность фреймов. Во фрейме-экземпляре значения

слотов заполнены.

Имя фрейма: VIP1

Является: экономист

Имя: Петров И.И.

Возраст: 40

Адрес: адрес_1 (указатель на экземпляр фрейма)

Существует несколько способов получения слотом значений во фрейме-экземпляре:

по умолчанию от фрейма-образца (Default-значение);

через наследование свойств от вышележащего фрейма;

по формуле, указанной в слоте;

явно из диалога с пользователем;

из базы данных.

С каждым слотом можно связать процедуру, которая выполняется автоматически при изменении значений слотов. Это процедуры следующих типов:

- IF ADDED – запускается при подстановке в слот значения;

- IF NEEDED – вызывается, когда требуется значение слота, но он пуст;
- IF REMOVED – выполняется при удалении информации из слота.

На рис. 1.1 приведен пример фрагмента иерархической структуры базы знаний, основанной на фреймовом представлении информации. Фрагмент представлен четырьмя фреймами, три из которых наследует какие-то свойства других («вышестоящих») фреймов, представляющих модели понятий более высоких уровней иерархии. Эти три фрейма имеют слоты, значения которых наследуются из других слотов (слоты «Ако»), и слоты, поименованные символами С1, … , С5. При этом, очевидно, что в разных фреймах эти символы означают разные признаки понятий, соответствующих тому или иному фрейму. Например, для фрейма «Прозаическое литературное произведение» признаки С1, С2 и С3 означают возможные разновидности прозаического литературного произведения («роман», «повесть», «рассказ»), а для фрейма «Война и мир» – соответственно признаки «автор», «год написания» и «год издания».

Наследуемые значения также различны. Например, фрейм «Роман» наследует от вышестоящего фрейма его название в качестве значения своего признака («Ако»), название которое призвано ответить на вопрос: что такое «роман», как концептуальная категория. Что касается фрейма «Война и мир», то он наследует от вышестоящего фрейма уже два значения, одно из которых говорит о том, что «Война и мир» – это роман, а другое – что это эпический роман.

Следует заметить, что фреймовая структура представления знаний имеет довольно широкое применение. Это во многом объясняется тем, что она естественно воспринимается человеком, поскольку адекватно ассоциативно – адресной организации памяти и механизму поиска информации. И все же, несмотря на то, что методология и инструментарий реализации фреймового представления знаний разработаны достаточно хорошо, процесс создания «рабочей» системы требует участия специалистов высокой квалификации и связан с немалыми затратами финансовых средств и времени.

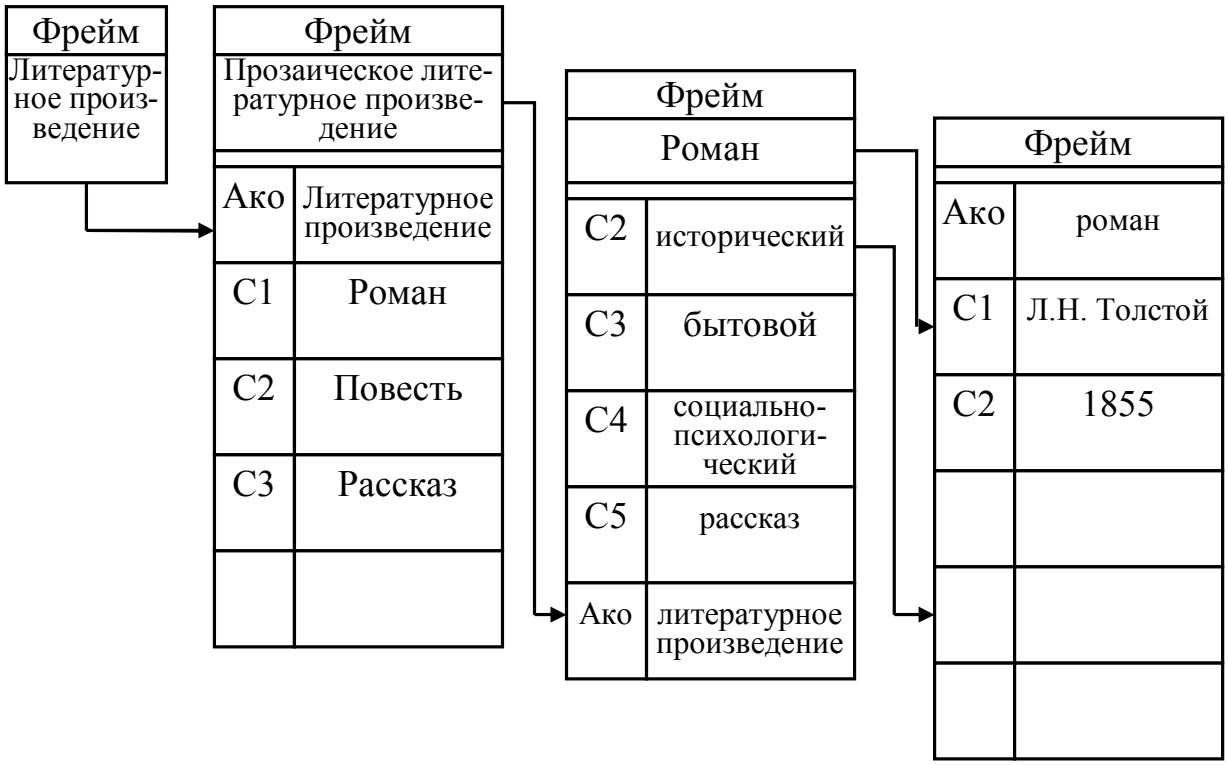



*Рис.1.1. Фрагмент иерархической структуры с фреймовым представлением знаний*

Фреймовая структура представления знаний хорошо согласуется с возможностями объектно-ориентированных языков программирования, как в части описания объектов, так средств описания процедур обработки таких описаний. Но существуют и специальные средства для программирования фреймовых моделей.

### *1.2.4. Сетевые модели представления знаний*

Эвристические модели представления знаний в отличие от логических моделей предоставляют более широкие возможности для описания сложных структур. Это достигается выделением и включением в модель в явной форме всех отношений, которые образуют информационную структуру, а также семантики этих отношений (поэтому сетевые модели называют еще семантико-сетевыми).

На греческом языке слово «semantikos» означает «обозначающий». В лингвистике семантика, как и семасиология, – это раздел языкознания, изучающий значение единиц языка (слов и устойчивых словосочетаний). В совре-

менной информатике часто используют это понятие в качестве признака, устанавливающего смысловые отношения между символами и объектами, которые обозначаются этими символами.

В некоторых источниках семантические и прагматические отношения в семиотике определяются как проблемы различных функциональных уровней. При этом полагают, что семантика означает общие отношения между символами и объектами, которые представлены этими символами, а прагматика – только выразительные (охватывающие) отношения между символами и создателями этих символов или людьми, применяющими эти символы. Изначально семантическая сеть (СС) была задумана как модель, представляющая структуру долговременной памяти человека и высших животных, и применялась в психологии. С помощью СС изучались семантические связи, возникающие на основе ассоциаций в системе поведенческого обучения человека.

Семантико-сетевая структура представления знаний подразумевает создание информационной модели предметной области, топологически напоминающую некую сеть или граф. Элементами конструкции семантической сети являются узлы и дуги, соединяющие дуги. Сама сеть может быть упорядоченной, то есть представлять собой ориентированный граф, или неупорядоченной, то есть представлять собой не ориентированный граф. Узлы такой сетевой структуры отображают значимые понятия (сущности) предметной области. Без рассмотрения хотя бы одного из таких объектов информационная модель не имеет смысла. Понятия могут иметь статус концептов (принципиальных, устойчивых смысловых категорий предметной области) или атрибутов. Они могут быть простыми (неразделимыми) или сложными, состоящими из нескольких простых понятий. Различного характера связи между понятиями выражаются с помощью бинарных (которые связывают пары объектов) или *N*- арных (которые связывают *N* объектов) отношений. Связи (а значит, и выражающие их отношения) обозначаются дугами, соединяющими узлы сети. В качестве понятий предметной области могут выступать абстракции или реальные объекты. Ха-

рактер связей между понятиями весьма многообразен. Чем больше типов различных связей между понятиями отображает сеть, тем выше ее семантика. Но большое число связей, естественно, делает структуру сети более сложной как в аспекте описания, так и в аспекте обработки (анализа) этих связей.

С позиций логики элемент сети, представленный двумя объектами, связанными неким числом бинарных отношений, можно рассматривать как эквивалент совокупности двухместных предикатов, число которых равно числу отображаемых бинарных отношений между объектами.

Таким образом, *семантическая сеть* – это ориентированный граф, в котором вершины соответствуют объектам и понятиям, а дуги отражают семантические отношения между вершинами.

Выделим в проблемной области конечное множество атрибутов или символов $A = \{A_1, A_2, ..., A_n\}$ и конечное множество отношений $R = \{R_1, R_2, ..., R_m\}$. Для любого отношения $R_i$, принадлежащего семантической сети, определим два понятия:

*интенсионал* или схема отношения $INT(R_i)$;

*экстенсионал* или вариант значения $EXT(R_i)$.

Интенсионал определяется как совокупность пар следующего вида $INT(R_i) = \{..., \lfloor A_j, DOM(A_j) \rfloor, ...\}$ где $R_i$ – имя отношения; $A_j \in A(j = 1, ..., n)$ – атрибуты отношения; $DOM(A_j)$ – множество значений. Базовое множество, на котором задаются отношения $R_i$, есть объединение всех доменов.

Экстенсионал определяется следующим образом: $EXT(R_i) = \{F_1, F_2, ..., F_n\}$, где $F_i$ – факт отношения, который задается парой *атрибут=значение*.

Таким образом, интенсиональная семантическая сеть описывает предметную область на обобщенном, концептуальном уровне, в то время как в экстенсиональной сети производятся конкретизация и наполнение сети фактическими данными.

В графической интерпретации факт – это подграф семантической сети, имеющий звездообразную структуру. Корень – вершина предикатного типа. Из вершины факта выходят ребра или дуги, помеченные именами атрибу-

тов. Ребра ведут к вершинам базового множества, которые являются значениями этих атрибутов.

Одним из подходов к созданию базы знаний на основе семантико-сетевого представления знаний с выделением ее экстенсиональной (Э) и интенсиональной (И) частей. При этом Э-компонента сети является частью единой базы данных, хранящей описания реальных объектов и их характеристик, а И-компонента сети – описанием концептуальных знаний об основных понятиях предметной области, их свойствах и отношениях между ними.

Характерной особенностью семантических сетей является обязательное наличие в их описаниях следующих типов отношений:

ЧАСТЬ – ЦЕЛОЕ;

РОД – ВИД;

ПРИЧИНА – СЛЕДСТВИЕ;

КЛАСС – ЭЛЕМЕНТ КЛАССА;

СВОЙСТВО – ЗНАЧЕНИЕ СВОЙСТВА;

ЭЛЕМЕНТ КЛАССА – ОБРАЗЕЦ ЭЛЕМЕНТА.

Пример простой семантической сети представлен на рис. 1.2.

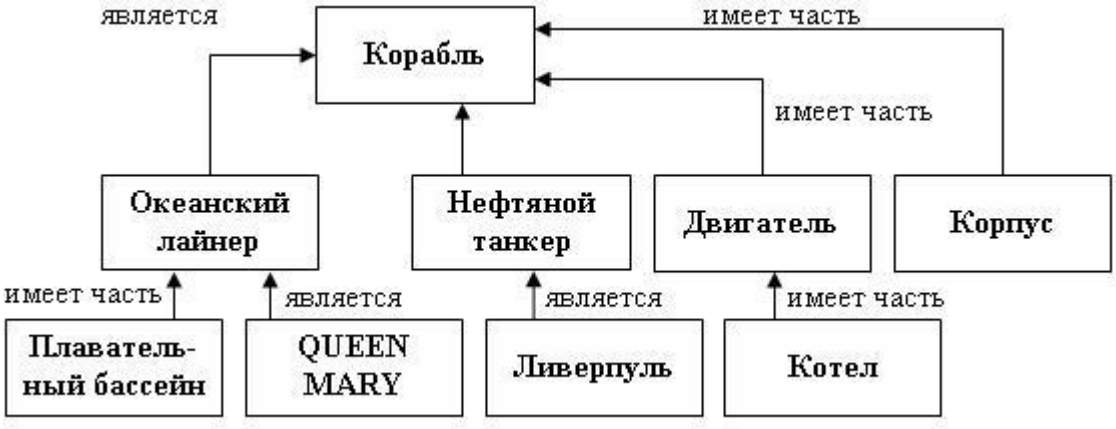



*Рис. 1.2. Пример семантической сети*

В зависимости от класса задач, для решения которых создается интеллектуальная информационная система, в семантико-сетевой модели представления

знаний может оказаться целесообразным учет и других типов связей прагматического, синтагматического или парадигматического характера, например:

количественные сравнения (больше, меньше, равно);

функциональные связи (производит, влияет, взаимодействует);

взаимное расположение в пространстве (около, за, под, над, вдали от);

связи, определяющие временные расстояния между событиями (раньше, позже, одновременно).

В семантико-сетевых моделях представления знаний декларативные знания не отделены от процедурных. А это означает, что база знаний должна содержать и набор правил, с помощью которых может быть сформирована процедура логического вывода, обеспечивающая решение задачи, содержащейся в запросе к системе.

Семантические сети широко и успешно применяются при решении задач распознавания образов, в системах управления различного рода сложными объектами, а в последнее время – и в экспертных системах.

**Пример**. Пусть задано базовое множество понятий {налог, банк, клиент, итог}. Множество отношений *R* ={платежный документ, выплата}.

*INT*(платежный документ)={[1 слагаемое,(итог, налог)],[2 слагаемое,(итог, налог)],[платежный документ,(итог, банк, клиент)]}.

*INT*(выплата)= {[с налогом, (банк, клиент)], [без налога, (клиент, итог)]}/

Экстенсионалы этих отношений можно записать в виде фактов:

*F1*: (платежный документ, 1 слагаемое=итог, 2 слагаемое=итог, платежный документ=2*итог);

*F2*: (платежный документ, 1 слагаемое=итог, 2 слагаемое=налог, платежный документ=банк);

*F3:* (платежный документ, 1 слагаемое=итог, 2 слагаемое=налог, платежный документ=клиент);

*F4*: (платежный документ, 1 слагаемое=налог, 2 слагаемое=налог, платежный документ=банк);

*F5*: (выплата, с налогом=банк, без налога=клиент).

*EXT*(платежный документ) = {F1, F2, F3, F4}.

*EXT*(выплата) = {F5}.

Семантическая сеть – это совокупность $\{\ldots[INT(R_i)EXT(R_i)]\ldots\}$ записывается в виде *ассоциативной* структуры данных (рис. 1.3).

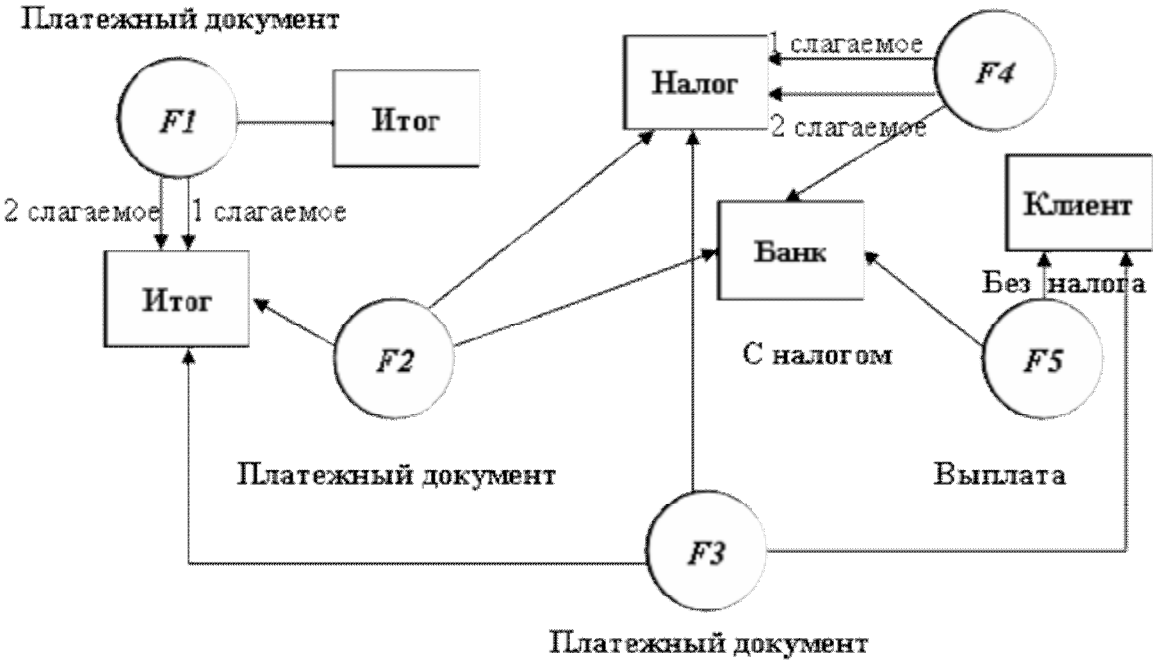



*Рис. 1.3. Семантическая сеть*

В семантической сети используются различные типы отношений, но свойство ассоциативности, т.е. группировки информации вокруг фактов, атрибутов и/или объектов, являются характерными.

В семантических сетях применяются три типа объектов: понятие, событие, свойство. *Понятия* – сведения об абстрактных или физических объектах проблемной области. *События*_– действия, происходящие в реальном мире. Для них указываются тип действия и роль, которую играют объекты в этом действии. *Свойства* используются для уточнения понятий, событий или других свойств. Применительно к понятиям свойства описывают их характеристики (цвет, качество и т.д.). Применительно к событиям: место, время и т.д.

Семантические отношения делятся на лингвистические, логические, теоретико-множественные, квантифицированные.

Наиболее распространенными являются *лингвистические* отношения.

Они отображают смысловую взаимосвязь между событиями, между событиями и понятиями или свойствами. К ним относятся, например, *агент* – отношение между событием и тем, что или кто его вызывает; *объект* – отношение между событием и тем, над чем или кем производится действие; *условие* – отношение, показывающее логическую зависимость между событиями; *инструмент* – объект, с помощью которого совершается событие.

*Логические* отношения – операции, используемые в исчислении высказываний.

*Теоретико-множественные* отношения – это отношение подмножества *SUB* и супермножества *SUP*, элемент множества, отношение части и целого, отношение множества и элемента (is-a, part-of). Этот класс отношений не обладает высокой степенью интеллектуальности, но позволяет строить иерархические структуры. В таких структурах все свойства *SUB*-понятий автоматически присваиваются SUPерпонятиям.

*Квантифицированные* отношения используются для представления знаний декларативного типа.

Основой для определения любого понятия является множество его отношений с другими понятиями, обязательными отношениями являются:

класс, которому принадлежит данное понятие;

свойства, выделяющие понятие из всех понятий данного класса;

примеры (экземпляры) данного понятия.

При представлении событий предварительно выделяются простые отношения, которые характеризуют основные компоненты события. В первую очередь из события выделяется *действие*, которое определяется глаголом. Затем определяются *объекты*, которые *действуют,* и *объекты*, *над* которыми выполняются действия.

Все связи понятий, событий и свойств с действиями называют *падежами*, или падежными отношениями, которые относятся к классу лингвистических отношений. Основные падежи приведены в таблице 1.2.

Семантические сети нашли применение в основном системах обработки естественного языка, частично в вопросно-ответных системах, а также в системах

искусственного видения. В последних семантические – сети используются для хранения знаний о структуре, форме и свойствах физических объектов.

***Таблица 1.2***

| **Падеж** | **Отношение, определяющее связь действия с:** |
|---|---|
| агент | предметом, являющимся инициатором действия |
| объект | предметом, подвергающимся действию |
| источник | размещение предмета перед действием |
| приемник | размещение предмета после действия |
| время | моментом выполнения действия |
| место | местом проведения действия |
| цель | действием другого события |

Особенность семантической сети как модели знаний состоит в единстве базы знаний и механизма вывода решения. При формировании запроса к базе знаний строится семантическая сеть, отображающая структуру запроса. Вывод обеспечивается за счет сопоставления общей структуры сети базы знаний и подсети для запроса.

Рассмотрим пример семантической сети, отражающий подчиненность сотрудников организации. На сети отображены факты (рис. 1.4):

Петров is-a сотрудник.

Сотрудник is-a человек.

Сидоров is-a сотрудник.

Руководит объект Сидоров.

Руководит агент Петров.

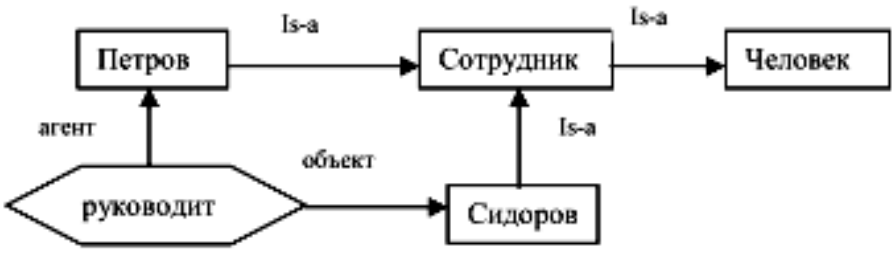



*Рис. 1.4. Семантическая сеть, отображающая подчиненность сотрудников организации*

Пусть имеем запрос: «Кто руководит Сидоровым?» Этот запрос представляется в виде подграфа (рис. 1.5).

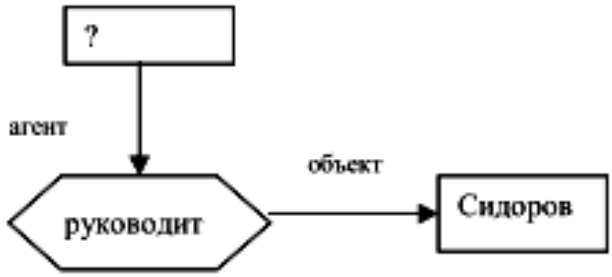



*Рис. 1.5. Подграф поиска*

Сопоставление общей сети с подсетью запроса начинается с поиска вершины «руководит», имеющей ветвь «объект», направленную в вершину «Сидоров». Затем происходит переход по ветви агент, что и приводит в ответу «Петров».

Итак, для поиска решений используется поиск по образцу, который сводится к изоморфному вложению подграфа запроса в семантическую сеть.

Недостатки представления знаний в виде семантических сетей вытекают из их достоинств:

имеют произвольную структуру, что усложняет их построение и модификацию;

разнообразие типов вершин и связей между ними требует разнообразия процедур их обработки.

В ряде ИС используются комбинации сетевых и продукционных моделей представления знаний. В таких моделях декларативные знания описываются в сетевом компоненте модели, а процедурные знания – в продукционном. В этом случае говорят о работе продукционной системы над семантической сетью.

### *1.2.5. Сценарии*

Особую роль в системах представления знаний играют стереотипные знания, описывающие известные стандартные ситуации реального мира. Такие знания позволяют восстанавливать информацию, пропущенную в описании ситуации, предсказывать появление новых фактов, которых можно ожидать в

данной ситуации, устанавливать смысл происхождения ситуации с точки зрения более общего ситуативного контекста.

Для описания стереотипного знания используются различные модели. Среди них наиболее распространенными являются сценарии.

*Сценарий* – это структурированное представление, описывающее стереотипную последовательность событий в частном контексте. Сценарии используются в системах понимания естественного языка для организации базы знаний в терминах ситуаций, которые система должна понимать. Объекты в этих сетях связаны единственным отношением строгого или нестрогого порядка с различной семантикой (например, «причина-следствие», «цель-подцель», «часть-целое»). Каждая последовательность действий в сценах обладает свойством каузальных цепочек: всякое предшествующее действие создает условия для совершения последующего действия.

На рис. 1.6 приведен пример сценария, в котором в качестве связей между вершинами сети выступает причинно-следственное отношение.

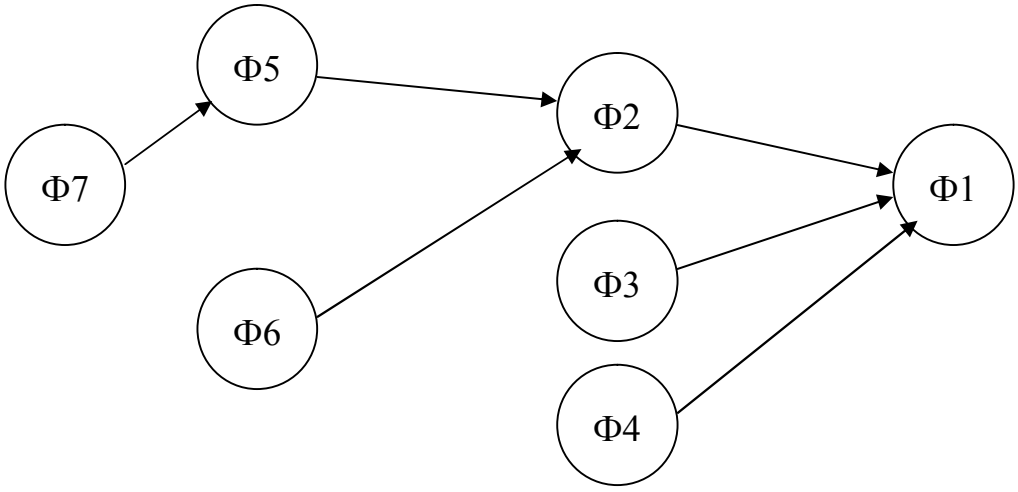



*Рис. 1.6. Пример сценария*

Вершины сети задают следующие факты: Ф1 – станок простаивает; Ф2 – на рабочем месте нет рабочего; Ф3 – станок неисправен; Ф4 – в цехе нет заготовок; Ф5 – обеденный перерыв; Ф6 – рабочий покинул станок в рабочее время; Ф7 – рабочий находится в столовой.

Сценарий включает следующие компоненты.

*Начальные условия*, которые должны быть истинными при вызове сценария.

*Результаты* или факты, которые являются истинными, когда сценарий завершается.

*Предположения*, которые поддерживают контекст сценария. Множество предположений описывают принятые по умолчанию условия реализации сценария.

*Роли* являются действиями, которые совершают отдельные участники.

*Сцены*. Сценарий состоит из последовательности сцен, каждая из которых представляет временные аспекты сценария.

Сценарии рассматриваются как средство представления проблемно-зависимых каузальных знаний и задаются в виде фреймоподобных списочных структур.

<сценарий: название

роли: список ролей (участники сценария)

цель: ключевое событие сценария, посылки и следствия ключевого события

сцена 1: наименование сцены

(ход сцены)

сцена 2: наименование сцены

(ход сцены) (место, время проведения сценария, побочные эффекты)

Сцена N: наименование сцены

(ход сцены)>.

Выбор модели представления знаний определяется с учетом разных факторов. Основными из них являются:

специфика предметной области (сущностный характер объектов, динамика ситуаций, стабильность свойств объектов, объективность предпосылок к появлению новых объектов);

класс решаемых интеллектуальных задач;

характер деятельности, для обеспечения которой создается интеллектуальная информационная система (теоретические исследования, прикладные научные исследования, производственная деятельность, управление динамической системой и пр.);

имеющийся опыт применения или проектирования конкретных категорий интеллектуальных информационных систем (систем, основанных на правилах, систем, основанных на примерах (аналогиях), или гибридных систем).

### *1.2.6. Представления знаний на основе нечеткой логики*

Математическая теория нечетких множеств (fuzzy sets) и нечеткая логика (fuzzy logic) являются обобщениями классической теории множеств и классической формальной логики. Данные понятия были впервые предложены американским ученым Лотфи Заде (Lotfi Zadeh) в 1965 г. Основной причиной появления новой теории стало наличие нечетких и приближенных рассуждений при описании человеком процессов, систем, объектов.

Предпосылкой разработки теории нечетких множеств является принцип несовместимости, согласно которому сложность системы и точность, с которой ее можно описать традиционными математическими методами, по своей сути противоречивы друг другу.

Методы нечеткой логики позволяют строить логико-лингвистические модели, отражающие общую смысловую постановку задачи, используя качественные представления соответствующие «человеческим» способам рассуждений и принятия решений.

Рассмотрим основные понятия теории нечетких множеств. Пусть $A$ – некоторое подмножество универсального множества $E$, а $x$ – элемент множества $E$. В обычной (четкой) теории множеств функция принадлежности элемента x подмножеству A может принимать два значения: $\mu_A(x) = 1$, если $x \in A$ и $\mu_A(x) = 0$, если $x \notin A$. В теории нечетких множеств функция принадлежности элемента x подмножеству $A$ может принимать любые значения на отрезке [0, 1] т.е. $\mu_A(x) \in [0, 1]$, при этом подмножество $A$ называют нечетким.

Рассмотрим поясняющий пример.

Пусть $E=R^1$ – множество действительных чисел, тогда нечеткие множества: «большой отрицательный», «приблизительно нулевой», «большой по-

ложительный» могут быть определены функциями принадлежности, приведенными на рис. 1.7.

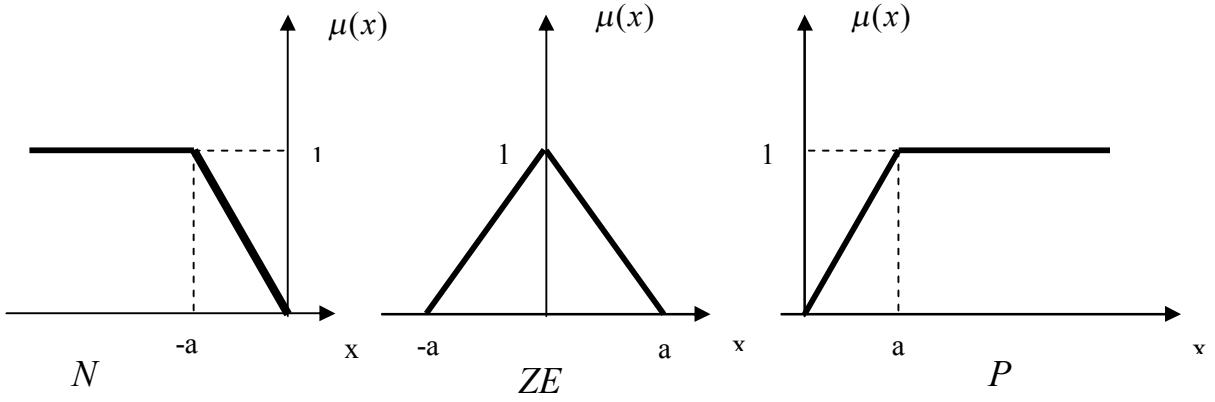



*Рис. 1.7. Функции принадлежности нечетких множеств: «большой отрицательный» (N), «приблизительно нулевой» (ZE), «большой положительный» (P)*

Для определения вида функций принадлежности разработаны различные экспертные методы. В ряде случаев используют типовые формы функций принадлежности, тогда методом экспертных оценок определяется тип функций принадлежности и их параметры.

Приведем некоторые типовые виды функций принадлежности.

Кусочно-линейные функции принадлежности могут описываться уравнениями:

$$\mu_Z(x) = \begin{cases} 1 - |x-a|/\lambda, \text{ при } |x-a| \le \lambda, \\ 0, \text{ при } |x-a| > \lambda, \end{cases} \quad (1.6,\text{а})$$

$$\mu_N(x) = \begin{cases} 1 - \mu_Z(x), \text{ при } x \le 0, \\ 0, \text{ при } x > 0, \end{cases} \quad (1.6,\text{б})$$

$$\mu_P(x) = \begin{cases} 1 - \mu_Z(x), \text{ при } x \ge 0, \\ 0, \text{ при } x < 0. \end{cases} \quad (1.6,\text{в})$$

На рис. 1.8 приведен вид графиков функций принадлежности а) – (1.6,б), б) – (1.6,а) и в) – (1.6,в) соответственно, при $a = 0$ и $\lambda = 1$.

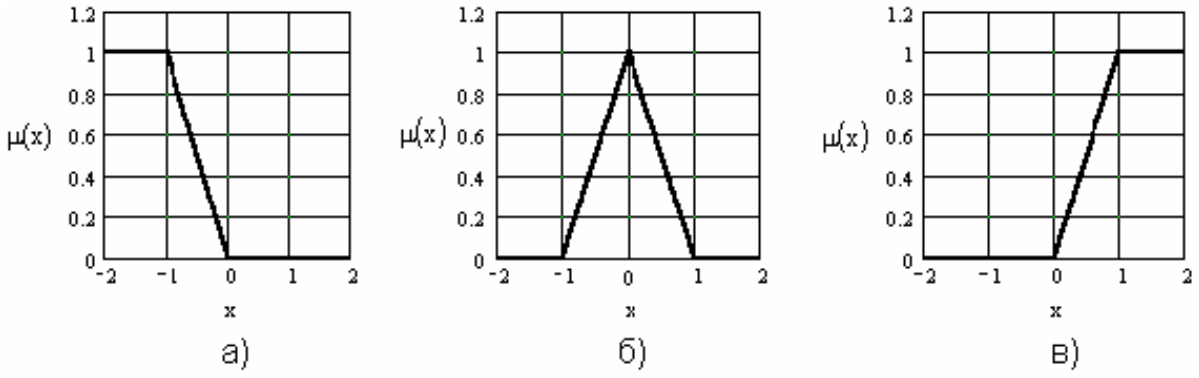



*Рис. 1.8. Кусочно-линейные функции принадлежности*

Показательные (Пуассона) функции принадлежности описываются уравнениями:

$$\mu_Z(x) = e^{-|x-a|/\lambda}, \tag{1.7,а}$$

$$\mu_N(x) = \begin{cases} 1-\mu_Z(x), \textit{при } x \le 0, \\ 0, \textit{при } x > 0, \end{cases} \tag{1.7,б}$$

$$\mu_P(x) = \begin{cases} 1-\mu_Z(x), \textit{при } x \ge 0, \\ 0, \textit{при } x < 0. \end{cases} \tag{1.7,в}$$

На рис. 1.9 приведен вид графиков функций принадлежности а) – (1.7,б), б) – (1.7,а) и в) – (1.7,в) соответственно, при $a = 0$ и $\lambda = 1$.

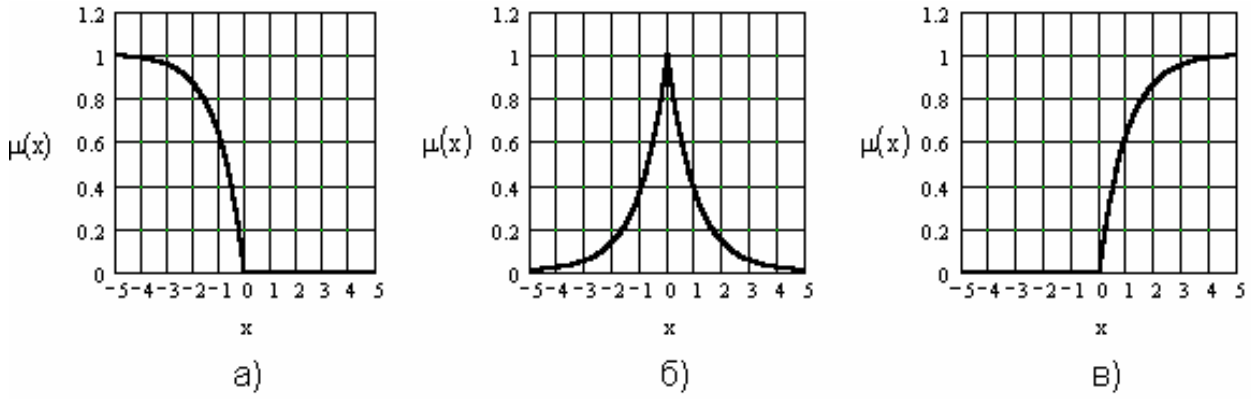



*Рис. 1.9. Показательные функции принадлежности*

Гауссовы функции принадлежности описываются уравнениями:

$$\mu_Z(x) = e^{-(x-a)^2/\lambda}, \tag{1.8,а}$$

$$\mu_N(x) = \begin{cases} 1 - \mu_Z(x), \textit{ при } x \le 0, \\ 0, \textit{ при } x > 0, \end{cases} \qquad (1.8,б)$$

$$\mu_P(x) = \begin{cases} 1 - \mu_Z(x), \textit{ при } x \ge 0, \\ 0, \textit{ при } x < 0. \end{cases} \qquad (1.8,в)$$

На рис. 1.10 приведен вид графиков функций принадлежности а) – (1.8,б), б) – (1.8,а) и в) – (1.8,в) соответственно, при $a = 0$ и $\lambda = 1$.

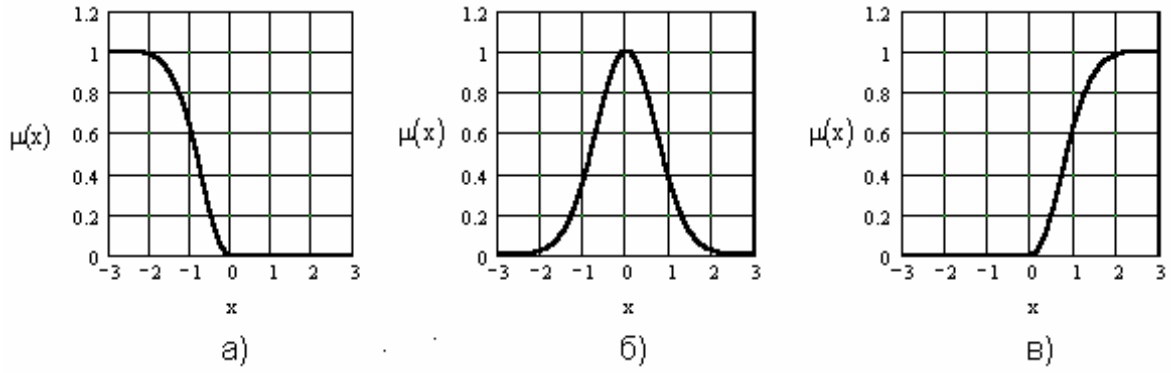



*Рис. 1.10. Гауссовы функции принадлежности*

По аналогии с обычной теорией множеств в теории нечетких множеств вводятся логические операции над множествами.

Пусть $A$ и $B$ – нечеткие множества на универсальном множестве $E$.

1. Равенство: $A = B$.

$A$ и $B$ равны, если $\forall\ x \in E \quad \mu_A(x) = \mu_B(x)$

2. Дополнение (отрицание): $A = \overline{B}$.

Обычно принимают, что операция, осуществляемая над функцией принадлежности при нахождении дополнения $\mu_A(x) = n[\mu(x)]$ должны обладать свойствами: 1) $n[0] = 1$ и $n[1] = 0$; 2) $n[\bullet]$ – невозрастающая функция.

Чаще всего функция дополнения определяется следующим образом:

$\forall x \in E \quad \mu_A(x) = 1 - \mu_B(x)$,

очевидно в этом случае $\overline{A} = B$ и $\overline{B} = A$.

3. Пересечение: $A \cap B$ – наибольшее нечеткое подмножество, содержащее одновременно $A$ и $B$.

Обычно функции принадлежности операции пересечения определяются в классе $T$-норм, удовлетворяющих следующим свойствам:

1) $T(0,0)=0$; $T(\mu_A,1)=\mu_A$; $T(1,\mu_A)=\mu_A$ – ограниченность;

2) $T(\mu_A,\mu_B)\le T(\mu_C,\mu_D)$, если $\mu_A\le\mu_C$ и $\mu_B\le\mu_D$ – монотонность;

3) $T(\mu_A,\mu_B)=T(\mu_B,\mu_A)$ – коммутативность;

4) $T(\mu_A,T(\mu_B,\mu_C))=T(T(\mu_A,\mu_B),\mu_C)$ – ассоциативность.

Простейшие примеры операции пересечение:

$$\mu_{A\cap B}=\min(\mu_A,\mu_B); \tag{1.9}$$

$$\mu_{A\cap B}=\mu_A\cdot\mu_B; \tag{1.10}$$

$$\mu_{A\cap B}=\max(0,\mu_A+\mu_B-1). \tag{1.11}$$

Более сложным примером является:

$$\mu_{A\cap B}(S)=\log_S(1+(S^{\mu_A}-1)\cdot(S^{\mu_B}-1)/(S-1)),\ 0<S<\infty. \tag{1.12}$$

При $S\to 0$ выражение (1.12) совпадает с (1.9), при $S\to 1$ совпадает с (1.10), при $S\to\infty$ совпадает с (1.11).

4. Объединение $A\cup B$ – наименьшее нечеткое подмножество, включающее как *A* так и *B*.

Обычно функции принадлежности операции объединения определяют в классе *T*-конорм (называемых также *S*-нормами).

Определение *T*-конорм основано на соотношении:

$$S(\mu_A,\mu_B)=1-T(1-\mu_A,1-\mu_B). \tag{1.13}$$

*T*-конормы удовлетворяют следующим свойствам:

1) $T(1,1)=1$; $T(\mu_A,0)=\mu_A$; $T(0,\mu_A)=\mu_A$ – ограниченность;

2) свойства аналогичные свойствам 2- 4 для Т-норм.

Простейшими примерами операции объединения являются:

$$\mu_{A\cup B}=\max(\mu_A,\mu_B); \tag{1.14}$$

$$\mu_{A\cup B}=\mu_A+\mu_B-\mu_A\cdot\mu_B; \tag{1.15}$$

$$\mu_{A\cup B}=\min(1,\mu_A+\mu_B). \tag{1.16}$$

Более сложным примером является:

$$\mu_{A\cup B}(S)=1-\log_S(1+(S^{1-\mu_A}-1)\cdot(S^{1-\mu_B}-1)/(S-1)),\ 0<S<\infty. \tag{1.17}$$

Выражение (1.17) – аналог выражения (1.13) для Т-норм.

При $S \to 0$ выражение (1.17) совпадает с (1.9), при $S \to 1$ совпадает с (1.15) и при $S \to \infty$ совпадает с (1.16).

Иногда функции принадлежности операций пересечения и объединения определяют операторами более сложными, чем Т-нормы и Т-конормы.

Применение различных видов операторов дополнения, объединения и пересечения, позволяет учесть разнообразные смысловые оттенки соответствующих им связок «НЕ», «И», и «ИЛИ».

Отметим, что применение операций min и max в качестве пересечения и объединения позволяет для нечетких множеств сохранить большинство свойств, известных из теории обычных множеств. В частности, в этом случае для нечетких множеств выполняются следующие свойства.

Пусть $A$, $B$, $C$ – нечеткие множества, тогда:

1. $\left.\begin{array}{l} A \cap B = B \cap A \\ A \cup B = B \cup A \end{array}\right\}$ – коммутативность;

2. $\left.\begin{array}{l} (A \cap B) \cap C = A \cap (B \cap C) \\ (A \cup B) \cup C = A \cup (B \cup C) \end{array}\right\}$ – ассоциативность;

3. $\left.\begin{array}{l} A \cap A = A \\ A \cup A = A \end{array}\right\}$ – идемпотентность;

4. $\left.\begin{array}{l} A \cap (B \cup C) = (A \cap B) \cup (A \cap C) \\ A \cup (B \cap C) = (A \cup B) \cap (A \cup C) \end{array}\right\}$ – дистрибутивность;

5. $A \cup \varnothing = A$, где $\varnothing$ – пустое множество, т.е. $\mu_{\varnothing}(x) = 0$ при $\forall x \in E$;

6. $A \cap \varnothing = \varnothing$;

7. $A \cap E = A$, где $E$ – универсальное множество, на котором определено $A$;

8. $A \cup E = E$;

9. $\left.\begin{array}{l} \overline{\mathrm{A} \cap \mathrm{B}} = \overline{\mathrm{A}} \cup \overline{\mathrm{B}} \\ \overline{\mathrm{A} \cup \mathrm{B}} = \overline{\mathrm{A}} \cap \overline{\mathrm{B}} \end{array}\right\}$ – теоремы де Моргана.

В отличие от четких множеств, для нечетких множеств в общем случае:

$A \cap \overline{A} \neq \varnothing,\ A \cup \overline{A} \neq E$.

Рассмотрим понятия нечеткой и лингвистической переменных, используемые при описании объектов и явлений с помощью теории нечетких множеств.

Нечеткой переменной называют набор $<\alpha, X, A>$, где $\alpha$ – наименование переменной, $X$ – область определения $\alpha$ (универсальное множество), $A$ – нечеткое множество на $X$, описывающие ограничения на значения нечеткой переменной $\alpha$ (т.е. $\mu_A(x)$).

Если областью определения нечеткой переменной является множество действительных чисел $X = R$, то такая нечеткая переменная называется действительным нечетким числом.

Лингвистической переменной называют набор $<\beta, T, X, G, M>$, где $\beta$ – наименование лингвистической переменной, $T$ – множество значений лингвистической переменной (терм-множество), $X$ – универсальное множество (область определения $\beta$), $G$ – синтаксическая процедура, позволяющая оперировать элементами терм-множества $T$, $M$ – семантическая процедура, позволяющая превратить каждое новое значение лингвистической переменной, образуемое процедурой $G$, в нечеткую переменную.

*Пример*. Лингвистическая переменная «сигнал» определена на множестве действительных чисел $X=R^1$ может принимать значения нечетких чисел «большой отрицательный», «приблизительно нулевой», «большой положительный» с функциями принадлежности, приведенными на рис. 1.6.

Перейдем к рассмотрению нечеткого логического вывода.

Механизм нечеткого логического вывода в своей основе имеет базу знаний, формируемую специалистами предметной области в виде совокупности нечетких продукционных правил: если $\vec{x}$ есть $A$, то $z$ есть $C$, где $A$ и $C$ – это нечеткие переменные, определяемые соответствующими функциями принадлежности. Отметим, что $A$ идентифицируется $n$-мерной функцией принадлежности ($n$ – размерность вектора $\vec{x}$).

Часть правила «$\vec{x}$ есть $A$» называется условием или предпосылкой, а «то $z$ есть $C$» – следствием или заключением.

Отметим, что предпосылки нечетких правил обычно записываются отдельно для каждой компоненты вектора $\vec{x}$, с использованием одномерных нечетких переменных. Например, в форме Мамдани (Mamdani) нечеткие продукционные правила имеют вид: если $x_1$ есть $A_1$ и если $x_2$ есть $A_2$ и … и если $x_n$ есть $A_n$ , то $z$ есть $C$.

При этом, очевидно, $A = A_1 \cap A_2 \cap \ ... \ \cap A_n$.

Важным понятием теории нечетких множеств является нечеткое отношение. Нечетким бинарным отношением $P : X_1 \to X_2$ называется подмножество декартового произведения двух множеств $X_1$ и $X_2$

$$P = \{(x_1, x_2) \mid \mu_p(x_1, x_2)\}, \forall x_1 \in X_1, \forall x_2 \in X_2$$

где $\mu_p(x_1, x_2)$ - функция принадлежности пары элементов $(x_1, x_2)$ к $P$.

Пусть множества $X$ и $Y$ состоят из элементов $X = \{1,2\}$, $Y = \{1.1,2.1,5\}$. Тогда, нечеткое бинарное отношение можно задать матрицей, элементами которой будут значения $\mu_p(x_i, y_i)$:

$$P = \begin{bmatrix} 0.9 & 0.1 & 0 & x_1 = 1 \\ 0.1 & 0.9 & 0 & x_2 = 2 \\ y_1 = 1.1 & y_2 = 2.1 & y_3 = 5 & \end{bmatrix}$$

В частности, данная матрица может представлять нечеткое отношение «$X$ примерно равно $Y$».

Часто нечеткие отношения используют для представления правил типа «если $A$, то $B$, где $A$ и $B$ нечеткие подмножества ($A \subset X_1$, $B \subset X_2$). Такое правило обозначается как $A \Rightarrow B$. Один из способов задания нечеткого отношения, соответствующего правилу $A \Rightarrow B$, состоит в использовании формулы *декартового произведения* множеств $A$ и $B$ ($A \times B$):

$$P = A \times B = \{(x_1, x_2) \mid \mu_A(x_1) \wedge \mu_B(x_2)\}, \forall x_1 \in X_1, \forall x_2 \in X_2 .$$

Здесь $\mu_A(x_1) \wedge \mu_B(x_2) = \min\{\mu_A(x_1) \wedge \mu_B(x_2)\} = \mu_p(x_1, x_2)$, причем $(x_1, x_2) \in X_1 \times X_2$.

В некоторых случаях полезной может оказаться операция «компенсирующее И», объединяющая в себе свойства конъюнкции и дизъюнкции. Функция принадлежности для этой операции задается следующей формулой:

$$\mu_{A\cap B}(x) = (\mu_A(x)\cdot\mu_B(x))^{1-\gamma}\cdot(1-(1-\mu_A(x))\cdot(1-\mu_B(x)))^{\gamma},$$

где $\gamma$ – параметр, заданный на интервале [0,1]. При малых значениях $\gamma$ «компенсирующее И» соответствует конъюнкции, при значениях близких к единице – дизъюнкции.

В настоящее время нечеткие технологии становятся все более актуальными среди представителей самых различных профессий. Существует несколько причин, на основании которых отдают предпочтение применению систем именно с нечеткой логикой:

эта логика концептуально легче для понимания;

нечеткая логика – гибкая система и устойчива к неточным входным данным;

она может моделировать нелинейные функции произвольной сложности;

в данной логике учитывается опыт специалистов-экспертов;

нечеткая логика основана на естественном языке человеческого общения.

Системы, основанные на нечеткой логике, разработаны и успешно внедрены в таких областях, как управление технологическими процессами, управление транспортом, управление бытовой техникой, медицинская и техническая диагностика, финансовый менеджмент, финансовый анализ, биржевое прогнозирование, распознавание образов, исследование рисковых и критических операций, прогнозирование землетрясений, составление автобусных расписаний, климатический контроль в зданиях.

Многие современные задачи управления просто не могут быть решены классическими методами из-за очень большой сложности математических моделей, их описывающих.

Коротко перечислим отличительные особенности fuzzy-систем по сравнению с прочими:

возможность оперировать входными данными, заданными нечетко: например, непрерывно изменяющиеся во времени значения (динамические задачи), значения, которые невозможно задать однозначно (результаты статистических опросов, рекламные компании и т.д.);

возможность нечеткой формализации критериев оценки и сравнения: оперирование критериями «большинство», «возможно», «предпочтительно» и т.д.;

возможность проведения качественных оценок как входных данных, так и выводимых результатов: оперируя не только собственно значениями данных, но их степенью достоверности (не путать с вероятностью!) и ее распределением;

возможность проведения быстрого моделирования сложных динамических систем и их сравнительный анализ с заданной степенью точности: оперируя принципами поведения системы, описанными fuzzy-методами, во-первых, не тратится много времени на выяснение точных значений переменных и составление уравнений, которые их описывают, во-вторых, возможна оценка разных вариантов выходных значений.

Использование аппарата нечеткой логики рекомендуется:

для очень сложных процессов, когда не существует простой математической модели;

для нелинейных процессов высоких порядков;

если должна производиться обработка (лингвистически сформулированных) экспертных знаний.

Использование аппарата нечеткой логики не рекомендуется, если:

приемлемый результат может быть получен с помощью общей теории управления;

уже существует формализованная и адекватная математическая модель;

проблема не разрешима.

# 2. МЕТОДЫ ПОИСКА РЕШЕНИЙ

Методы решения задач, используемые в интеллектуальных информационных системах, зависят от особенностей проблемной области и от требований, предъявляемых к их решению.

Особенности проблемной области с точки зрения методов решения можно характеризовать следующими параметрами:

размер пространства, в котором ищется решение;

степень изменяемости объекта во времени и пространстве;

полнота модели, описывающей область;

определенность данных о задаче.

Требования пользователя к решению можно характеризовать количеством решений и способом их получения.

Существующие методы решения задач можно классифицировать следующим образом:

методы поиска в одном пространстве – это методы, предназначенные для использования в малых статических областях, использующие точные и полные данные.

методы поиска в иерархических пространствах – это методы, предназначенные для работы в больших статических областях;

методы поиска при неточных и неполных данных;

методы поиска в динамических областях;

методы поиска, использующие несколько моделей.

## 2.1. Представление задачи в пространстве состояний

Задача в пространстве состояний представляется на графе. Вершины графа соответствуют *состояниям* решения частных задач, а дуги – этапам

решения задачи. Одно или несколько *начальных состояний*, соответствующих исходной информации поставленной задачи, образуют корень дерева. Граф также включает одно или несколько *целевых* условий, которые соответствуют решениям исходной задачи. *Поиск в пространстве состояний* характеризует решение задачи как процесс нахождения *пути* решения (цепочки, ведущей к решению задачи) от исходного состояния к целевому.

Определим представление задачи в пространстве состояний формально.

*Пространство состояний* представляется четверкой **(S, F, $S_0$, $S_k$)** со следующими обозначениями.

**S** – множество вершин графа или состояний в процессе решения задачи.

**F** – множество дуг между вершинами или состояний в процессе решения задачи.

**$S_0$** – непустое множество начальных состояний задачи.

**$S_k$** – непустое подмножество **S**, состоящее из целевых состояний. Эти состояния описываются одним из следующих способов.

1. Измеряемыми свойствами состояний, встречающихся в процессе поиска.

2. Свойствами путей, возникающих в процессе поиска, например, стоимостью перемещения по дугам пути.

Решить задачу – значит определить такую последовательность операторов, которая преобразует начальные состояния в конечные. Процесс решения можно представить в виде графа $G = (X, Y)$, где $X = \{x_0, x_1, \ldots\}$ – множество вершин графа, каждая из которых отождествляется с одним из состояний, $Y$ – множество, содержащее пары вершин $(x_i, x_j)$, $x_i, x_j \in X$. Наличие пары $(x_i, x_j)$ свидетельствует о существовании некоторого оператора $f \in F$, преобразующего состояние, соответствующее вершине $x_i$, в состояние $x_j$.

Итак, граф $G$ задает пространство состояний. Построение пространства осуществляется с помощью следующего процесса. Берется некая вершина из $x_0 \in X$, к ней применяются все возможные операторы, порождающие все

дочерние вершины. Порождение всех дочерних вершин для некоторой вершины $x_i$ называется *процессом раскрытия вершин*. Целевая вершина не раскрывается. Процесс построения пространства состояния заканчивается, когда все нераскрытые вершины являются целевыми или терминальными (т.е. вершинами, к которым нельзя применить никаких операторов).

Для иллюстрации поиска в пространстве состояний рассмотрим следующую задачу.

*Пример.* Пусть перед интеллектуальным роботом поставлена задача по переупорядочиванию кубиков, поставленных друг на друга.

Условия задачи.

1. На каждом шаге можно переставлять только один кубик.

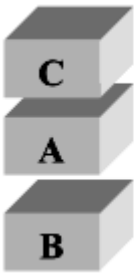
2. Кубик можно взять только тогда, когда его верхняя грань свободна.

3. Кубик можно поставить либо на стол, либо на другой кубик.

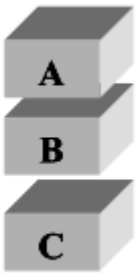
Задачу можно представить как задачу выбора среди множества возможных альтернатив.

В исходной ситуации альтернатива одна – поставить кубик С на стол.

После этого имеется три альтернативы: 1 – поставить кубик А на стол; 2 – поставить кубик А на кубик С; 3 – поставить кубик С на кубик А.




*Исходное состояние*




*Целевое (конечное) состояние*

На рис. 2.1 показано пространство состояний. Путь, ведущий к решению задачи, выделен. Состояниями в пространстве являются все возможные конфигурации положений кубиков А, В и С, которые могут возникнуть в процессе их перестановки. Дуги определяются допустимыми перестановками кубиков, целевое состояние определяется условием задачи.

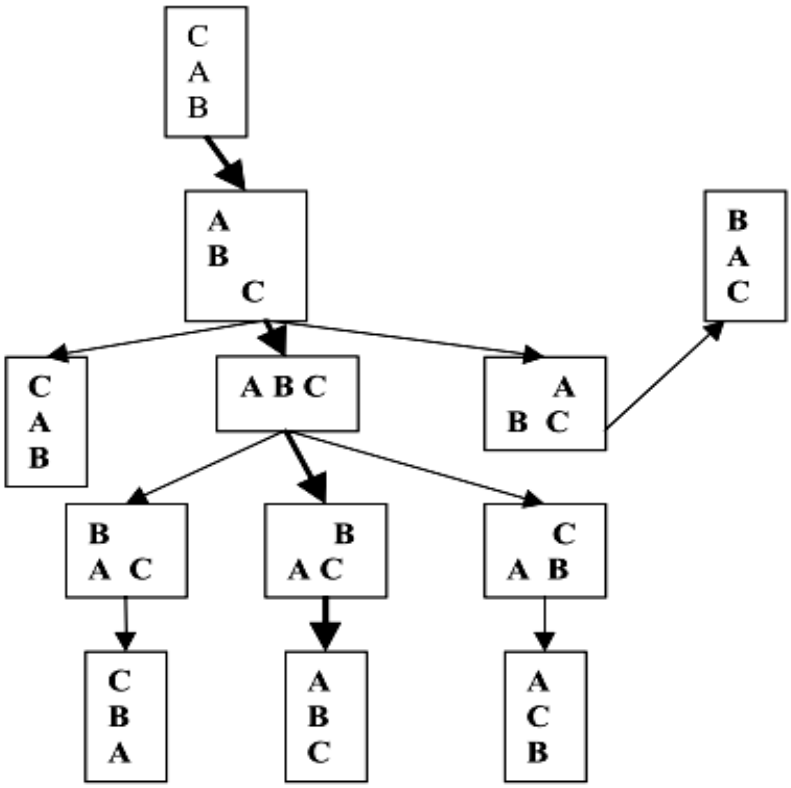



*Рис. 2.1. Пространство состояний решения задачи*

## 2.2. Стратегии поиска в пространстве состояний

Поиск в пространстве состояний можно вести в двух направлениях: от исходных данных задачи к цели и в обратном направлении от цели к исходным данным.

При *поиске на основе данных* (поиск, управляемый данными), который еще называют прямой цепочкой, исследователь начинает процесс решения задачи, анализируя ее условие, а затем применяет допустимые ходы или правила изменения состояния. В процессе поиска правила применяются к известным фактам для получения новых фактов, которые в свою очередь, используются для генерации новых фактов. Этот процесс продолжается до тех пор, пока не будет достигнута цель.

Возможен и альтернативный подход. Рассмотрим цель, которую мы хотим достичь. Проанализируем правила или допустимые ходы, ведущие к цели, и определим условия их применения. Эти условия становятся новыми целя-

ми, или подцелями, поиска. Поиск продолжается в обратном направлении от достигнутых подцелей до тех пор, пока мы не достигнем исходных данных задачи. Таким образом, определяется путь от данных к цели, который на самом еле строится в обратном направлении. Этот подход называется *поиском от цели*, или *обратной цепочкой*.

Таким образом, поиск на основе данных начинается с условий задачи и выполняется путем применения правил для получения новых фактов, ведущих к цели. Поиск от цели начинается с обращения к цели и продолжается путем определения правил, которые могут привести к цели, и построения цепочки подцелей, ведущей к исходным данным задачи.

Следует заметить, что в обоих случаях исследователь работает с одним и тем же графом пространства состояний, однако порядок и число состояний в процессе поиска могут различаться. Какую стратегию выбрать, зависит от самой задачи. При этом следует учитывать сложность правил, «форму» пространства состояний и доступность данных задачи.

Как пример зависимости сложности поиска от выбора стратегии рассмотрим задачу, в которой нужно подтвердить или опровергнуть утверждение «Я – потомок лейтенанта Шмидта». Положительным решением является путь по генеалогическому дереву от «Я» до «Шмидт». Поиск на этом графе можно вести в двух направлениях: начиная от вершины «Я» строить цепочку предков к вершине «Шмидт», или начиная с вершины «Шмидт», анализировать цепочку его потомков.

Сравним сложность поиска в обоих направлениях. Лейтенант Шмидт родился примерно 250 лет назад. Если считать, что новое поколение рождается примерно каждые 25 лет, то длина искомого пути составляет примерно 10. Поскольку каждый потомок имеет двух родителей, то путь от «Я» требует анализа $2^{10}$ предков. С другой стороны, поиск от вершины «Шмидт» требует анализа большего числа состояний, поскольку родители обычно имеют больше двух детей (особенно это касается XVIII и XIX столетий). Примем,

что каждая семья имеет в среднем троих детей. Тогда в процессе поиска нужно проанализировать $3^{10}$ вершин генеалогического дерева.

Процесс поиска от цели рекомендован в следующих случаях.

1. Цель поиска (или гипотеза) явно присутствует в постановке задачи или ее можно легко сформулировать. Многие диагностические системы рассматривают возможные диагнозы, систематически подтверждая или отвергая некоторые из них способом поиска от цели.

2. Имеется большое число правил, которые на основе полученных фактов позволяют продуцировать возрастающее число заключений или целей. Своевременный отбор целей позволяет отсеять множество возможных ветвей, что делает процесс поиска в пространстве состояний более эффективным.

3. Исходные данные не приводятся в задаче, но подразумевается, что они известны решателю. В этом случае поиск от цели может служить руководством для правильной постановки задачи. Например, в программе медицинской диагностики имеются всевозможные диагностические тесты. Доктор выбирает из них только те, которые позволяют подтвердить или опровергнуть конкретную гипотезу о состоянии пациента.

Таким образом, при поиске от цели подходящие правила применяются для исключения неперспективных ветвей поиска.

Поиск на основе данных применим в следующих случаях.

1. Все или большинство исходных данных заданы в постановке задачи. Задача интерпретации состоит в выборе этих данных и их представлении в виде, подходящем для использования в интерпретирующих системах более высокого уровня. На стратегии поиска от данных основаны системы анализа данных.

2. Существует большое число потенциальных целей, но всего лишь несколько способов применения фактов и представления информации о конкретном примере задачи.

3. Сформировать цели и гипотезы трудно

При поиске на основе данных знания и ограничения, заложенные в исходной постановке задачи, используются для нахождения пути к решению.

## 2.3. Поиск в глубину и в ширину

Определив направление поиска (от данных или от цели), алгоритм поиска должен определить порядок исследования состояний дерева или графа. Рассмотрим два возможных варианта последовательности обхода узлов графа: *поиск в глубину* и *поиск в ширину*.

Базовый алгоритм *поиска в ширину* состоит из следующей последовательности шагов:

**Шаг 1.** Поместить начальную вершину в список нераскрытых вершин **`Open`**.

**Шаг 2.** Если список **`Open`** пуст, то завершить поиск и выдать сообщение о неудаче, в противном случае перейти к следующему шагу.

**Шаг 3.** Выбрать *первую* вершину из списка **`Open`** (назовем ее **`Current`**) и перенести ее в список раскрытых вершин **`Closed`**.

**Шаг 4.** Раскрыть вершину **`Current`**, образовав все ее дочерние вершины. Если дочерних вершин нет, то перейти к шагу 2, иначе поместить все дочерние вершины (в любом порядке) в *конец* списка **`Open`** и построить *указатели*, ведущие от этих вершин к родительской вершине **`Current`**.

**Шаг 5.** Проверить, нет ли среди дочерних вершин целевых. Если есть хотя бы одна *целевая* вершина, то завершить поиск и выдать решение задачи, получающееся просмотром указателей назад от найденной целевой вершины к начальной. В противном случае перейти к шагу 2.

**Конец алгоритма.**

Основу алгоритма составляет цикл последовательного раскрытия (шаги 2-5) концевых вершин (листьев) дерева перебора, хранящихся в списке **`Open`**. Описанный алгоритм поиска в ширину является полным. Можно также показать, что при переборе в ширину непременно будет найден самый короткий путь к целевой вершине – при условии, что этот путь вообще существует. Если же решающего пути нет, то в случае конечного пространства состояний ал-

горитм завершит работу (с сообщением о неудаче поиска), в случае же бесконечного пространства алгоритм не остановится.

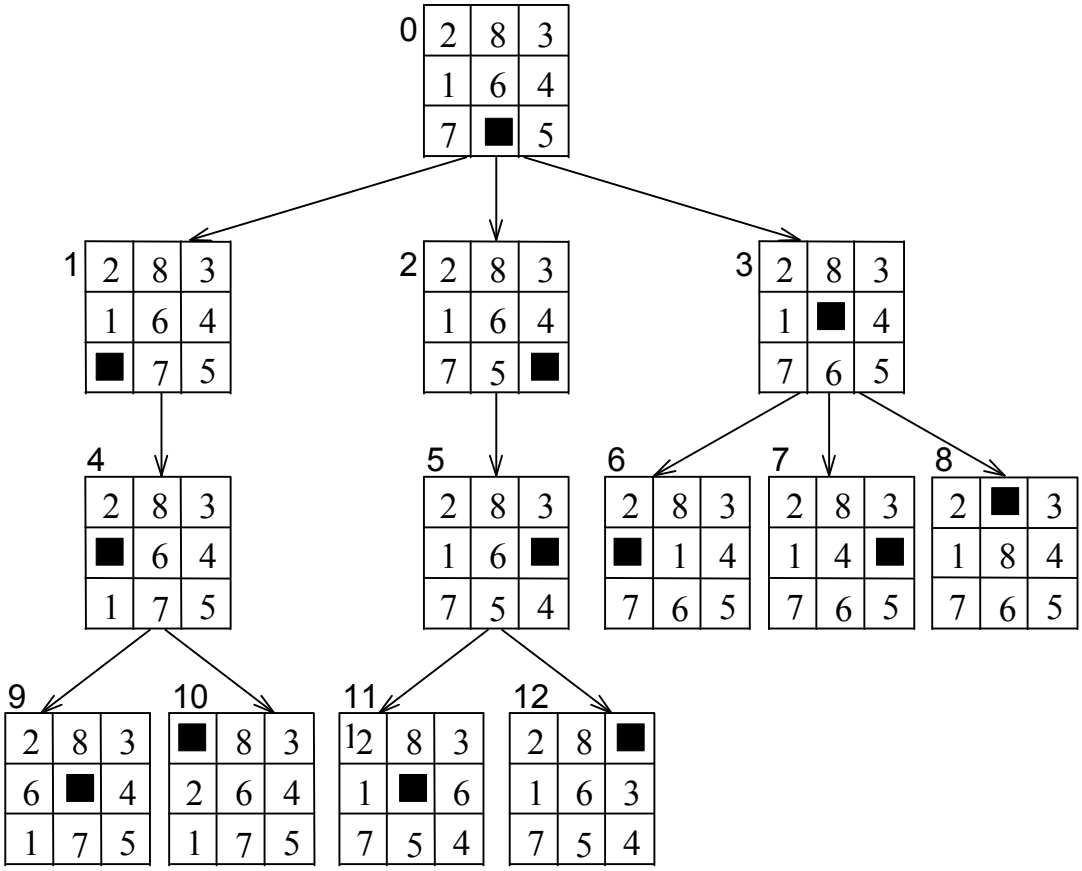



*Рис. 2.2. Поиск в ширину для игры в восемь*

На рис. 2.2 показана часть дерева, построенного в результате применения алгоритма поиска в ширину к некоторой начальной конфигурации игры в восемь, причем выполнение алгоритма прервано после построения первых 12 вершин (при этом раскрыто 6 вершин). В вершинах дерева помещены соответствующие описания состояний. Эти вершины перенумерованы в том порядке, в котором они были построены в ходе поиска. На следующем шаге цикла алгоритма будет раскрываться одна из вершин с номерами 6, 7 или 8, поскольку они расположены в начале списка нераскрытых вершин.

Для формулировки алгоритма поиска в глубину необходимо определить понятие *глубины вершины* в дереве поиска. Это можно сделать следующим образом:

глубина *корня* дерева равна нулю;

глубина каждой *некорневой* вершины на единицу больше глубины ее родительской вершины.

В алгоритме перебора в глубину раскрытию в первую очередь подлежит вершина, имеющая наибольшую глубину. Такой принцип может привести к не завершающемуся процессу – это происходит, если пространство состояний бесконечно, и поиск в глубину пошел по бесконечной ветви дерева, не содержащей целевой вершины. Поэтому необходимо то или иное ограничение поиска, самый распространенный способ – ограничить глубину просмотра дерева. Это означает, что в ходе перебора можно строить только такие вершины, глубина которых не превышает некоторую заданную *граничную глубину*. Тем самым, раскрытию в первую очередь подлежит вершина наибольшей глубины, но только если она расположена выше фиксированной границы. Соответствующий алгоритм поиска называется *ограниченным поиском в глубину*.

Основные шаги базового алгоритма *ограниченного поиска в глубину* (с граничной глубиной **D**) таковы:

**Шаг 1.** Поместить начальную вершину в список нераскрытых вершин **Open**.

**Шаг 2.** Если список **Open** пуст, то завершить поиск и выдать сообщение о неудаче, в противном случае перейти к следующему шагу.

**Шаг 3.** Выбрать *первую* вершину из списка **Open** (назовем ее **Current**) и перенести ее в список раскрытых вершин **Closed**.

**Шаг 4.** Если глубина вершины **Current** равна *граничной глубине* **D**, то перейти к шагу 2, в ином случае перейти к следующему шагу.

**Шаг 5.** Раскрыть вершину **Current**, построив все ее дочерние вершины. Если дочерних вершин нет, то перейти к шагу 2, иначе поместить все дочерние вершины (в произвольном порядке) в *начало* списка **Open** и построить *указатели*, ведущие от этих вершин к родительской вершине **Current**.

**Шаг 6.** Если среди дочерних есть хотя бы одна целевая вершина, то завершить поиск и выдать решение задачи, получающееся просмотром назад указа-

телей от найденной целевой вершины к начальной. В противном случае перейти к шагу 2.

**Конец алгоритма.**

Приведенное описание очень похоже на описание алгоритма поиска в ширину, разница заключается только в ограничении глубины (шаг 4) и в участке списка **Open**, куда помещаются построенные дочерние вершины (шаг 5).

Поскольку глубина поиска ограничена, то будучи примененным к деревьям-пространствам состояний, описанный базовый алгоритм поиска в глубину всегда заканчивает работу. В отличие от алгоритма поиска в ширину, он осуществляет неполный поиск, поскольку целевая вершина может располагаться ниже граничной глубины (в таком случае она не будет обнаружена).

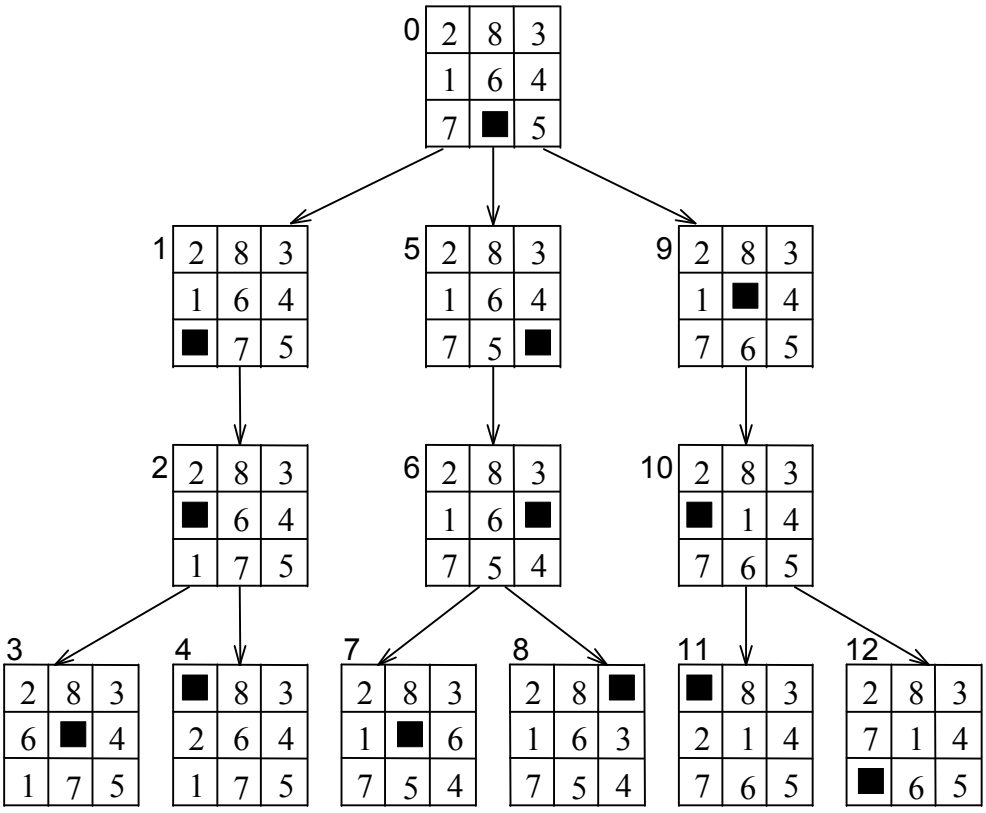



*Рис. 2.3. Поиск в глубину для игры в восемь*

На рис. 2.3 показана часть дерева перебора, построенного алгоритмом поиска в глубину при граничной глубине равной 4. В качестве начального состояния взята та же самая конфигурация игры в восемь, что и в примере на

рис.13. В ходе поиска также построено 12 вершин (раскрыто 7), вершины перенумерованы в том порядке, в котором они были построены. Как нетрудно убедиться, сравнивая два указанных рисунка, алгоритмами поиска в ширину и в глубину построены разные деревья поиска. Видно, что в алгоритме поиска в глубину сначала идет поиск вдоль одного пути, пока не будет достигнута установленная граничная глубина, затем рассматриваются альтернативные пути той же или меньшей глубины, которые отличаются от первого пути лишь последней (концевой) вершиной, после чего рассматриваются пути, отличающиеся последними двумя вершинами, и т.д.

## 2.4. Слепой перебор и бектрекинг

Если продолжить выполнение алгоритмов перебора в ширину и в глубину для рассмотренного на рис.13 и 14 начального состояния игры в восемь, то она будет найдена на глубине 5, при этом алгоритмом поиска в ширину будет раскрыто 26 и построено 46 вершин, а алгоритмом поиска в глубину – соответственно 18 и 35 вершин.

В целом, алгоритмы поиска в ширину и в глубину сравнимы по эффективности (в частности, по количеству построенных вершин). В то же время алгоритм поиска в глубину может оказаться предпочтительнее – в тех случаях, когда он начат с ветви дерева, содержащей целевое состояние, решение задачи будет обнаружено раньше, чем при поиске в ширину.

Подчеркнем, что, также как и при переборе в ширину, при переборе в глубину формируется именно дерево, а не граф поиска, даже если пространство состояний представлялось графом с циклами. В последнем случае, однако, дерево перебора может содержать дубликаты состояний и возможно зацикливание алгоритма. Если, к примеру, в графе две вершины являются друг для друга дочерними, то они будут многократно дублироваться в списке **Open**, что приводит к зацикливанию.

Чтобы избежать такого дублирования вершин в случае перебора на графах общего вида, необходимо внести некоторые очевидные изменения в описанные базовые алгоритмы поиска в ширину и в глубину.

В алгоритме перебора в ширину следует дополнительно проверять, не находится ли каждая вновь построенная вершина (точнее, соответствующее описание состояния) в списках **Open** и **Closed** по той причине, что она уже строилась раньше в результате раскрытия какой-то другой вершины. Если это так, то такую вершину не надо снова помещать в список **Open** (таким образом, разрывается цикл графа-пространства и обрывается соответствующая ветвь дерева перебора). В алгоритме же ограниченного поиска в глубину, кроме указанной проверки, может оказаться необходимым пересчет глубины порожденной дочерней вершины, уже имеющейся либо в списке **Open**, либо в списке **Closed**.

Усовершенствованный таким образом алгоритм поиска в ширину всегда завершит работу в случае существования решения, а усовершенствованный алгоритм поиска в глубину закончится в любом случае, независимо от существования решения.

Алгоритм поиска в глубину демонстрирует также способ решения поисковых задач, называемый *бектрекингом* (*backtracking*). Этот способ предлагает определенную организацию перебора всех возможных вариантов решения задачи, число которых может быть велико.

Суть бектрекинга состоит в том, чтобы в каждой точке процесса решения задачи, где существует несколько априори равноправных альтернативных вариантов (путей) дальнейшего продолжения, выбрать один из них и следовать ему, предварительно запомнив другие альтернативы – для того, чтобы в случае неуспешности выбранного варианта-пути вернуться в точку выбора и выбрать для продолжения решения другой возможный вариант-путь. В общем случае в процессе решения возможно возникновение многих подобных точек выбора, называемых обычно *точками бектрекинга* или *развилками*; к каждой из таких точек может потребоваться *возврат* для выбора других вариантов решения.

В базовом алгоритме поиска в глубину по существу проводится бектрекинг. Действительно, запоминание всех альтернатив продолжения поиска (нераскрытых вершин) осуществляется в списке **Open**, на шаге 3 производится выбор варианта-альтернативы, а возврат к этому шагу для выбора следующей альтернативы осуществляется на шагах 4 и 5.

В целом алгоритмы слепого перебора являются неэффективными методами поиска решения, и в случае нетривиальных задач их невозможно использовать из-за большого числа порождаемых вершин. Действительно, если $L$ – длина решающего пути, а $B$ – количество ветвей (дочерних вершин) у каждой вершины, то в общем случае для нахождения решения надо исследовать $B^L$ путей, ведущих из начальной вершины. Эта величина растет экспоненциально с ростом длины решающего пути, что приводит к ситуации, называемой комбинаторным взрывом.

Один из способов повышения эффективности поиска связан с использованием информации, отражающей специфику решаемой задачи и позволяющей более целенаправленно двигаться к цели. Такая информация обычно называется *эвристической*, а соответствующие алгоритмы и методы поиска – эвристическими.

## 2.5. Эвристический поиск

Идея, лежащая в основе эвристического поиска, состоит в том, чтобы с помощью эвристической информации оценивать перспективность нераскрытых вершин пространства состояний (с точки зрения достижения цели) и выбирать для продолжения поиска наиболее перспективную вершину.

Самый распространенный способ использования эвристической информации – введение так называемой *эвристической оценочной функции*. Эта функция определяется на множестве вершин пространства состояний и принимает числовые значения. Значение эвристической оценочной функции $Est(V)$ может интерпретироваться как перспективность раскрытия вершины (иногда – как вероятность ее расположения на решающем пути). Обычно считают, что меньшее значение

*Est*(*V*) соответствует более перспективной вершине, и вершины раскрываются в порядке увеличения (точнее, неубывания) значения оценочной функции.

### *2.5.1. Алгоритм эвристического поиска*

Последовательность шагов формулируемого ниже базового *алгоритма эвристического (упорядоченного) перебора* похожа на последовательность шагов алгоритмов слепого перебора, отличие заключается в использовании эвристической оценочной функции. После порождения нового состояния производится его оценивание (т.е. вычисление значения этой функции). Списки открытых и закрытых вершин содержат как вершины, так и их оценки, которые и используются для упорядочения поиска.

В цикле каждый раз для раскрытия выбирается наиболее перспективная концевая вершина дерева перебора. Как и в случае алгоритмов слепого поиска, множество порождаемых алгоритмом вершин и указателей образует дерево, в листьях которого находятся нераскрытые вершины.

Предполагаем, что исследуемое алгоритмом пространство состояний представляет собой дерево. Укажем основные шаги базового *алгоритма эвристического поиска* (*best_first_search*):

**Шаг 1.** Поместить начальную вершину в список нераскрытых вершин **Open** и вычислить ее оценку.

**Шаг 2.** Если список **Open** пуст, то завершить поиск и выдать сообщение о неудаче, в противном случае перейти к шагу 3.

**Шаг 3.** Выбрать из списка **Open** вершину с *минимальной оценкой* (среди вершин с одинаковой минимальной оценкой выбирается любая) и перенести эту вершину (назовем ее **Current)** в список **Closed**.

**Шаг 4.** Если **Current** – *целевая* вершина, то завершить поиск и выдать решение задачи, получающееся просмотром указателей от нее к начальной вершине, в противном случае перейти к шагу 5.

**Шаг 5.** Раскрыть вершину **Current**, построив все ее дочерние вершины. Если таких вершин нет, то перейти к шагу 2, в ином случае – к шагу 6.

**Шаг 6.** Для каждой дочерней вершины вычислить оценку (значение оценочной функции), поместить все дочерние вершины в список **Open** и построить указатели, ведущие от этих вершин к родительской вершине **Current**. Перейти к шагу 2.

**Конец алгоритма.**

Заметим, что поиск в глубину можно рассматривать как частный случай эвристического поиска с оценочной функцией $Est(V) = d(V)$ , а поиск в ширину – с оценочной функцией $Est(V) = 1/d(V)$ , где $d(V)$ – глубина вершины $V$.

Чтобы модифицировать рассмотренный алгоритм для перебора на произвольных графах-пространствах состояний, необходимо предусмотреть в нем реакцию на случай построения дочерних вершин, которые уже имеются либо в списке раскрытых, либо в списке нераскрытых вершин.

В принципе, эвристическая оценочная функция может зависеть не только от внутренних свойств оцениваемого состояния (т.е. свойств, входящих в описание состояния элементов), но и от характеристик всего пространства состояний, например, от глубины местонахождения оцениваемой вершины в дереве перебора или других свойств пути к этой вершине. Поэтому значение оценочной функции для вновь построенной дочерней вершины, входящей в список **Open** или **Closed**, может понизиться, и надо скорректировать старую оценку вершины, заменив ее на новую, меньшую. Если вновь построенная вершина с меньшей оценкой входит в список **Closed**, необходимо вновь поместить ее в список **Open**, но с меньшей оценкой. Потребуется также изменить направления указателей от всех вершин списков **Open** и **Closed**, оценка которых уменьшилась, направив их к вершине **Current**.

Если оценочная функция учитывает только внутренние характеристики вершин-состояний, то для предотвращения зацикливания требуется более простая модификация алгоритма – надо просто исключить дублирование состояний в списках **Open** и **Closed**, оставляя в них лишь по одному состоянию.

Проиллюстрируем работу алгоритма эвристического поиска опять же на примере игры в восемь. Воспользуемся в качестве оценочной следующей простой функцией:

$Est1(V) = d(V) + k(V)$,

где $d(V)$ – глубина вершины $V$, или число ребер дерева на пути от этой вершины к начальной вершине;

$k(V)$ – число фишек позиции-вершины $V$, стоящих не на «своем» месте (фишка стоит не на «своем» месте, если ее позиция отлична от позиции в целевом состоянии).

На рис. 2.4 показано дерево, построенное алгоритмом эвристического перебора с использованием указанной оценочной функции. Оценка каждой вершины приведена рядом с ней внутри кружка. Отдельно стоящие цифры, как и раньше, показывают порядок, в котором строились вершины. Двойной рамкой обведена найденная целевая вершина, она построена двенадцатой.

Видно, что поскольку каждый раз выбор вершины с минимальной оценкой производится внутри всего построенного к текущему моменту дерева перебора, то раскрываемые друг за другом вершины могут располагаться в отдаленных друг от друга частях дерева. Применяемая оценочная функция такова, что при прочих равных преимущество имеет менее глубокая вершина.

Решение задачи длиною в пять ходов найдено в результате раскрытия 6 и построения 13 вершин – это существенно меньше, чем при использовании слепого перебора (соответствующие числа были: для поиска в ширину – 26 и 46 вершин, для поиска в глубину – 18 и 35 вершин). Таким образом, использование эвристической информации приводит к существенному сокращению перебора.

Эффективность алгоритмов поиска может быть оценена при помощи такого показателя, как *целенаправленность*. Он вычисляется по формуле $P=L/N$, где $L$ – длина найденного пути до цели (она равна глубине целевой вершины);

$N$ – общее число вершин, построенных в ходе перебора.

Легко заметить, что $P=1$, если строятся только вершины решающего пути, а в остальных случаях $P<1$. Вообще, эта величина тем меньше, чем больше строится бесполезных вершин. Таким образом, этот критерий показывает, насколько дерево, построенное при переборе, вытянуто, а не кустисто. Для рас-

смотренных примеров работы алгоритмов для игры в восемь этот показатель равен соответственно: эвристический поиск – 5/13, поиск в ширину – 5/46, поиск в глубину – 5/35,

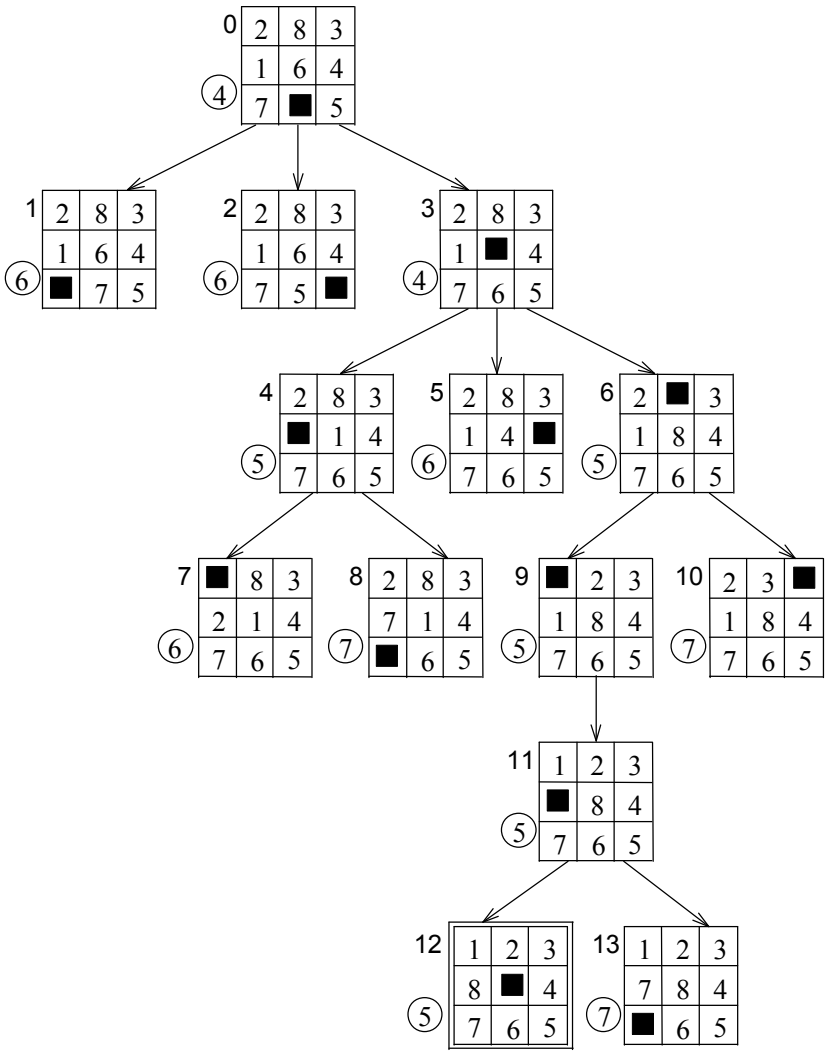



*Рис. 2.4. Эвристический поиск для игры в восемь*

Ясно, что алгоритм эвристического поиска с хорошо подобранной оценочной функцией находит решение задачи быстрее алгоритмов слепого перебора. Однако подбор удачной эвристической функции, существенно сокращающей поиск, – наиболее трудный момент при формализации задачи, часто подходящая оценочная функция выявляется только экспериментально.

В одной и той же задаче можно сравнивать различные оценочные функции по их *эвристической силе*, т.е. по тому, насколько они убыстряют поиск, делают его эффективным. Заметим, что эвристическая сила функции должна

учитывать общий объем вычислительных затрат при поиске, поэтому кроме числа раскрытых и построенных вершин важен и такой показатель, как сложность вычисления самой оценочной функции.

Для игры в восемь можно предложить еще одну эвристическую функцию: $Est2(V) = d(V) + s(V)$.

Первое слагаемое $d(V)$ этой функции имеет тот же смысл, что и для функции $Est1$. Второе слагаемое получается подсчетом для каждой из восьми фишек суммы двух расстояний – по вертикали и горизонтали – между клетками, где находится фишка в оцениваемом и целевом состояниях, а затем вычислением общей суммы $s(V)$ таких расстояний для всех восьми фишек. Тем самым, $s(V)$ выражает «суммарное расстояние» всех фишек от их целевого положения.

Например, для начальной конфигурации на рис. 2.4 расстояние текущего положения фишки с номером 8 от ее положения в целевой конфигурации равно 1 и по вертикали, и по горизонтали, а сумма их равна 2. Общая же сумма таких расстояний для всех фишек равна 5 (фишки 3, 4, 5, 7 стоят уже на «своем» месте, поэтому их вклад в суммарное расстояние равен 0). Интуитивно ясно, и это можно показать на примерах, что новая эвристическая функция имеет большую эвристическую силу, т.е. более эффективно направляет поиск к цели.

### *2.5.2. Алгоритм «подъема на холм»*

Сильным упрощением базового алгоритма эвристического поиска с произвольной оценочной функцией является алгоритм ***«подъема на холм»***. Этот алгоритм при раскрытии каждой вершины производит упорядочение (по значению оценочной функции) порожденных дочерних вершин, и выбирает для последующего раскрытия дочернюю вершину с наименьшей оценкой, а не вершину с наименьшей оценкой среди всех нераскрытых вершин дерева поиска, как в базовом алгоритме эвристического поиска. Очевидно, что такой локальный выбор среди только что построенных дочерних вершин реализовать гораздо проще, чем глобальный выбор вершины во всем дереве перебора.

Идея этого алгоритма аналогична идее известного вне области искусственного интеллекта метода «подъема на гору», применяемого для поиска максимума (или минимума) функции. Согласно этому методу, для того чтобы, в конечном счете, найти максимум функции, на каждом шаге метода производится движение в направлении наибольшей крутизны функции. Для определенного класса функций (имеющих единственный максимум и некоторые другие свойства роста) такое использование локальной информации, т.е. знания направления наиболее крутого подъема в текущей точке, позволяет найти глобальное решение, т.е. максимум функции.

В алгоритме «подъема на холм», применяемом для поиска в пространстве состояний, роль функции метода «подъема на гору» играет эвристическая оценочная функция, взятая с обратным знаком. Поиск продолжается всегда от той дочерней вершины, которая имеет меньшее значение эвристической функции (при этом случай, когда вершин с одинаковой минимальной оценкой несколько, является нежелательным).

Важно, что алгоритм «подъема на холм» дает тот же результат, что и базовый алгоритм эвристического поиска в тех случаях, когда оценочная функция обладает определенными свойствами, в частности, имеет один (глобальный) экстремум. Алгоритм становится несостоятельным, если у эвристической функции имеется несколько локальных экстремумов. Бывают и другие случаи бесперспективности «подъема на холм»: если поверхность-множество значений функции имеет равнинный участок (как горное плато) или же участки узкого и длинного возвышения (в виде горного хребта), и процесс поиска вывел как раз на них. Таким образом, рассматриваемый алгоритм имеет ограниченную применимость, но иногда возникающие проблемы можно разрешить, построив более подходящую эвристическую функцию.

## 2.6. Редукция задач на подзадачи

При поиске решения методом редукции задач на подзадачи решение поставленной задачи сводится к решению совокупности образующих ее подзадач. Этот процесс повторяется для каждой подзадачи до тех пор, пока каждая подзадача не будет иметь очевидное решение. Решение подзадачи считается очевидным, если оно было получено ранее или считается общеизвестным. Поиск решения задачи методом редукции можно представить в виде И/ИЛИ графа. Приведем пример.

Пусть имеются следующие высказывания:

$q \rightarrow p$

$r \rightarrow p$

$v \rightarrow q$

$s \rightarrow r$

$t \rightarrow r$

$s \rightarrow u$

Известны начальные высказывания $s$ и $t$. Отношения между начальными утверждениями и выведенными можно представить в виде графа в пространстве состояний (рис. 2.5).

Дуги на рис. 2.6 соответствуют логическим импликациям (→).

Утверждения, которые считаются истинными ($s$ и $t$), соответствуют исходным данным задачи. Логические следствия из данного набора утверждений, соответствуют достижимым узлам. Путь на графе отражает последовательность вывода.

Приведенные правила содержат альтернативные высказывания для определения $p$ и $r$. Эти высказывания мы можем записать в виде правила $s \rightarrow r \vee s \rightarrow u$.

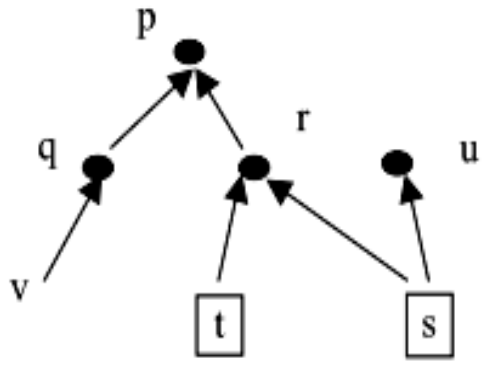



*Рис. 2.5. Граф пространства состояний*

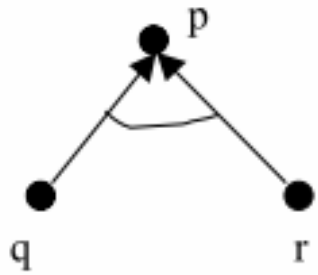



*Рис. 2.6. Граф И/ИЛИ*

Кроме того, можно объединить некоторые высказывания и получить сложные высказывания: $q \wedge r \to p$ и $s \wedge t \to r$.

Для иллюстрации этого подхода рассмотрим один из вариантов известной головоломки – задачи о ханойской башне (рис. 2.7). В ней используются 3 колышка (обозначим их буквами A, B, C) и 3 диска разного диаметра, которые можно нанизывать на колышки через отверстия в центре. В начале все диски расположены на колышке A, причем диски меньшего диаметра лежат на дисках большего диаметра (диски нумерованы в порядке возрастания диаметра). Требуется переместить все диски на колышек C, соблюдая следующие правила. Перемещать можно только самый верхний диск, и нельзя никакой диск класть на диск меньшего размера. На рис.6 показаны начальное и целевое состояния задачи о ханойской башне.

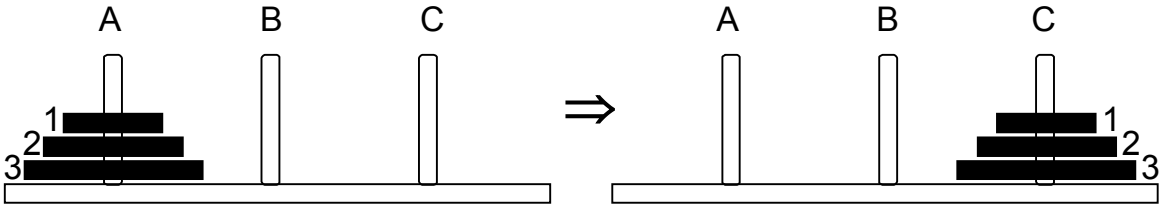



*Рис. 2.7. Задача о ханойской башне*

Эта задача легко может быть формализована в пространстве состояний: состояние задачи задается списком из трех элементов, каждый из которых ука-

зывает местоположение соответствующего диска (первый элемент – первого диска, второй – второго, третий – третьего). Начальное состояние описывается списком (AAA), а целевое – (CCC). При этом предполагается, что если на одном колышке находится более одного диска, то любой больший диск расположен ниже любого меньшего.

Полное пространство состояний задачи о пирамидке представлено графом на рис. 2.8, включающем 27 вершин. Путь в этом графе, выделенный жирными дугами со стрелками, представляет собой решение задачи, которое включает 7 перемещений дисков.

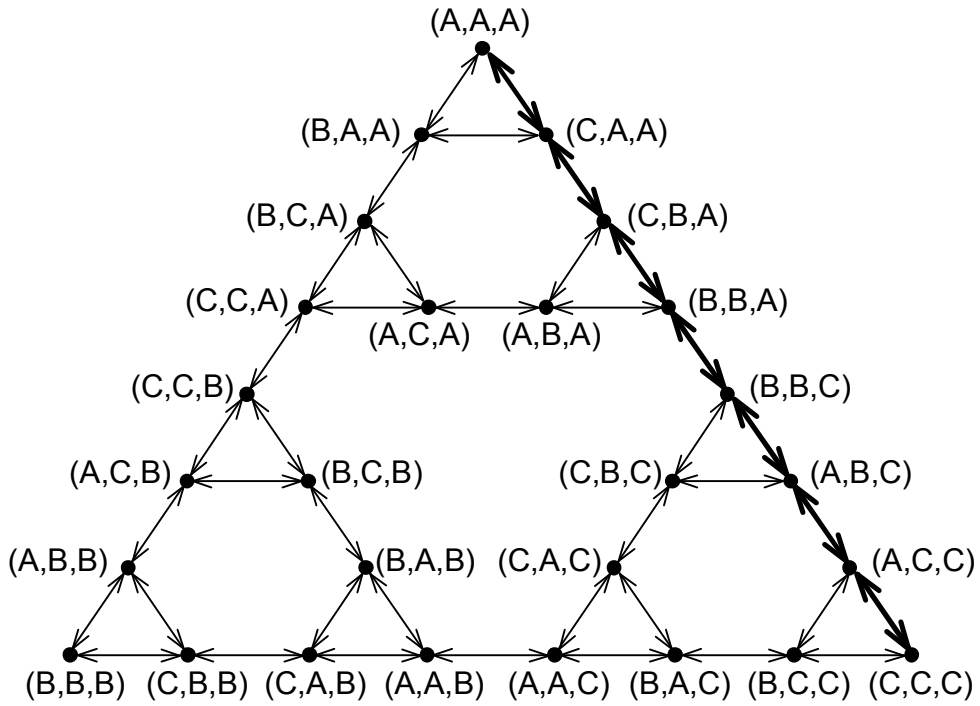



*Рис. 2.8.. Пространство состояний в задаче о ханойской башне*

Это решение можно обнаружить перебором всех возможных перемещений дисков, но метод редукции задач позволяет решить рассматриваемую задачу быстрее. Ключевая идея редукции состоит в том, что для перемещения всей пирамидки необходимо переложить самый нижний диск 3, а это возможно, только если располагающаяся над ним пирамидка из двух меньших дисков 1 и 2 перенесена на колышек B – см. рис. 2.9,а.

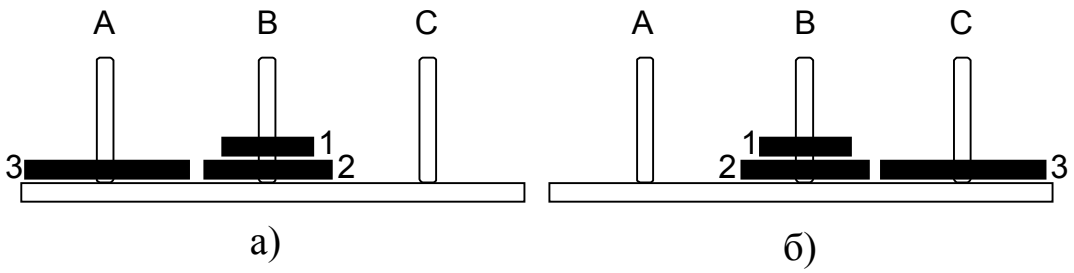



*Рис. 2.9. Задача о ханойской башне: два решающих состояния*

Таким образом, исходную задачу можно свести к трем следующим подзадачам:

переместить диски 1 и 2 с колышка A на колышек B (1, 2 : A→B);

переместить диск 3 с колышка A на колышек C (3 : A→C);

переместить диски 1 и 2 с колышка B на колышек C (1, 2 : B→C).

На рис. 2.9,б показано исходное состояние для третьей задачи.

Каждая из трех указанных задач проще исходной. Действительно, в первой и в третьей задачах требуется переместить всего два диска, вторая же задача может рассматриваться как элементарная, так как ее решение состоит ровно из одного хода – перемещения диска 3 на колышек C. В первой и третьей задачах можно вновь применить метод редукции задач, и свести их к элементарным задачам. Весь процесс редукции можно схематически представить в виде дерева на рис. 2.10.

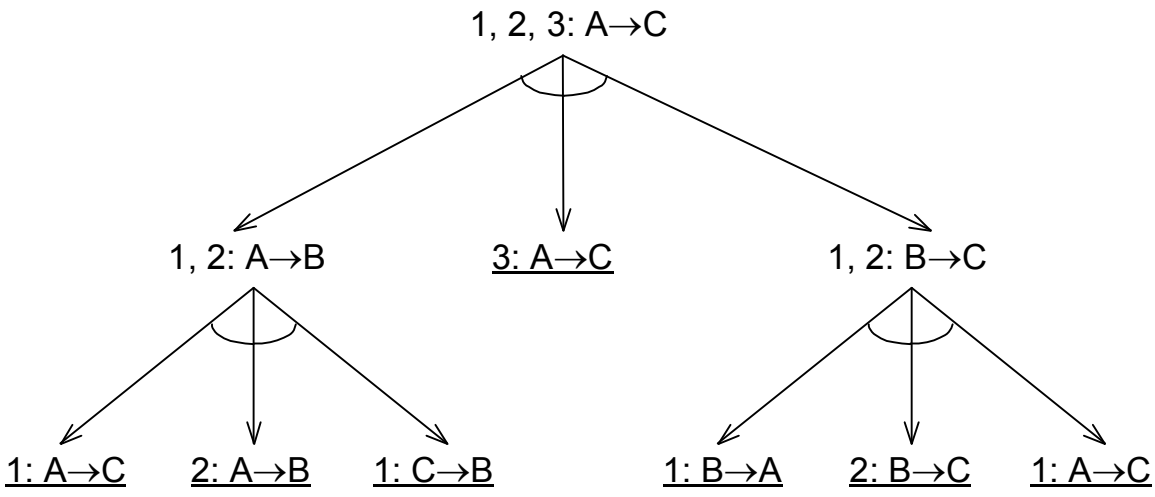



*Рис. 2.10. Редукция задачи о ханойской башне*

Вершины дерева соответствуют решаемым задачам/подзадачам, причем листья дерева соответствуют элементарным задачам перемещения дисков, а дуги связывают редуцируемую задачу с ее подзадачами.

При решении задач методом редукции, как и при решении в пространстве состояний, может возникнуть необходимость перебора. Действительно, на каждом этапе редукции может оказаться несколько применимых операторов (т.е. способов сведения задачи к подзадачам) и, соответственно, несколько альтернативных множеств подзадач. Некоторые способы, возможно, не приведут к решению исходной задачи, поскольку могут обнаружиться неразрешимые подзадачи, другие же способы могут дать окончательное решение. В общем случае для полной редукции исходной задачи необходимо перепробовать несколько операторов. Процесс редукции продолжается, пока исходная задача не будет сведена к набору элементарных задач, решение которых известно.

Аналогично представлению в пространстве состояний, формализация задачи в рамках подхода, основанного на редукции задач, включает определение следующих составляющих:

формы описания задач/подзадач и описание исходной задачи;

множества операторов и их воздействий на описания задач;

множества элементарных задач.

Эти составляющие задают неявно пространство задач, в котором требуется провести поиск решения задачи.

Что касается формы описания задач/подзадач, то часто их удобно описывать в терминах пространства состояний, т.е. задавая начальное состояние и множество операторов, а также целевое состояние или его свойства. В этом случае элементарными задачами могут быть, к примеру, задачи, решаемые за один шаг в пространстве состояний.

Если выбрать такую форму описания в задаче о пирамидке, то элементарная задача перекладывания самого большого диска с колышка A на C записывалась бы как (BBA)⇨(BBC), а исходная задача – как (AAA)⇨(CCC). Причем, по-

скольку множество операторов предполагается одним и тем же для всех задач/подзадач, то в их описаниях оно в явном виде может не фигурировать. При выбранной форме описания задач результирующие подзадачи естественно интерпретируются как задачи нахождения пути между определенными состояниями-вехами в пространстве состояний. К примеру, такими вехами являются состояния (BBA) и (BBC) задачи о пирамидке, изображенные на рис. 2.9. Они обладают тем свойством, что через них пройдет и искомый решающий путь.

Для изображения процесса редукции задач и получающихся при этом альтернативных множеств подзадач используются обычно графоподобные структуры, вершины которых представляют описания задач и подзадач, а каждая дуга связывает пару вершин, соответствующих редуцируемой задаче и одной из результирующих подзадач, причем стрелки на дугах указывают направление редукции. Пример такой структуры приведен на рис. 2.11,а: задача G может быть решена путем решения либо задач $D_1$ и $D_2$, либо $E_1$, $E_2$ и $E_3$ либо задачи F. При этом ребра, относящиеся к одному и тому же множеству подзадач, связываются специальной дугой. Чтобы сделать такую структуру более наглядной, вводятся дополнительные промежуточные вершины, и каждое множество результирующих задач группируется под своей родительской вершиной. При этом структура на рис. 2.11,а преобразуется в структуру, изображенную на рис. 2.11,б: для двух из трех альтернативных множеств подзадач добавлены соответственно вершины D и E.

Если считать, что вершины D и E соответствуют описаниям альтернативных путей решения исходной задачи, то вершину G можно назвать ИЛИ-*вершиной*, так как задача G разрешима или способом D, или способом E, или способом F. Аналогично вершины D и E можно назвать И-*вершинами*, поскольку каждый из соответствующих им способов требует решения всех подчиненных задач, что и обозначается специальной дугой. По этой причине структуры, подобные структурам, изображенным на рис.10(б) и рис.11, называются И/ИЛИ-*графами* (или *графами целей*).

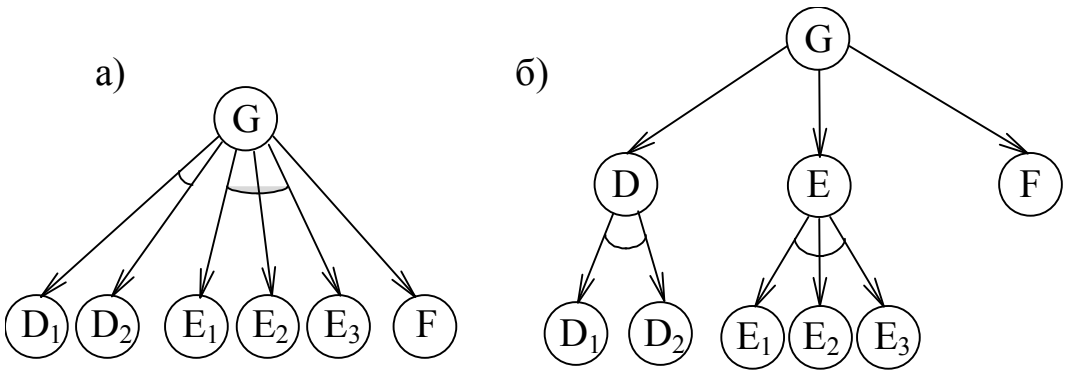



*Рис. 2.11. Схематическое изображение редукции задач*

Если некоторая вершина такого графа имеет непосредственно следующие за ней (дочерние) вершины, то либо все они являются И-вершинами, либо все они – ИЛИ-вершины. Заметим, что если у некоторой вершины И/ИЛИ-графа имеется ровно одна дочерняя вершина, то последнюю можно считать как И-вершиной, так и ИЛИ-вершиной – такова как, например, вершина F на рис. 2.11,б .

На языке И/ИЛИ-графов применение нескольких операторов редукции задачи будет означать, что сначала будет построена промежуточная И-вершина, а затем непосредственно следующие за ней ИЛИ-вершины подзадач. Исключение составляет случай, когда множество задач состоит только из одного элемента, в этом случае будет образована ровно одна вершина, будем для определенности считать ее ИЛИ-вершиной.

Вершину И/ИЛИ-графа, соответствующую описанию исходной задачи, будем называть *начальной* вершиной. Вершины же, которые соответствуют описаниям элементарных задач, будем называть *заключительными* вершинами. В графе, показанном на рис. 2.12, начальной является вершина $P_0$, а заключительными – вершины $P_1$, $P_4$, $P_5$, $P_7$ и $P_8$ (они изображены жирными кружками).

Поиск решения задачи, осуществляемый путем перебора вершин графа, применим и в подходе, основанном на редукции задач. Цель поиска на И/ИЛИ-графе – показать, что *разрешима* исходная задача, т.е. начальная вершина. Разрешимость этой вершины зависит от разрешимости других вершин графа.

Сформулируем общее рекурсивное *определение разрешимости вершины в И/ИЛИ-графе*:

заключительные вершины разрешимы, так как они соответствуют элементарным задачам;

ИЛИ-вершина, не являющаяся заключительной, разрешима тогда и только тогда, когда разрешима по крайней мере одна из ее дочерних вершин;

И-вершина, не являющаяся заключительной, разрешима тогда и только тогда, когда разрешима каждая из ее дочерних вершин.

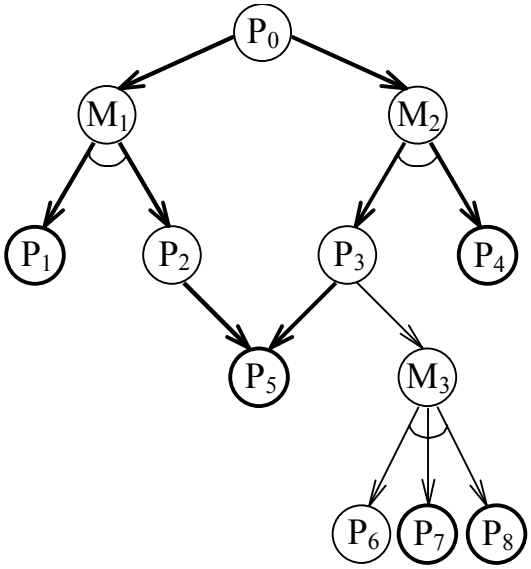



*Рис. 2.12. Пример И/ИЛИ-графа*

Если в процессе поиска удалось показать, что начальная вершина разрешима, то это значит, что обнаружено решение исходной задачи, которое заключено в так называемом решающем графе. *Решающий граф* – это подграф И/ИЛИ-графа, состоящий только из разрешимых вершин и доказывающий разрешимость начальной вершины.

Для И/ИЛИ-графа, изображенного на рис. 2.12, разрешимыми являются (кроме заключительных) вершины $M_1$, $M_2$, $P_2$, $P_3$. Этот граф содержит два решающих графа: первый состоит из вершин $P_0$, $M_1$, $P_1$, $P_2$ и $P_5$ ; а второй – из вершин $P_0$, $M_2$, $P_3$, $P_4$ и $P_5$. Заметим, что вершины $M_3$ и $P_6$ не являются разрешимыми ($M_3$ неразрешима в силу неразрешимости вершины $P_6$).

В качестве еще одного примера применения метода редукции рассмотрим решение задачи символьного интегрирования, т.е. нахождения неопределен-

ного интеграла $\int F(x)dx$. Обычно эта задача решается путем последовательного преобразования интеграла к выражению, содержащему известные табличные интегралы. Для этого используется несколько правил интегрирования, в том числе: правило интегрирования суммы функций, правило интегрирования по частям, правило вынесения постоянного множителя за знак интегрирования, а также применение алгебраических и тригонометрических подстановок и использование различных алгебраических и тригонометрических тождеств.

Для формализации этой задачи в рамках подхода, основанного на редукции задач, необходимо определить форму описания задач/подзадач, операторы редукции и элементарные задачи. В качестве возможной формы описания задач может быть взята символьная строка, содержащая запись подынтегральной функции и переменной интегрирования (если последняя не фиксирована заранее). Операторы редукции будут основаны, очевидно, на упомянутых правилах интегрирования. Например, правило интегрирования по частям $\int udx = u\int dv - \int vdu$ сводит исходную задачу (неопределенный интеграл в левой части равенства) к двум подзадачам интегрирования (два соответствующих интеграла в правой части равенства).

Заметим, что часть получаемых таким образом операторов редукции (как, например, операторы, соответствующие правилу интегрирования по частям или и правилу интегрирования суммы функций), действительно редуцируют исходную задачу и порождают И-вершину в И/ИЛИ-графе, в то время как алгебраические и тригонометрические подстановки и тождества (как, например, деление числителя на знаменатель или дополнение до полного квадрата) лишь заменяют одно подынтегральное выражение на другое, порождая, таким образом, ИЛИ-вершины.

Элементарные задачи интегрирования соответствуют табличным интегралам, например: $\int \sin x\, dx = -\cos x + C$

Отметим, что поскольку каждая из таких табличных формул содержит переменные, на самом деле она является схемой, задающей бесконечное множество элементарных задач.

Особенностью задачи интегрирования является то, что, например, при интегрировании по частям может оказаться несколько способов разбиения исходного подынтегрального выражения на части и соответственно несколько способов применения этого правила интегрирования. Это означает, что в общем случае для одного правила возможно несколько вариантов редукции задачи, т.е. несколько способов применения одного и того же оператора редукции.

Другая особенность рассматриваемой задачи состоит в том, что на каждом шаге процесса редукции применимо обычно большое количество операторов (включая несколько применений одного и того же оператора), и получающийся И/ИЛИ-граф задачи слишком велик даже для несложных задач интегрирования. Поэтому, чтобы сделать поиск на таком графе достаточно эффективным, необходимо как-то ограничивать и/или упорядочивать множество порождаемых при поиске вершин. К примеру, можно упорядочить операторы редукции по степени их полезности, и приписать больший приоритет операторам, соответствующим правилам интегрирования суммы и интегрирования по частям.

На рис. 2.13 показан решающий граф для одной задачи интегрирования. В вершинах графа указаны соответствующие задачи/подзадачи, заключительные вершины заключены в двойные рамки. Решение задачи может быть собрано из содержащейся в графе информации. Оно состоит из следующих шагов:

применения подстановки $z=\mathrm{tg}(x)$;

эквивалентного преобразования подынтегрального выражения;

применения правила интегрирования суммы функций, причем одна из трех результирующих подзадач оказывается элементарной, а две остальные решаются за один шаг (первая – путем вынесения постоянного множителя за знак интеграла, вторая – применением подстановки $z=\mathrm{tg}(w)$).

Подход, основанный на редукции задач, применим и имеет преимущества по сравнению с подходом, использующим представление в пространстве состояний, когда получающиеся при редукции подзадачи можно решать независимо друг от друга, как в примере с интегрированием. Впрочем, это условие

взаимной независимости результирующих задач можно несколько ослабить. Метод редукции применим также, если решения получающих подзадач зависят друг от друга, но при этом существует такой порядок их редукции, при котором найденные решения одних, более ранних подзадач не разрушаются при решении других, более поздних – как в головоломке о ханойской башне, в которой важен лишь порядок решения выделяемых подзадач.

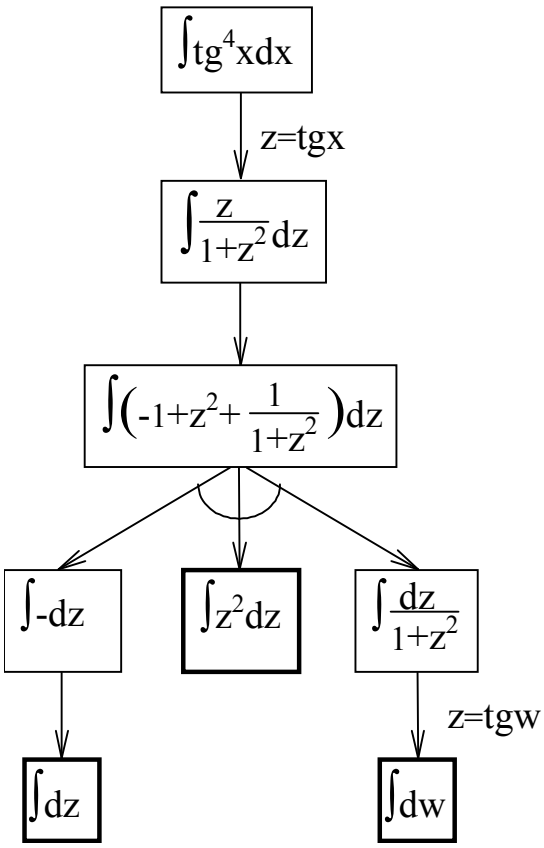



*Рис. 2.13. Решающий граф для одной задачи интегрирования*

В дополнение отметим, что подход с использованием пространства состояний можно рассматривать как вырожденный случай подхода, основанного на редукции задач, так как применение оператора в пространстве состояний сводит обычно исходную задачу к несколько более простой задаче, т.е. редуцирует ее. При этом результирующее множество подзадач состоит только из одного элемента, т.е. мы имеем простейший случай замены редуцируемой задачи на ей эквивалентную.

## 2.7. Нечеткий логический вывод

Нечеткий логический вывод – это алгоритм, позволяющий на основании базы знаний из совокупности нечетких продукционных правил по заданному $\vec{x} = \vec{x}_0$ определить заключение $z = z_0$.

Проиллюстрируем механизм нечеткого логического вывода на примере вычисления значений некоторой функции $z = f(x, y)$. Допустим, имеется база знаний, состоящая из двух правил:

П1: если $x$ есть $A_1$ и $y$ есть $B_1$, то $z$ есть $C_1$,

П2: если $x$ есть $A_2$ и $y$ есть $B_2$, то $z$ есть $C_2$,

где $A_i$, $B_i$, $C_i$ – нечеткие переменные, определенные на $x$, $y$, $z$ и имеющие функции принадлежности $\mu_{A_i}(x)$, $\mu_{B_i}(x)$ и $\mu_{C_i}(x)$ соответственно.

Отметим, что рассматриваемая задача легко обобщается на случай произвольного числа входных переменных и продукционных правил.

Требуется: по конкретным значениям $x = x_0$ и $y = y_0$ определить $z_0$.

Логический вывод осуществляется за следующие четыре этапа.

1. Введение нечеткости (фаззификация, fuzzification).

Для четко заданных входных значений рассчитываются принадлежности к отдельным термам.

Для рассматриваемого примера определяют численные значения

$\mu_{A_i}(x_0),\ \mu_{A_i}(y_0),\ i = 1, 2$.

2. Нечеткая импликация. Находят функции принадлежности предпосылок каждого отдельного правила при конкретных входных сигналах $x_0$ и $y_0$.

$$\alpha(i) = \mu_{Ai}(x_0) \cap \mu_{Bi}(y_0), \quad i = 1, 2.$$

Затем находят результирующие функции принадлежности каждого правила

$$\mu_i(z) = \alpha_i \cap \mu_{Ci}(z), \quad i = 1, 2.$$

3. Нечеткая композиция. Находится результирующая функция принадлежности всей совокупности правил при входных сигналах $x_0$ и $y_0$.

$$\mu_\Sigma(z) = \mu_1(z) \cup \mu_2(z)$$

4. Приведение к четкости (дефаззификация, defuzzification).

Используется, если необходимо преобразовать выходную функцию принадлежности в конкретное четкое число $z_0$.

Имеется большое количество методов дефаззификации, одним из наиболее распространенных является центроидный:

$$z_0 = \frac{\int\limits_R z \cdot \mu_\Sigma(z) \partial z}{\int\limits_R \mu_\Sigma(z) \partial z}.$$

Если в приведенном выше алгоритме логические операции пересечение реализуется как min, а объединение как max, то данный алгоритм называется алгоритмом Мамдани (в некоторых источниках алгоритмом Мамдани-Заде); его графическая иллюстрация приведена на рис. 2.14.

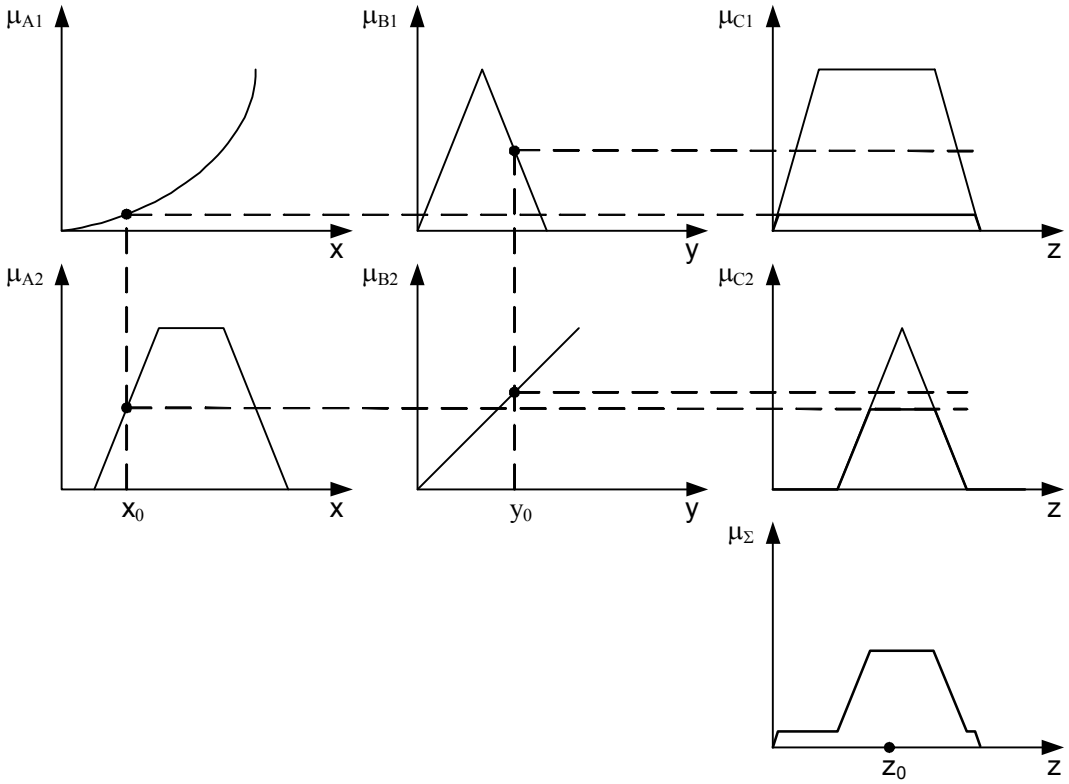



*Рис. 2.14. Иллюстрация работы алгоритма Мамдани*

На практике широкое распространение получил алгоритм нечеткого логического вывода Сугэно (Sugeno), в некоторых источниках известный как алгоритм Такаги-Сугено-Канга (TSK), отличающийся простотой вычислений.

Продукционные правила в рассматриваемом алгоритме имеют вид:

если $x_1$ есть $A_1$ и если $x_2$ есть $A_2$ и … и если $x_n$ есть $A_n$ ,

то $z=f(x_1, x_2, \ldots x_n)$,

где $f(\vec{x})=f(x_1, x_2, \ldots x_n)$ – обычная четкая функция.

Принципиальное отличие от алгоритма Мамдани в данном случае касается заключения, которое представляется в форме функциональной зависимости.

Реализация алгоритма Сугэно состоит из трех шагов.

1. Введение нечеткости (фаззификация). Полностью аналогично алгоритму Мамдани.

2. Находятся функции принадлежности предпосылок каждого отдельного правила при конкретных входных сигналах $\vec{x}_0$: $\alpha_i$, $i = 1, 2, \ldots M$ , где $M$ – число продукционных правил.

В классическом алгоритме Сугэно логические операции пересечение реализуется как min.

3. Определяется четкое значение выходной переменной:

$$z_0 = \frac{\sum_{i=1}^{M} \alpha_i \cdot f_i(\vec{x}_0)}{\sum_{i=1}^{M} \alpha_i},$$

где $i$ – текущий номер правила.

В качестве функции $f(\vec{x})$ часто используются полиномы 0-го или 1-го порядка: $f(\vec{x}) = c$ , $f(\vec{x}) = c + \sum_{j=1}^{n} p_j \cdot x_j$ .

В таких случаях обычно используют названия: алгоритм Сугэно 0-го и 1-го порядка соответственно.

В алгоритме Сугэно 0-го порядка (называемом также упрощенным алгоритмом нечеткого вывода) продукционные правила имеют вид:

если $x_1$ есть $A_1$ и если $x_2$ есть $A_2$ и … и если $x_n$ есть $A_n$ , то $z=c_i$,

а значение выходной переменной вычисляется по формуле:

$$z_0 = \frac{\sum_{i=1}^{M} \alpha_i \cdot c_i}{\sum_{i=1}^{M} \alpha_i}.$$

Отметим, что алгоритм Сугэно 0-го порядка можно интерпретировать, как частный случай алгоритма Мамдани, когда функции принадлежности заключений правил имеют вид:

$$\mu_i(x) = \begin{cases} 1, \textit{при } x = c_i, \\ 0, \textit{при } x \neq c_i. \end{cases}$$

Графическая интерпретация алгоритма Сугэно 0-го порядка в случае двух входных переменных и базы знаний из двух продукционных правил приведена на рис. 2.15.

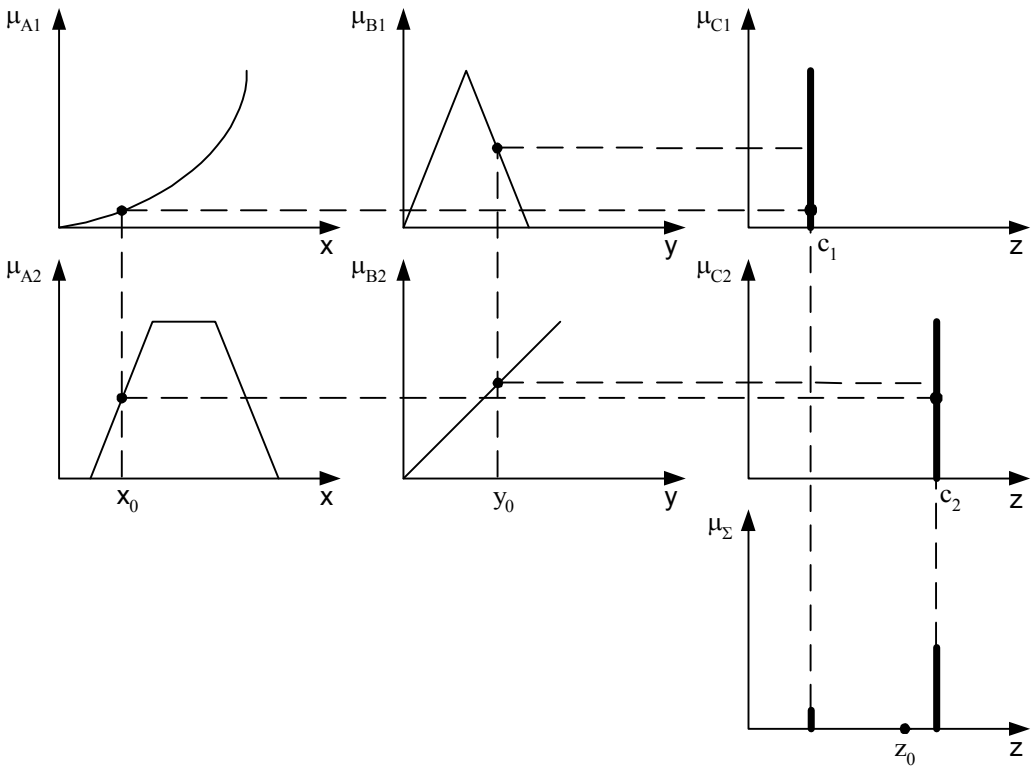



*Рис. 2.15. Иллюстрация работы алгоритма Сугэно 0-го порядка*

Известно большое число алгоритмов нечеткого вывода, различающихся набором исходных правил, видом функций принадлежности, способами нечеткой импликации и композиции, а также методом приведения к четкости.

Рассмотрим наиболее распространенные методы приведения к четкости.

Выше был приведен один из наиболее распространенных методов – центроидный; обозначим результат, полученный с помощью этого метода через $z_c$.

На практике также часто встречаются:

минимальный максимум: результат $z_{cl}$ – наименьшая точка, в которой $\mu_{\Sigma}(z)$ достигает максимума;

максимальный максимум: результат $z_{cr}$ – наибольшая точка, в которой $\mu_{\Sigma}(z)$ достигает максимума;

средний максимум:

$$z_{CM} = \frac{\sum_{i=1}^{m} z_i}{m},$$

где $z_i$ – точки, в которых $\mu_{\Sigma}(z)$ принимает максимальное значение;

высотная дефаззификация: элементы области определения $R$, для которых значения функции принадлежности меньше, чем некоторый уровень $\alpha$, в расчет не принимаются, четкое значение определяется по формуле

$$z_{ch} = \frac{\int_{C_\alpha} z \cdot C(z) \cdot dz}{\int_{C_\alpha} C(z) \cdot dz},$$

где $C_\alpha$ – нечеткое множество $\alpha$ уровня.

Графическая интерпретация приведенных методов деффазификации приведена на рис. 2.16.

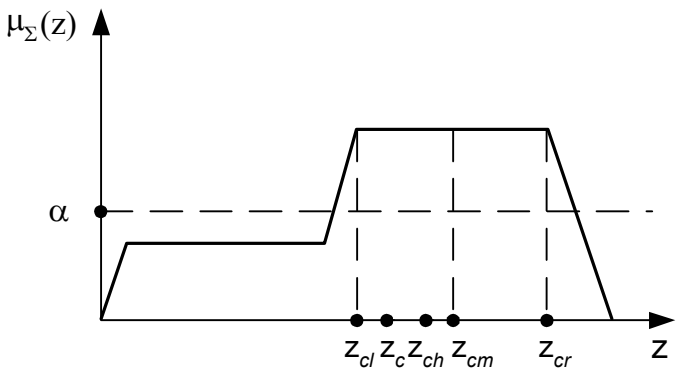



*Рис. 2.16. Иллюстрация методов деффазификации*

Приведем еще один способ деффазификации – относительно средних центров:

$$Z_{CA} = \frac{\sum_{i=1}^{M} \alpha_i \cdot c_i}{\sum_{i=1}^{M} \alpha_i},$$

где $\alpha_i$ – значения функции принадлежности предпосылки i-го правила,

$c_i = \int_R z \cdot \mu_i(z) dz / \int_R \mu_i(z) dz$ – центр заключения *i*-го правила,

$\mu_i(z)$ – функция принадлежности заключения i-го правила.

Замечательной особенностью данного способа деффазификации является то, что при его применении алгоритм с нечеткими правилами Мамдани, по сути, сводится к алгоритму Сугэно 0-го порядка.

Эффективность применения аппарата нечеткой логики базируется на ряде теорем, смысл которых сводиться к тому, что система на основе алгоритма нечеткого вывода, при выполнении определенных, не очень жестких условий, является универсальным аппроксиматором.

## 2.8. Алгоритм поиска пути на графе

В ряде случаев при наличии графа, описывающего полное пространство состояний, возникает задача поиска за минимальное время короткого пути, ведущего из начальной ситуации $x_0$ в одно из конечных $x_t$. Для ее решения существует несколько алгоритмов, один из которых состоит в следующем.

Рассматриваемый алгоритм определяет расстояние между вершинами в простом орграфе с неотрицательными весами. Алгоритм, начиная из вершины $x_0$ , просматривает граф в ширину, помечая вершины $x_j$ значениями – метками их расстояний от $x_0$ . Временная метка вершины $x_j$ – это минимальное расстояние от $x_0$ до $x_j$ , когда в определении пути на графе учитываются не все маршруты из $x_0$ в $x_j$ .Окончательная метка $x_j$ содержит минимальное рас-

стояние на графе от $x_0$ до $x_j$ . Таким образом, в каждый момент времени работы алгоритма некоторые вершины будут иметь окончательные метки, а остальная их часть – временные. Алгоритм заканчивается, когда вершина $x_t$ получает окончательную метку, то есть расстояние от $x_0$ до $x_t$. Вначале вершине $x_0$ присваивается окончательная метка 0 (нулевое расстояние до самой себя), а каждой из остальных | X | – 1 вершин присваивается временная метка $\infty$ (бесконечность). На каждом шаге алгоритма одной вершине с временной меткой присваивается окончательная и поиск продолжается дальше. На каждом шаге вер- шины меняются следующим образом.

1. Если имеется дуга из $x_i$ в $x_j$ , то каждой вершине $x_j$ , не имеющей окончательной метки, присваивается новая временная метка – наименьшая из ее временной и числа ($w_{ij}$ +окончательная метка $x_i$), где $x_i$ – вершина, которой присвоена окончательная метка на предыдущем шаге.

2. Определяется наименьшая из всех временных меток, которая становится окончательной меткой своей вершины, в случае равенства меток выбирается любая из них.

Циклический процесс п.1, п.2 продолжается до тех пор, пока вершина $x_t$ не получит окончательной метки, которая представляет собой короткий путь от этой вершины до начала $x_0$. Рассмотрим работу представленного выше алгоритма для графа, заданного в виде матрицы весов графа (табл. 2.1).

***Таблица 2.1***

| | ***X0*** | ***X1*** | ***X2*** | ***X3*** | ***X4*** | ***X5*** | ***X6*** |
|---|---|---|---|---|---|---|---|
| ***X0*** | 0 | 7 | 2 | 0 | 0 | 0 | 0 |
| ***X1*** | 0 | 0 | 0 | 1 | 5 | 0 | 10 |
| ***X2*** | 0 | 3 | 0 | 5 | 0 | 0 | 0 |
| ***X3*** | 0 | 0 | 0 | 0 | 3 | 7 | 0 |
| ***X4*** | 0 | 0 | 8 | 0 | 0 | 5 | 2 |
| ***X5*** | 0 | 0 | 0 | 0 | 5 | 0 | 6 |
| ***X6*** | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Граф со структурой, соответствующей данной матрице представлен на рис. 2.17.

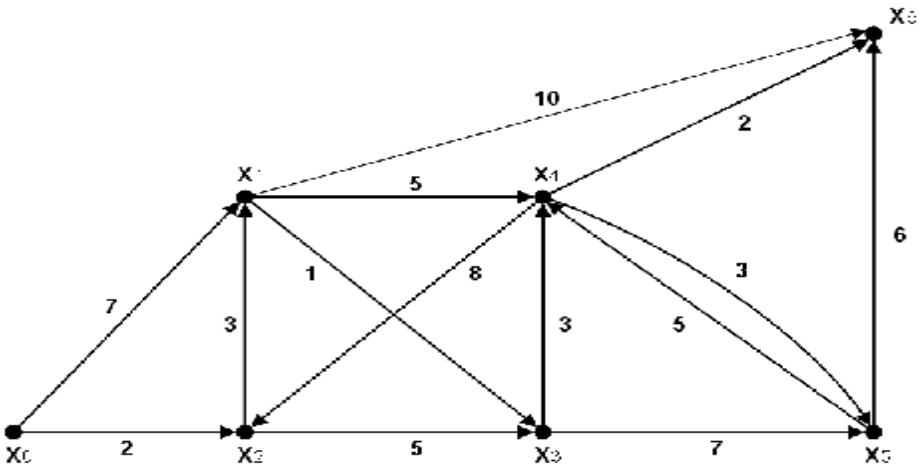



*Рис. 2.17. Граф для демонстрации работы алгоритма*

***Таблица 2.2***

| Шаг | X0 | X1 | X2 | X3 | X4 | X5 | X6 |
|---|---|---|---|---|---|---|---|
| 0 | 0 | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ |
| 1 | | 7 | 2 | ∞ | ∞ | ∞ | ∞ |
| 2 | | 5 | | 7 | ∞ | ∞ | ∞ |
| 3 | | | | 6 | 10 | ∞ | 15 |
| 4 | | | | | 9 | 13 | 15 |
| 5 | | | | | | 12 | 11 |

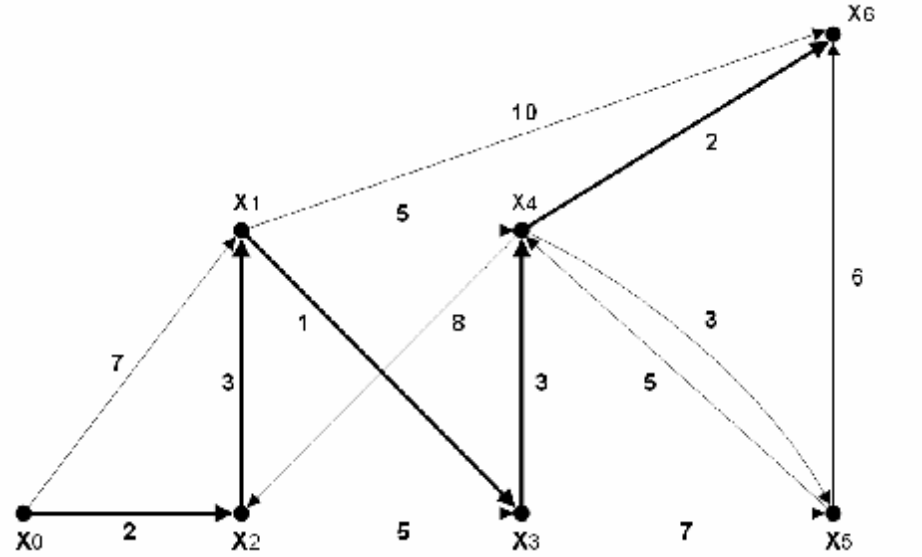



*Рис. 2.18. Представление найденного пути на графе примера*

# 3. МЕТОД АНАЛИЗА ИЕРАРХИЙ

Опыт использования интеллектуальных информационных систем в различных сферах целенаправленной человеческой деятельности привел к пониманию многих принципиальных трудностей, возникающих при их внедрении в реальную практику, сотканную из непрерывной череды актов принятия решений. Оказалось, что лицо, принимающее решение (ЛПР), при принятии решения учитывает огромное число разнообразных показателей, представить которые в виде единственного критерия удается только в редких случаях. Стало ясно, что методики естественных наук, успешно применявшиеся при моделировании технологического уровня интеллектуальных информационных систем, совершенно недостаточно для решения более сложных проблем, которые по сути своей многокритериальны. При поиске «наилучшего» плана или альтернативы существенное значение имеют факторы, не поддающиеся формализации (социальные, организационные, политические, психологические и т.п.). Поэтому руководитель (ЛПР), предложенное ему специалистом по математическому моделированию, и понимающий, что неформализуемые факторы могут оказать более сильное воздействие на результат, чем, например, оптимальное распределение ресурсов, склонен отнестись скептически к такому решению, не учитывающему ясные для ЛПР возможности повышения эффективности принимаемых решений. Если, кроме того, учесть, что ЛПР обычно имеет в голове (но не в модели!) огромное число ограничений, которые он не хотел бы нарушить, то станет ясно, почему он склонен принять собственное решение, отличное от полученного с помощью компьютера.

Один из способов практического преодоления перечисленных трудностей состоит во включении ЛПР в процесс построения моделей и принятия решений на их основе. Для этого предназначены человеко-машинные интеллектуальные системы системы. Одним из классов таких систем являются системы поддержки принятия решений (СППР), в рамках которых опыт и неформализованные знания ЛПР сочетаются с математическим исследованием.

СППР, основанная на методе анализа иерархий (МАИ), является простым и удобным средством, которое поможет структурировать проблему, построить набор альтернатив, выделить характеризующие их факторы, задать значимость этих факторов, оценить альтернативы по каждому из факторов, найти неточности и противоречия в суждениях ЛПР/эксперта, проранжировать альтернативы, провести анализ решения и обосновать полученные результаты.

СППР МАИ может использоваться при решении следующих типовых задач:

оценка качества организационных, проектных и конструкторских решений;

определение политики инвестиций в различных областях;

задачи размещения (выбор места расположения вредных и опасных производств, пунктов обслуживания);

распределение ресурсов;

проведение анализа проблемы по методу "стоимость-эффективность";

стратегическое планирование;

проектирование и выбор оборудования, товаров;

выбор профессии, места работы, подбор кадров.

Основные положения метода анализа иерархий были разработаны известным американским математиком Т.Л.Саати и опубликованы в 1977 г.

МАИ используется для решения слабо структуризованных и неструктуризованных проблем. Методология решения таких проблем опирается на системный подход, при котором проблема рассматривается как результат взаимодействия и, более того, взаимозависимости множества разнородных объектов, а не просто как их изолированная и автономная совокупность.

Человеку присущи два характерных признака аналитического мышления: один – умение наблюдать и анализировать наблюдения, другой – способность устанавливать отношения между наблюдениями, оценивая уровень (интенсивность) взаимосвязей, а затем синтезировать эти отношения в общее восприятие наблюдаемого.

На основе этих свойств человеческого мышления были сформулированы три принципа, реализация которых и является содержанием МАИ:

принцип идентичности и декомпозиции;

принцип дискриминации и сравнительных суждений;

принцип синтеза.

Реализация принципа идентичности и декомпозиции осуществляется на первом этапе применения МАИ, в котором предусматривается структурирование проблемы в виде иерархии. Иерархия строится с вершины – это общая цель или фокус проблемы. В общем случае целей может быть несколько. За фокусом следует уровень наиболее важных критериев. Каждый из критериев может разделяться на субкритерии, за которыми следует уровень альтернатив. ЛПР при построении иерархии вынужден вникнуть в проблему. От этого этапа во многом зависят конечные результаты принятия решений. Формирование множества альтернатив и критериев осуществляется с учетом рекомендаций. Этап является неформализуемым.

*Пример.* При обсуждении проблемы улучшения жилищных условий семьей была сформулирована цель – покупка дома. Обсуждались и другие цели решения этой проблемы (например, ремонт имеющегося жилья). Из каталога были отобраны три наиболее предпочтительных дома (варианты A, B, C), которые и были осмотрены семьей непосредственно. Для выбора окончательного варианта она решила воспользоваться методом анализа иерархий. Итогом первого этапа МАИ, который явился результатом семейного обсуждения, стала иерархия, представленная на рис. 3.1.

Иерархия – есть определенный тип системы, основанный на предположении, что элементы системы могут группироваться в несвязанные множества. Элементы каждой группы находятся под влиянием элементов другой группы и в свою очередь оказывают влияние на элементы следующей группы. Считается, что элементы в каждой группе иерархии (называемые уровнем, кластером, стратой) независимые.

Рассмотрим общий вид иерархии (рис. 3.2).

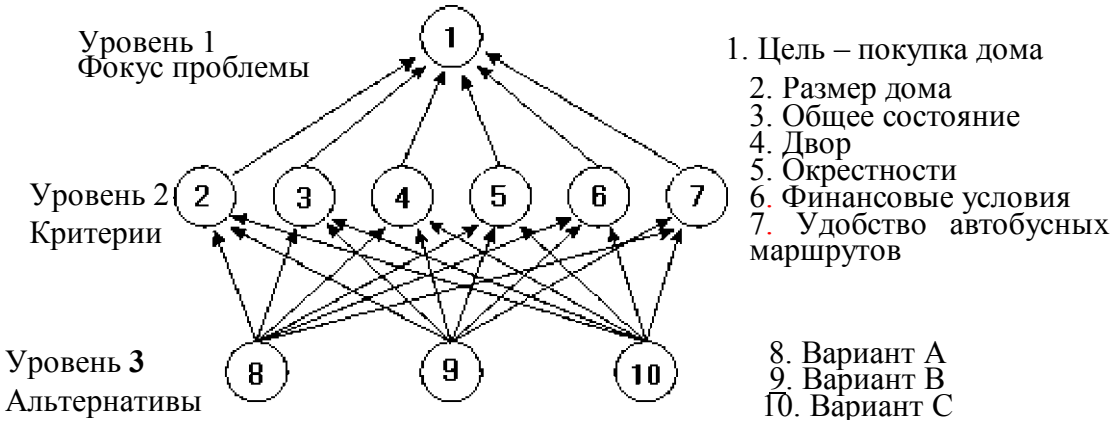



*Рис. 3.1. Иерархия проблемы улучшения жилищных условий*

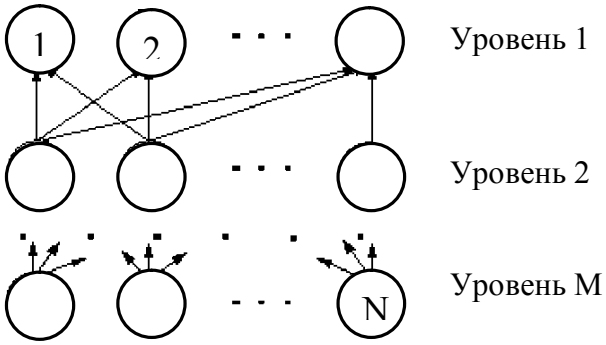



*Рис. 3.2. Общий вид иерархии*

Математически иерархия и ее свойства могут быть описаны следующим образом. На множестве объектов $i=\{1,2,...,N\}$ определяется иерархическая структура путем задания орграфа $G = (i,W)$, $W \subset i \times i$, который:

a) разбивает вершины на непересекающиеся уровни

$$i = U_i V_j;\ \ l = \overline{1,m};\ \ V_i \cap V_j = \varnothing;\ \ i, j = \overline{1,m}.$$

b) $(i, j) \in W$ означает, что вес $Z_i$ объекта $i$ непосредственно зависит от веса $Z_j$ объекта $j$;

c) если $(i, j)$ – дуга графа $G$, т. е. $(i, j) \in W$, то объекты $i$ и $j$ находятся на смежных уровнях, т. е. найдется такое $k$, что $i \in V_{k+1}, j \in V_k$.

d) веса объекта $i \in V_{k+1}$ определяются через веса $Z_j$ вершин множества $L_i = \{j \mid (i, j) \in W\} \subseteq V_k$, в которые ведут дуги из вершины $i$ с помощью феноменологически вводимой зависимости $Z_i = \sum_{j \in L} \vartheta_{ij} Z_j$, где $\vartheta_{ij}$ – вес дуги $(i, j)$.

Принцип дискриминации и сравнительных суждений реализуется на втором этапе МАИ. Суть его заключается в том, что, используя суждения ЛПР/эксперта и определенные алгоритмы их обработки, устанавливаются веса $\vartheta_{ij}$ дуг $(i, j) \in W$ и веса $Z_j$ объектов первого уровня $(j \in V1)$. Если на первом уровне один объект, то вес его принимается за 1.

Суждения ЛПР/эксперта являются результатом исследования его структуры предпочтений. При этом исследовании применяется метод парных сравнений, содержание которого состоит в следующем. Пусть задано некоторое фиксированное множество объектов $K = \{k_i\}, i = \overline{1,n}, K \subset L$, которые сравниваются попарно с точки зрения их предпочтительности, желательности, важности ит. п. Результаты записываются в виде матрицы парных сравнений $R = \{r_{ij}\}, i, j = \overline{1,n}$.

Результат сравнения отражает не только факт, но и степень (силу, интенсивность и т.п.) превосходства. При этом используется шкала относительной важности, выбор которой зависит от следующих требований:

шкала должна давать возможность улавливать различия в ощущениях людей, когда они проводят сравнение;

диапазон измеряемой интенсивности шкалы должен соответствовать результатам когнитивной психологии.

Удовлетворяет этим требованиям шкала, приведенная в табл. 3.1.

***Таблица 3.1***

| **Количественная оценка интенсивности относительной важности** | **Качественная оценка интенсивности относительной важности** | **Пояснения** |
|---|---|---|
| 1 | Равная важность | Равный вклад двух объектов |
| 3 | Умеренное превосходство одного над другим | Опыт и суждения дают легкое превосходство одного объекта над другим |
| 5 | Существенное или сильное превосходство | Опыт и суждения дают сильное превосходство одного объекта над дру- |

| | | |
|---|---|---|
| | | гим |
| 7 | Значительное превосходство | Один объект имеет настолько сильное превосходство, что оно становится практически значительным |
| 9 | Очень сильное превосходство | Очевидность превосходства одного объекта над другим подтверждается наиболее сильно |
| 2,4,6,8 | Промежуточные решения между двумя соседними суждениями | Применяются в компромиссном случае |
| Обратные величины приведенных выше чисел | Если объекту *i* при сравнении с объектом *j* приписывается одно из приведенных выше чисел, то действию *j* при сравнении с *i* приписывается обратное значение | |

Из шкалы следует свойство гомогенности (однородности) объектов. Это свойство соответствует способности людей сравнивать объекты, которые не слишком сильно отличаются друг от друга. Гомогенность существенна для сравнения объектов одного порядка, т.к. человеческий разум склонен к допущению больших ошибок при сравнении несопоставимых элементов. Когда эта несопоставимость большая, объекты располагаются в отдельные кластеры сравниваемых размеров, что выдвигает идею об уровнях и их декомпозиции.

Рассмотрим метод парных сравнений на примере покупки дома.

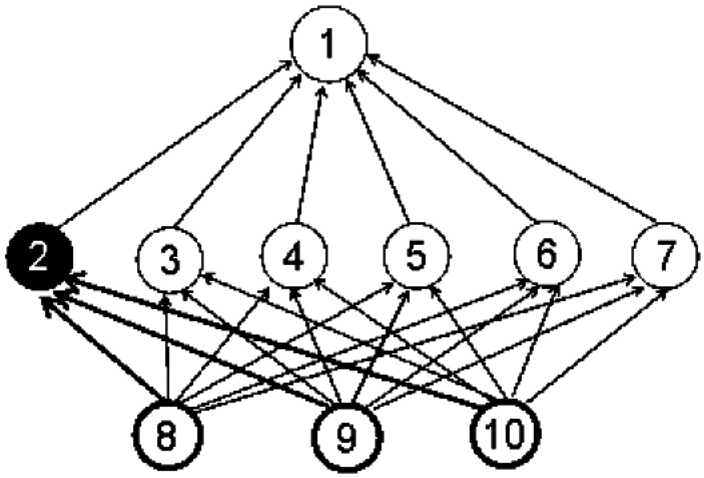



*Рис. 3.3. Иллюстрация к методу парных сравнений*

Допустим необходимо оценить предпочтения ЛПР/эксперта на множестве вариантов A, B, C относительно критерия – размера дома. Лучше всего эту задачу свести к заполнению матрицы парных сравнений в виде таблицы 3.2.

***Таблица 3.2***

| Размер дома | Вариант A | Вариант B | Вариант C |
|---|---|---|---|
| Вариант A | 1 | 1/3 | 5 |
| Вариант B | 3 | 1 | 1/7 |
| Вариант C | 1/5 | 7 | 1 |

Размерность таблицы определяется количеством дуг, которые входят в рассматриваемую вершину. Элементы таблицы $j$ $i, j = 1, 3$ являются количественной оценкой интенсивности предпочтения $i$-го объекта, находящегося в $i$-й строке, относительно $j$-го объекта, находящегося в $j$-м столбце, в соответствии с выше рассмотренной шкалой. При этом сравнении ЛПР/эксперту задавался следующий вопрос: насколько один вариант (например A) превосходит по размеру другой вариант (например C)? Ответом ЛПР/эксперта, как следует из таблицы, было следующее суждение: существенное или сильное превосходство.

Таким же образом осуществляется оценка предпочтений ЛПР/эксперта относительно остальных критериев путем заполнения еще пяти аналогичных матриц размерностью 3x3. После чего метод парных сравнений распространяется на множество самих критериев относительно цели – покупки дома. В этом случае ЛПР/эксперту задается следующий вопрос: насколько важнее один критерий (например, размер дома) для реализации цели по сравнению с другим (например, финансовые условия)? Как следует из иерархии, размерность этой таблицы 6x6.

Принимая во внимание свойство матрицы, т. е.:

$$\forall i, j = \overline{1,n},\ r_{ij} = 1/r_{ji}$$

и, как следствие, $r_{ii} = 1$, количество вопросов равно $n \times (n-1)/2$. Формализацией понятия непротиворечивости для метода парных сравнений является выполнение следующего равенства:

$$r_{ij}^{*} = r_{ik}^{*} \cdot r_{kj}^{*},\ \forall i, j, k, \tag{3.1}$$

где $r_{ij}^*$ – это элементы матрицы полученные в результате идеально согласованного эксперимента. Соотношение (3.1) соответствует правилу логического вывода, которое в этом случае формулируется следующим образом: если *i*-й объект предпочтительнее *k*-го объекта на $r_{ik}^*$ и *k*-й объект предпочтительнее *j*-го объекта на $r_{kj}^*$ ,то *i*-й объект предпочтительней *j*-го объекта на $r_{ij}^*$

Можно доказать, что если матрица $R^*$ обладает свойством (3.1), то тогда существуют такие числа $\vartheta_i^*$, что имеет место равенство

$$r_{ij}^* = \vartheta_i^* / \vartheta_j^*, \ \forall i, j = \overline{1,n}. \tag{3.2}$$

Числа $\vartheta_i^*, i = \overline{1,n}$ отождествляются с весами дуг (это множество *W* в графе *G*) либо с весами объектов первого уровня (это $Z_i, i \in V1$).

Матрица $R^*$ имеет единичный ранг, $\{\vartheta_i^*\}, i = \overline{1,n}$ собственный вектор матрицы, где *n* – соответствующее ей собственное число. Действительно,

$$\begin{pmatrix} \frac{\vartheta_1^*}{\vartheta_1^*} & \frac{\vartheta_1^*}{\vartheta_2^*} & \cdots & \frac{\vartheta_1^*}{\vartheta_n^*} \\ \frac{\vartheta_2^*}{\vartheta_1^*} & \frac{\vartheta_2^*}{\vartheta_2^*} & \cdots & \frac{\vartheta_{2n}^*}{\vartheta_n^*} \\ \cdots & \cdots & \cdots & \cdots \\ \frac{\vartheta_n^*}{\vartheta_1^*} & \frac{\vartheta_n^*}{\vartheta_2^*} & \cdots & \frac{\vartheta_n^*}{\vartheta_n^*} \end{pmatrix} \cdot \begin{pmatrix} \vartheta_1^* \\ \vartheta_2^* \\ \cdots \\ \vartheta_n^* \end{pmatrix} = \begin{pmatrix} n\vartheta_1^* \\ n\vartheta_2^* \\ \cdots \\ n\vartheta_n^* \end{pmatrix} = n \begin{pmatrix} \vartheta_1^* \\ \vartheta_2^* \\ \cdots \\ \vartheta_n^* \end{pmatrix} \text{ или } R^* \overline{\vartheta^*} = n\overline{\vartheta^*}. \tag{3.3}$$

Практически добиться полной согласованности (т.е. непротиворечивости) суждений ЛПР/эксперта далеко не всегда возможно. Поэтому в общем случае $r_{ij}$ будут отклоняться от «идеальных» $r_{ij}^* = \vartheta_i^* / \vartheta_j^*$, вследствие чего соотношения 1, 2, 3 не будут иметь место.

Для дальнейшего анализа полезными являются следующие два факта из теории матриц:

Во-первых, если $\lambda_1, ..., \lambda_n$ , являются собственными числами матрицы *R* и если $r_{ij} = 1, \ i = j = \overline{1,n}$, то $\sum_{i=1}^{n} \lambda_i = n.$ Согласно этому утверждению, если имеет место (3.3) (т.е. матрица является идеально согласованной), то все собственные числа ее – нули, за исключением одного, равного *n*.

Во-вторых, если элемент положительной обратносимметричной матрицы *R* незначительно изменить, то собственные числа этой матрицы также изменятся незначительно, т.е. они являются непрерывными функциями ее элементов.

Объединяя эти результаты, находим, что при малых изменениях $r_{ij}$, от $r_{ij}^*$ наибольшее собственное число $\lambda_{\max}$ (практически получаемой матрицы *R* при использовании метода парных сравнений) остается близким к *n*, а остальные собственные значения – близкими к нулю.

Отсюда можно сформулировать следующую задачу: для нахождения весов дуг или объектов первого уровня по полученной в результате метода парных сравнений матрице R необходимо определить собственный вектор $\overline{\vartheta}$, соответствующий максимальному собственному числу, т.е. решить уравнение:

$$R\ \overline{\vartheta} = \lambda_{\max}\overline{\vartheta}\ . \tag{3.4}$$

Так как малые изменения в $r_{ij}, i, j = \overline{1,n}$ вызывают малое изменение $\lambda_{\max}$, отклонение последнего от *n* является мерой согласованности. Она может быть выражена с помощью индекса согласованности (ИС):

$$ИС = \frac{\lambda_{\max} - n}{n-1}\ . \tag{3.5}$$

Если *ИС*< 0,1, то практически считается, что мера согласованности находится на приемлемом уровне.

Индекс согласованности матрицы парных сравнений, элементы которой сгенерированы случайным образом, называется случайным индексом (*СИ*). Ниже представлена таблица соответствия порядка и среднего значения СИ, определенная на базе 100 случайных выборок (табл. 3.3).

Отношение *ИС* к среднему *СИ* для матрицы того же порядка называется отношением согласованности (*ОС*). Значение *ОС* меньшее или равное 0,10 считается приемлемым. Обычно *ИС* и *ОС* указываются в процентах. Согласно определению, ИС можно трактовать как отклонение от идеально проведенного эксперимента (метода парных сравнений), а *ОС* указывает, на сколько оцениваемая степень согласованности сходится со степенью согласованности самого неидеально проведенного эксперимента.

***Таблица 3.3***

| Порядок матрицы | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| *СИ* | 0,00 | 0,00 | 0,58 | 0,9 | 1,12 | 1,24 | 1,32 | 1,41 | 1,45 | 1,49 |

Таким образом, МАИ допускает несогласованность (как неотъемлемую часть метода), признавая, что человеческие суждения находятся в постоянном процессе изменения и эволюции (поэтому не следует настаивать на 100% согласованности, так как суждения могут измениться после того, как проблема решена). Но надежные решения не могут быть приняты без приемлемого уровня согласованности.

Существуют два метода решения уравнения $RV = \lambda_{\max} \cdot V$. Это прямой и итерационный.

Рассмотрим *прямой метод*. Проверим алгоритм данного метода. $R$ - идеально согласованная матрица, т. е.

$$R = \begin{pmatrix} \frac{v_1}{v_1} & \frac{v_1}{v_2} & \cdots & \frac{v_1}{v_n} \\ \frac{v_2}{v_1} & \frac{v_2}{v_2} & \cdots & \frac{v_2}{v_n} \\ \cdots & \cdots & \cdots & \cdots \\ \frac{v_n}{v_1} & \frac{v_n}{v_2} & \cdots & \frac{v_n}{v_n} \end{pmatrix} = \left\{ \frac{v_i}{v_j} \right\}, \; i, j = \overline{1, n} .$$

1. Определим среднее геометрическое каждой строки матрицы $R$:

$$\sqrt[n]{\prod_{j=1}^{n} \frac{v_i}{v_j}} = \frac{v_i}{\sqrt[n]{\prod_{j=1}^{n} v_j}}, \; i = \overline{1, n}$$

2. Вычислим сумму средних геометрических:

$$\sum_{i=1}^{n} \frac{v_i}{\sqrt[n]{\prod_{j=1}^{n} v_j}} = \frac{\sum_{i=1}^{n} v_i}{\sqrt[n]{\prod_{j=1}^{n} v_j}} .$$

3. Разделим среднее геометрическое каждой строки матрицы $R$ на сумму средних геометрических строк:

$$\frac{v_i}{\sqrt[n]{\prod_{j=1}^{n} v_j}} \Bigg/ \frac{\sum_{i=1}^{n} v_i}{\sqrt[n]{\prod_{j=1}^{n} v_j}} = \frac{v_i}{\sum_{i=1}^{n} v_i};\ i = \overline{1,n},$$

т.е. получим нормированное значение собственного вектора.

Для получения $\lambda_{\max}$ выполним следующие шаги:

1. Определим сумму элементов для каждого столбца матрицы *R*:

$$\sum_{i=1}^{n} \frac{v_i}{v_j} = \frac{\sum_{i=1}^{n} v_i}{v_j}; j = \overline{1,n}.$$

2. Определим скалярное произведение векторов:

$$\sum_{i=1}^{n} \frac{v_i}{\sum_{i=1}^{n} v_i} \cdot \frac{\sum_{j=1}^{n} v_j}{v_i} = n,$$

что соответствует собственному числу для идеально согласованной матрицы.

Итерационный метод основан на следующей теореме.

Для положительной квадратной матрицы *R* собственный вектор *V*, соответствующий максимальному собственному значению $\lambda_{\max}$, с точностью до постоянного сомножителя *C* определяется по формуле:

$$\lim \frac{R^k e}{e^T \cdot R^k \cdot e} = CV,$$

где *e*=(1, 1, 1) – единичный вектор;

*k*=1, 2, 3,... показатель степени;

С – константа;

Т – знак транспонирования.

Вычисление собственного вектора *V* производится до достижения заданной точности $e^T \cdot \left|V^{(l)} - V^{(l-1)}\right| \le \xi,$

где *l* – номер итерации, такой, что *l* = 1 соответствует *k* = 1; *l* = 2, *k* = 2 и т.д.

Х – допустимая погрешность

С достаточной для практики точностью принимается равной 0,01 независимо от порядка матрицы.

Максимальное собственное значение вычисляется по формуле:

$$\lambda_{\max} = e^T RV.$$

Содержание третьего этапа заключается в реализации принципа синтеза. Искомые веса объектов определяются последовательно, начиная со второго уровня иерархии в соответствии с решающим правилом

$$Z_i = \sum_{j \in L} \vartheta_{ij} Z_j, \forall i \in V_2, ..., i \in V_m.$$

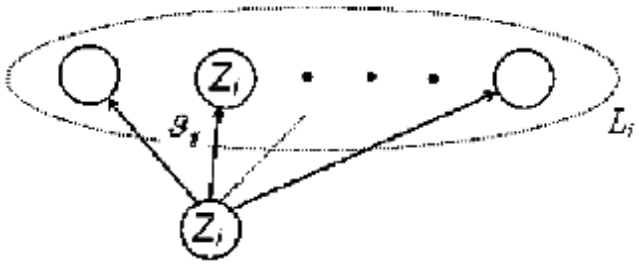



*Рис. 3.4. Фрагмент иерархии*

Веса объектов, принадлежащих уровню альтернатив, можно считать как результат измерения их в шкале отношений в диапазоне [0,1].

Согласованность всей иерархии *C* определяется по следующему выражению:

$$C = \frac{\sum_{\forall i \in D} ИС_1 Z_1}{\sum_{\forall i \in D} СИ_1 Z_1},$$

где $ИС_i, СИ_i$ – соответственно индекс согласованности и случайный индекс таблицы парных сравнений, рассмотренной относительно i-го объекта. Если $i \in V_1$ и $i$>1, то для $\forall i \in V_1$ $ИС_i = ИС_1$ и $СИ_i = СИ_1$; $ИС_1$ и $СИ_1$ – соответствующие параметры таблицы парных сравнений, которая была сформирована для определения весов объектов первого уровня.

Приемлемым является значение *C* меньше или равное 10%. В противном случае качество суждений следует улучшить. Возможно, следует пересмотреть формулировку вопросов при проведении парных сравнений. Если это не поможет улучшить согласованность, то, вероятно, задачу следует более точно структурировать.

Решающим преимуществом МАИ над большинством существующих методов оценивания альтернатив является вклад в анализ структуры проблемы и отчетливое выражение суждений.

Сложность, как было уже отмечено, характеризуется большим числом взаимодействий между многими субъективными и объективными факторами различного типа и степени важности, а также группами людей с различными целями и противоречивыми интересами. Эти факторы определяют вероятность или невозможность выбора одной из альтернатив, которая приемлема для всех с определенной степенью компромисса.

Чтобы разобраться с этой сложностью, нужна систематическая процедура для представления групп, их целей, критериев и поведения, обусловленных этими целями, альтернативных исходов и ресурсов, распределяемых по этим альтернативам. В МАИ эта процедура сводится к построению иерархии проблемы.

Общая цель (фокус) проблемы (например, выбор наилучшего автомобиля, построение наилучшей системы, распределение ресурса в соответствии с важностью) является обычно высшим уровнем иерархии. За фокусом следует уровень наиболее важных критериев (таких, как стоимость, стиль, комфортабельность и размеры автомобиля, или же в планировании -прибыльность инвестиции, конкуренция и т.д.). Каждый из критериев может разделяться на субкритерии. За субкритериями следует уровень альтернатив, число которых может быть очень большим.

С целью иллюстрации этапов МАИ рассмотрим задачу о выборе работы.

Со студентом, только что получившим диплом, беседовали о трех возможных местах работы (А, Б и В). Он решил использовать МАИ для осуществления выбора. В результате первого этапа применения МАИ была получена следующая иерархия.

*Уровень цели:*

1. Удовлетворение работой.

*Уровень критериев:*

2. Исследовательская работа;

3. Рост;

4. Доходы;

5. Коллеги;

6. Местонахождение;

7. Репутация.

*Уровень альтернатив:*

8. A;

9. B;

10. C.

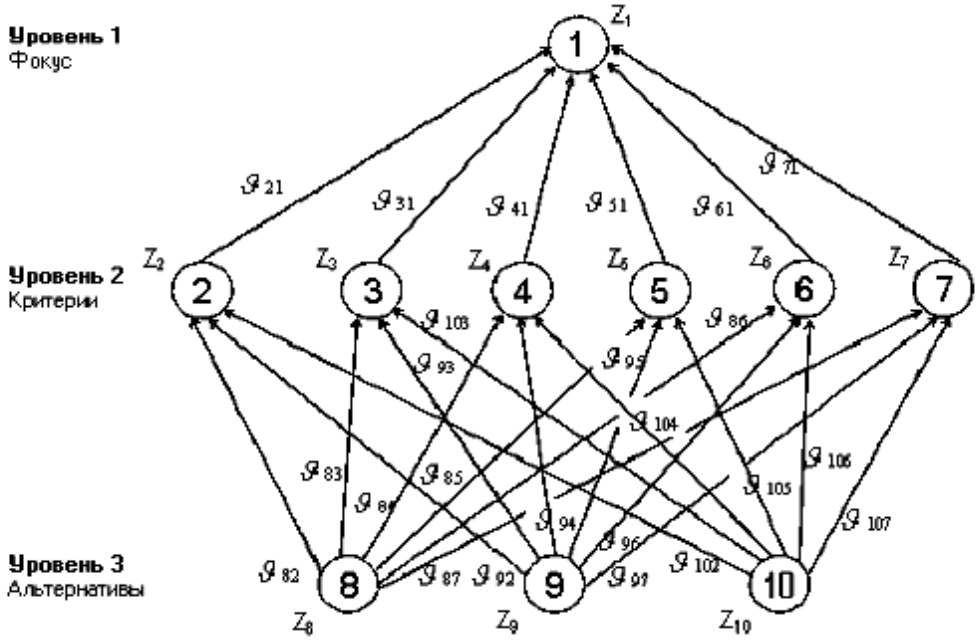



*Рис. 3.5. Иерархия проблемы выбора работы*

Выполнение второго этапа связано с заполнением нижеприведенных таблиц по методу парных сравнений с применением шкалы относительной важности. В результате обработки таблиц получаем собственные вектора, которые определяют веса соответствующих дуг.

В таблице пары критериев сравниваются с точки зрения их относительного вклада в общее понятие «удовлетворение работой». Задавался вопрос: который из заданной пары критериев представляется вносящим больший вклад в понятие «удовлетворение работой» и насколько? Например, число 5 в третьей

строке и четвертом столбце показывает, что «доходы» намного важнее, чем «общество коллег».

| Удовлетворение работой | Исследо-вание | Рост | Доходы | Коллеги | Место-нахождение | Репутация | Собственный вектор |
|---|---|---|---|---|---|---|---|
| Исследование | 1 | 1 | 1 | 4 | 1 | 1/2 | $_{21}$=0,16 |
| Рост | 1 | 1 | 2 | 4 | 1 | 1/2 | $_{31}$=0,19 |
| Доходы | 1 | 1/2 | 1 | 5 | 3 | 1/2 | $_{41}$=0,19 |
| Коллеги | 1/4 | 1/4 | 1/5 | 1 | 1/3 | 1/3 | $_{51}$=0,05 |
| Местонахождение | 1 | 1 | 1/3 | 3 | 1 | 1 | $_{61}$=0,12 |
| Репутация | 2 | 2 | 2 | 3 | 1 | 1 | $_{71}$=0,30 |

$\lambda_{\max}$=6,35; *ИС* = 0,07; *ОС* = 0,06.

В следующей таблице представлены результаты парных сравнений относительно соответствующих критериев.

| Исследование | A | B | C | Собственный вектор |
|---|---|---|---|---|
| A | 1 | 1/3 | 1/2 | $\vartheta_{82}$=0,16 |
| B | 3 | 1 | 1/7 | $\vartheta_{92}$=0,59 |
| C | 2 | 1/3 | 1 | $\vartheta_{102}$=0,25 |
| $\lambda_{max}$ = 3,05; ИС = 0,025; ОС = 0,04 | | | | |

| Рост | A | B | C | Собственный вектор |
|---|---|---|---|---|
| A | 1 | 1 | 1 | $\vartheta_{83}$=0,33 |
| B | 1 | 1 | 1 | $\vartheta_{93}$=0,33 |
| C | 1 | 1 | 1 | $\vartheta_{103}$=0,33 |
| $\lambda_{max}$ = 3,0; ИС = 0; ОС = 0 | | | | |

| Доходы | A | B | C | Собственный вектор |
|---|---|---|---|---|
| A | 1 | 5 | 1 | $\vartheta_{84}$=0,45 |
| B | 1/5 | 1 | 1/5 | $\vartheta_{94}$=0,09 |
| C | 1 | 5 | 1 | $\vartheta_{104}$=0,46 |
| $\lambda_{max}$ = 3,0; ИС = 0; ОС = 0 | | | | |

| Коллеги | A | B | C | Собственный вектор |
|---|---|---|---|---|
| A | 1 | 9 | 7 | $\vartheta_{85}$=0,77 |
| B | 1/9 | 1 | 1/5 | $\vartheta_{95}$=0,05 |
| C | 1/7 | 5 | 1 | $\vartheta_{105}$=0,17 |
| $\lambda_{max}$ = 3,21; ИС = 0,105; ОС = 0,18 | | | | |

| Место-нахождение | A | B | C | Собственный вектор |
|---|---|---|---|---|
| A | 1 | 1/2 | 1 | $\vartheta_{86}$=0,25 |
| B | 2 | 1 | 2 | $\vartheta_{96}$=0,50 |
| C | 1 | 1/2 | 1 | $\vartheta_{106}$=0,25 |
| $\lambda_{max}$ = 3,0; ИС = 0; ОС = 0 | | | | |

| Репутация | A | B | C | Собственный вектор |
|---|---|---|---|---|
| A | 1 | 6 | 4 | $\vartheta_{87}$=0,69 |
| B | 1/6 | 1 | 1/3 | $\vartheta_{97}$=0,09 |
| C | 1/4 | 3 | 1 | $\vartheta_{107}$=0,22 |
| $\lambda_{max}$ = 3,05; ИС = 0,025; ОС = 0,04 | | | | |

Результатом третьего этапа (синтеза) является определение весов $Z_i, i = \overline{1,10}$. Так как уровень 1 имеет одну цель, то $Z_1 = 1$. Отсюда

$$Z_2 = \vartheta_{21} \cdot Z_1 = 0{,}16;$$
$$Z_3 = \vartheta_{31} \cdot Z_1 = 0{,}19;$$
$$Z_4 = \vartheta_{41} \cdot Z_1 = 0{,}19;$$
$$Z_5 = \vartheta_{51} \cdot Z_1 = 0{,}05;$$
$$Z_6 = \vartheta_{61} \cdot Z_1 = 0{,}12;$$
$$Z_7 = \vartheta_{71} \cdot Z_1 = 0{,}30;$$

Вычислив веса критериев, переходим к вычислению весов альтернатив (т.е. объектов третьего уровня):

$$Z_8 = \vartheta_{82} \cdot Z_2 + \vartheta_{83} \cdot Z_3 + \vartheta_{84} \cdot Z_4 + \vartheta_{85} \cdot Z_5 + \vartheta_{86} \cdot Z_6 + \vartheta_{87} \cdot Z_7 = 0{,}16 \cdot 0{,}16 + 0{,}33 \cdot 0{,}19 + 0{,}45 \cdot 0{,}19 + 0{,}77 \cdot 0{,}05 + 0{,}25 \cdot 0{,}12 + 0{,}69 \cdot 0{,}3 = 0{,}45$$

$$Z_9 = \vartheta_{92} \cdot Z_2 + \vartheta_{93} \cdot Z_3 + \vartheta_{94} \cdot Z_4 + \vartheta_{95} \cdot Z_5 + \vartheta_{96} \cdot Z_6 + \vartheta_{97} \cdot Z_7 = 0{,}59 \cdot 0{,}16 + 0{,}33 \cdot 0{,}19 + 0{,}09 \cdot 0{,}19 + 0{,}05 \cdot 0{,}05 + 0{,}05 \cdot 0{,}12 + 0{,}09 \cdot 0{,}3 = 0{,}25$$

.....

Таким образом, в конечном счете альтернатива *A* имеет вес 0,45, *B* – 0,25 и *C* – 0,3.

Обобщением МАИ на случай, когда взаимодействием иерархических уровней (компонентов) и/или их элементов нельзя пренебречь, является мМетод аналитических сетей (МАС). Многие проблемы принятия решений нельзя представить иерархическими структурами, потому что в них существуют зависимости и взаимодействия между элементами разных уровней иерархии. Часто возникают задачи, в которых не только важность критериев влияет на приоритеты альтернатив (как в иерархиях), но также важность альтернатив влияет на приоритеты критериев. Не всегда можно говорить и об альтернативах в прямом смысле этого слова. Так, при оценке мероприятий по инновационному развитию фирмы различные направления этого развития уже не являются чисто альтернативными и могут реализовываться одновременно (так называемые «мультипроекты»), хотя и с различной интенсивностью. Взаимодействовать могут также и критерии оценки альтернатив.

Решения, полученные на сети, могут отличаться от решений, полученных даже на сложной иерархии. Далеко не всегда искусственное преодоление сложности путем сведения реальной задачи к простой структуре позволит получить результат взаимодействия между элементами проблемы в форме обобщенных суждений, правильно отражающих действительность. Поэтому

важно представлять эти суждения такими структурами, которые адекватно отражают реальность.

Главное преимущество сетей с обратными связями – возможность получения решений, которые позволяют предвидеть будущее.

# 4. ИСКУССТВЕННЫЕ НЕЙРОННЫЕ СЕТИ

Одним из наиболее перспективных современных направлений в области искусственного интеллекта является попытка смоделировать мыслительные процессы человека. При этом за основу берутся достижения не в области информатики и математического моделирования, а результаты исследования процессов функционирования нервной системы человека. В то же время знания о работе мозга столь ограничены, что при разработке искусственных нейронных сетей (ИНС) приходится выходить за пределы современных биологических знаний в поисках структур, способных выполнять полезные функции. Несмотря на то, что связь с биологией слаба, искусственные нейронные сети продолжают сравниваться с мозгом. Их функционирование часто напоминает человеческое познание, поэтому трудно избежать этой аналогии.

## 4.1. Возможности и основные положения искусственных нейронных сетей

Нервная система человека, построенная из элементов, называемых нейронами, имеет огромную сложность. Около $10^{11}$ нейронов участвуют в примерно $10^{15}$ передающих связях, имеющих длину метр и более. Каждый нейрон обладает многими качествами, общими с другими элементами тела, но его уникальной способностью являются прием, обработка и передача электрохимических сигналов по нервным путям, которые образуют коммуникационную систему мозга.

На рис. 4.1 показана структура биологического нейрона. Каждый нейрон имеет отростки нервных волокон двух типов – дендриты, по которым принимаются импульсы, и единственный аксон, по которому нейрон может передавать импульс.

Дендриты идут от тела нервной клетки к другим нейронам, где они принимают сигналы в точках соединения, называемых синапсами. Принятые синапсом входные сигналы подводятся к телу нейрона.

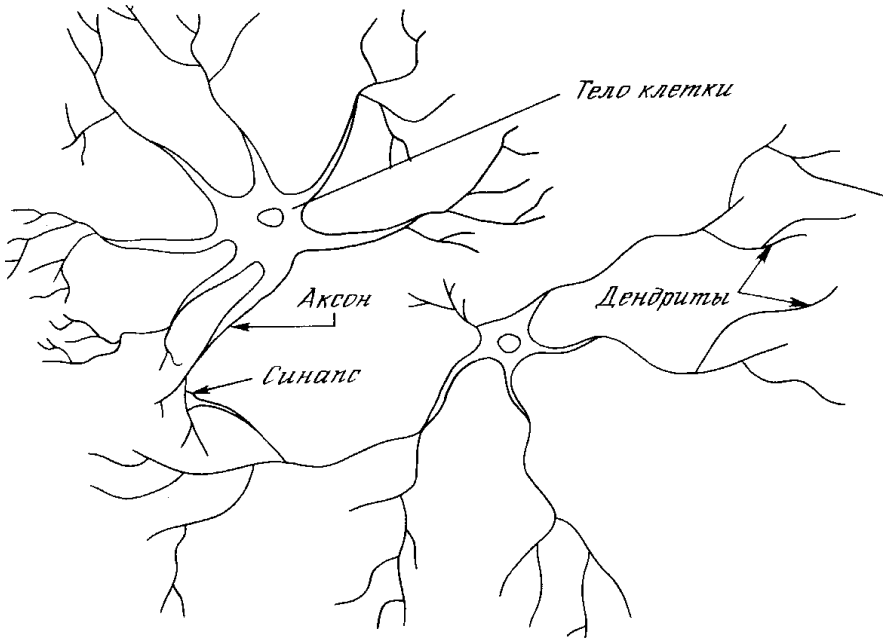



*Рис. 4.1. Биологический нейрон*

Импульсы, поступившие к нейрону одновременно по нескольким дендритам, суммируются. Если суммарный импульс превышает некоторый порог, нейрон возбуждается, формирует собственный импульс и передает его далее по аксону другим нейронам.

У этой основной функциональной схемы много усложнений и исключений, тем не менее, большинство искусственных нейронных сетей моделируют лишь эти простые свойства.

Таким образом, ИНС является приближенной электронной моделью нейронной структуры головного мозга. Известно, что принцип работы мозга отличается от вычислительной работы современных компьютеров с архитектурой фон Неймана, поскольку главным образом связан с процессами обучения на приобретаемом опыте. Поэтому применение нейронных сетей часто дает выигрыш в тех задачах, где применение традиционных подходов к их решению проблематично.

В то же время нейросети нельзя считать универсальным инструментом для решения всех вычислительных проблем. Традиционные компьютеры и вычислительные методы часто являются идеальными для многих применений. Современные цифровые вычислительные машины превосходят человека по способно-

сти производить числовые и символьные вычисления. Однако, изменив вычислительный характер задачи на восприятие внешней информации (например, узнавание человека в толпе по его лицу), можно убедиться в обратном, когда свойственное мозгу естественное мышление в виде сложных образов позволяет решить подобную задачу с высокой скоростью и абсолютной точностью.

Такие преимущества ИНС могут быть объяснены наличием у них ряда качеств, которыми не обладают компьютеры с традиционной архитектурой: распределенное представление информации и параллельные вычисления; способность к обучению и обобщению; адаптивность и самоорганизуемость; толерантность к ошибкам; низкое энергопотребление.

В свою очередь, реализация ИНС на аппаратном уровне, то есть в виде нейрокомпьютера, обладающего отмеченными выше особенностями, приведет к существенным изменениям большинства характеристик, свойственных традиционным компьютерам (табл. 4.1).

***Таблица 4.1***

| **Характеристика** | **Традиционный компьютер** | **Нейрокомпьютер** |
|---|---|---|
| Процессор | Сложный<br>Высокоскоростной<br>Один или несколько | Простой<br>Низкоскоростной<br>Большое количество |
| Память | Отдельно от процессора<br>Локальная<br>Адресация по адресу | В процессоре<br>Распределенная<br>Адресация по смыслу |
| Вычисления | Централизованные<br>Последовательные<br>Алгоритмические | Распределенные<br>Параллельные<br>Самообучение |
| Надежность | Уязвимость | Живучесть |
| Специализация | Числовые операции | Проблемы восприятия |
| Среда функционирования | Строго ограниченная | Без ограничений |
| Метод обучения | По правилам | По примерам |
| Применение | Числовая обработка информации | Распознавание, классификация, управление на основе образов |

Как видно из табл. 4.1, нейрокомпьютер в сравнении с машиной фон Неймана имеет принципиально другой способ организации вычислительного процес-

са: он не программируется с использованием явных правил и кодов в соответствии с заданным алгоритмом, а обучается.

Актуальность исследований в этом направлении подтверждается широким применением НС. Опыт использования ИНС показал их эффективность в таких задачах, как прогнозирование, распознавание, классификация, аппроксимация функций, адаптивное управление, сжатие данных и др.

В качестве научного направления ИНС впервые заявили о себе в 40-е годы 20 в. и связывают это с работами ученых Маккалока и Питтса. Стремясь воспроизвести функции человеческого мозга, исследователи создали простые программные модели биологического нейрона и системы его соединений.

Основу любой ИНС составляют отдельные элементы (ячейки),

имитирующие работу нейронов мозга – искусственные нейроны. Каждый нейрон характеризуется своим текущим состоянием по аналогии с нервными клетками головного мозга, которые могут быть возбуждены или заторможены. Он обладает группой синапсов – однонаправленных входных связей, соединенных с выходами других нейронов, а также имеет аксон – выходную связь данного нейрона, с которой сигнал (возбуждения или торможения) поступает на синапсы следующих нейронов. На рис. 4.2 представлен общий вид нейрона.

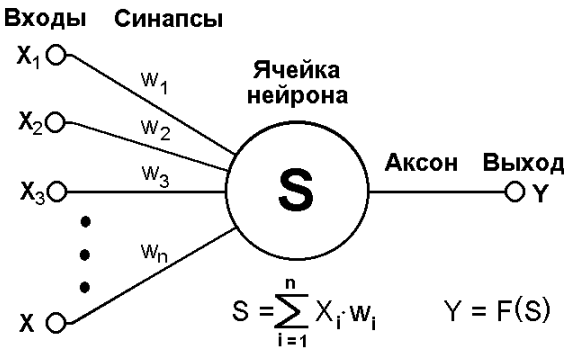



*Рис. 4.2. Обобщенная модель нейрона*

Модели НС могут быть программного и аппаратного исполнения. В дальнейшем речь пойдет в основном о первом типе.

Несмотря на существенные различия, отдельные типы НС обладают несколькими общими чертами.

Во-первых, основу каждой НС составляют относительно простые, в большинстве случаев – однотипные, элементы (ячейки), имитирующие работу нейронов мозга. Далее под нейроном будет подразумеваться искусственный нейрон, то есть ячейка НС. Каждый нейрон характеризуется своим текущим состоянием по аналогии с нервными клетками головного мозга, которые могут быть возбуждены или заторможены. Он обладает группой синапсов – однонаправленных входных связей, соединенных с выходами других нейронов, а также имеет аксон – выходную связь данного нейрона, с которой сигнал (возбуждения или торможения) поступает на синапсы следующих нейронов. Каждый синапс характеризуется величиной синаптической связи или ее весом $w_i$, который по физическому смыслу эквивалентен электрической проводимости.

Текущее состояние нейрона определяется, как взвешенная сумма его входов:

$$s = \sum_{i=1}^{n} x_i \cdot w_i \ . \qquad (4.1)$$

Выход нейрона есть функция его состояния $y = f(s)$.

Нелинейная функция $f$ называется активационной и может иметь различный вид, как показано на рисунке 4.3.

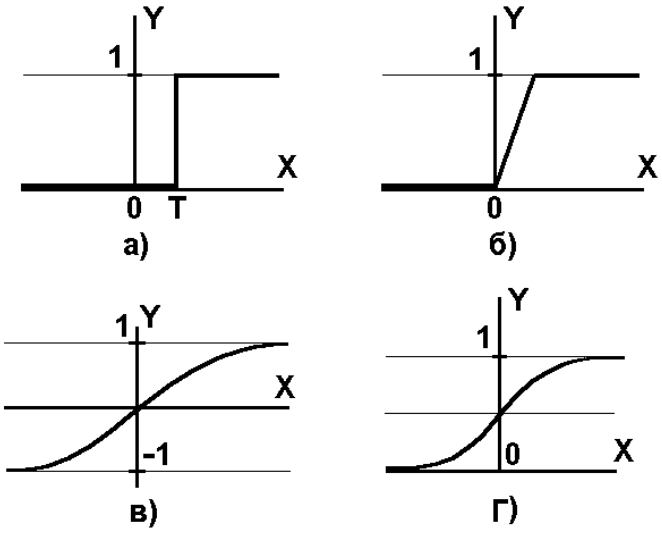



*Рис.4.3. а) функция единичного скачка; б) линейный порог (гистерезис); в) сигмоид – гиперболический тангенс; г) сигмоид – формула (4.2)*

Одной из наиболее распространенных является нелинейная функция с насыщением, так называемая логистическая функция или сигмоид (т.е. функция *S*-образного вида):

$$f(x) = \frac{1}{1 + e^{-\alpha x}}. \qquad (4.2)$$

При уменьшении $\alpha$ сигмоид становится более пологим, в пределе при $\alpha$=0 вырождаясь в горизонтальную линию на уровне 0.5, при увеличении $\alpha$ сигмоид приближается по внешнему виду к функции единичного скачка с порогом *T* в точке *x*=0. Из выражения для сигмоида очевидно, что выходное значение нейрона лежит в диапазоне [0,1]. Одно из ценных свойств сигмоидной функции – простое выражение для ее производной.

$$f'(x) = \alpha \cdot f(x) \cdot (1 - f(x)). \qquad (4.3)$$

Сигмоидная функция дифференцируема на всей оси абсцисс, что используется в некоторых алгоритмах обучения. Кроме того она обладает свойством усиливать слабые сигналы лучше, чем большие, и предотвращает насыщение от больших сигналов, так как они соответствуют областям аргументов, где сигмоид имеет пологий наклон.

Возвращаясь к общим чертам, присущим всем НС, отметим, во-вторых, принцип параллельной обработки сигналов, который достигается путем объединения большого числа нейронов в так называемые слои и соединения определенным образом нейронов различных слоев, а также, в некоторых конфигурациях, и нейронов одного слоя между собой, причем обработка взаимодействия всех нейронов ведется послойно.

В качестве примера простейшей НС рассмотрим трехнейронный персептрон (рис. 4.4), то есть такую сеть, нейроны которой имеют активационную функцию в виде единичного скачка . На n входов поступают некие сигналы, проходящие по синапсам на 3 нейрона, образующие единственный слой этой НС и выдающие три выходных сигнала:

$$y_j = f\left[\sum_{i=1}^{n} x_i \cdot w_{ij}\right], j=1...3. \qquad (4.4)$$

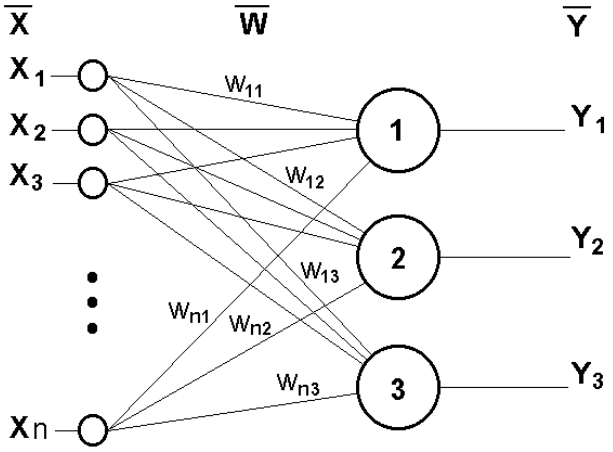



*Рис. 4.4. Однослойный перцептрон*

Очевидно, что все весовые коэффициенты синапсов одного слоя нейронов можно свести в матрицу **W**, в которой каждый элемент $w_{ij}$ задает величину i-ой синаптической связи *j*-ого нейрона. Таким образом, процесс, происходящий в НС, может быть записан в матричной форме:

$$\mathbf{Y=F(XW),} \tag{4.5}$$

где **X** и **Y** – соответственно входной и выходной сигнальные векторы,

**F(V)** – активационная функция, применяемая поэлементно к компонентам вектора **V**.

Теоретически число слоев и число нейронов в каждом слое может быть произвольным, однако фактически оно ограничено ресурсами компьютера или специализированной микросхемы, на которых обычно реализуется НС. Чем сложнее НС, тем масштабнее задачи, подвластные ей.

Выбор структуры НС осуществляется в соответствии с особенностями и сложностью задачи. Для решения некоторых отдельных типов задач уже существуют оптимальные, на сегодняшний день, конфигурации. Если же задача не может быть сведена ни к одному из известных типов, разработчику приходится решать сложную проблему синтеза новой конфигурации. При этом он руководствуется несколькими основополагающими принципами: возможности сети возрастают с увеличением числа ячеек сети, плотности связей между ними и числом выделенных слоев (влияние числа слоев на способность сети

выполнять классификацию плоских образов показано на рис. 4.5; введение обратных связей наряду с увеличением возможностей сети поднимает вопрос о динамической устойчивости сети; сложность алгоритмов функционирования сети (в том числе, например, введение нескольких типов синапсов – возбуждающих, тормозящих и др.) также способствует усилению мощи НС. Вопрос о необходимых и достаточных свойствах сети для решения того или иного рода задач представляет собой целое направление нейрокомпьютерной науки. Так как проблема синтеза НС сильно зависит от решаемой задачи, дать общие подробные рекомендации затруднительно. В большинстве случаев оптимальный вариант получается на основе интуитивного подбора.

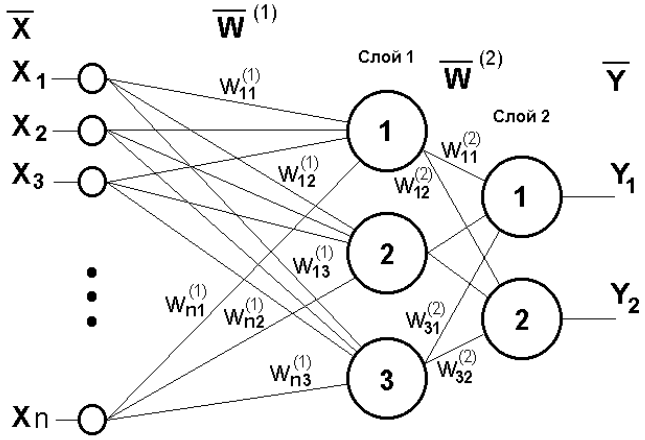



*Рис. 4.5. Двухслойный персептрон*

Очевидно, что процесс функционирования НС, то есть сущность действий, которые она способна выполнять, зависит от величин синаптических связей, поэтому, задавшись определенной структурой НС, отвечающей какой-либо задаче, разработчик сети должен найти оптимальные значения всех переменных весовых коэффициентов (некоторые синаптические связи могут быть постоянными).

Этот этап называется обучением НС, и от того, насколько качественно он будет выполнен, зависит способность сети решать поставленные перед ней проблемы во время эксплуатации. На этапе обучения кроме параметра качества подбора весов важную роль играет время обучения. Как правило, эти два параметра связаны обратной зависимостью и их приходится выбирать на основе компромисса.

Обучение НС может вестись с учителем или без него. В первом случае сети предъявляются значения как входных, так и желательных выходных сигналов, и она по некоторому внутреннему алгоритму подстраивает веса своих синаптических связей. Во втором случае выходы НС формируются самостоятельно, а веса изменяются по алгоритму, учитывающему только входные и производные от них сигналы.

Существует великое множество различных алгоритмов обучения, которые однако делятся на два больших класса: детерминистские и стохастические. В первом из них подстройка весов представляет собой жесткую последовательность действий, во втором – она производится на основе действий, подчиняющихся некоторому случайному процессу.

Развивая дальше вопрос о возможной классификации НС, важно отметить существование бинарных и аналоговых сетей. Первые из них оперируют с двоичными сигналами, и выход каждого нейрона может принимать только два значения: логический ноль («заторможенное» состояние) и логическая единица («возбужденное» состояние). К этому классу сетей относится и рассмотренный выше персептрон, так как выходы его нейронов, формируемые функцией единичного скачка, равны либо 0, либо 1. В аналоговых сетях выходные значения нейронов способны принимать непрерывные значения, что могло бы иметь место после замены активационной функции нейронов персептрона на сигмоид.

Еще одна классификация делит НС на синхронные и асинхронные. В первом случае в каждый момент времени свое состояние меняет лишь один нейрон. Во втором – состояние меняется сразу у целой группы нейронов, как правило, у всего слоя. Алгоритмически ход времени в НС задается итерационным выполнением однотипных действий над нейронами. Далее будут рассматриваться только синхронные НС.

Сети также можно классифицировать по числу слоев. На рис. 4.5 представлен двухслойный персептрон, полученный из персептрона рис. 4.4 путем добавления второго слоя, состоящего из двух нейронов. Здесь уместно отметить

важную роль нелинейности активационной функции, так как, если бы она не обладала данным свойством или не входила в алгоритм работы каждого нейрона, результат функционирования любой *p*-слойной НС с весовыми матрицами $\mathbf{W}^{(i)}$, i=1,2,...p для каждого слоя *i* сводился бы к перемножению входного вектора сигналов **X** на матрицу

$$\mathbf{W}^{(\Sigma)}=\mathbf{W}^{(1)}\cdot\mathbf{W}^{(2)}\cdot\ldots\cdot\mathbf{W}^{(p)}, \tag{4.6}$$

то есть фактически такая *p*-слойная НС эквивалентна однослойной НС с весовой матрицей единственного слоя $\mathbf{W}^{(\Sigma)}$:

$$\mathbf{Y}=\mathbf{X}\mathbf{W}^{(\Sigma)}. \tag{4.7}$$

Иногда нелинейность вводится и в синаптические связи. Большинство известных на сегодняшний день НС используют для нахождения взвешенной суммы входов нейрона формулу (4.1), однако в некоторых приложениях НС полезно ввести другую запись, например:

$$s=\sum_{i=1}^{n}x_i^2\cdot w_i \tag{4.8}$$

или даже

$$s=\sum_{i=1}^{n}x_i\cdot x_{((i+1)\bmod n)}\cdot w_i\ . \tag{4.9}$$

Введение такого рода нелинейности, вообще говоря, увеличивает вычислительную мощь сети, то есть позволяет из меньшего числа нейронов с «нелинейными» синапсами сконструировать НС, выполняющую работу обычной НС с большим числом стандартных нейронов и более сложной конфигурации.

Из рисунка функции единичного скачка видно, что пороговое значение *T*, в общем случае, может принимать произвольное значение. Более того, оно должно принимать некое произвольное, неизвестное заранее значение, которое подбирается на стадии обучения вместе с весовыми коэффициентами. То же самое относится и к центральной точке сигмоидной зависимости, которая может сдвигаться вправо или влево по оси *X*, а также и ко всем другим акти-

вационным функциям. Это, однако, не отражено в формуле (4.1), которая должна была бы выглядеть так:

$$s = \sum_{i=1}^{n} x_i \cdot w_i - T \quad . \tag{4.10}$$

Дело в том, что такое смещение обычно вводится путем добавления к слою нейронов еще одного входа, возбуждающего дополнительный синапс каждого из нейронов, значение которого всегда равняется 1. Присвоим этому входу номер 0. Тогда

$$s = \sum_{i=0}^{n} x_i \cdot w_i \quad , \tag{4.11}$$

где $w_0 = -T$, $x_0 = 1$.

Очевидно, что различие формул (4.1) и (4.11) состоит лишь в способе нумерации входов.

Из всех активационных функций, изображенных на рис. 4.3, одна выделяется особо. Это гиперболический тангенс, зависимость которого симметрична относительно оси *X* и лежит в диапазоне [-1,1].Выбор области возможных значений выходов нейронов во многом зависит от конкретного типа НС и является вопросом реализации, так как манипуляции с ней влияют на различные показатели эффективности сети, зачастую не изменяя общую логику ее работы.

Работа всех сетей сводится к классификации (обобщению) входных сигналов, принадлежащих *n*-мерному гиперпространству, по некоторому числу классов. С математической точки зрения это происходит путем разбиения гиперпространства гиперплоскостями (запись для случая однослойного персептрона)

$$\sum_{i=1}^{n} x_i \cdot w_{ik} = T_k \quad , \; k=1...m. \tag{4.12}$$

Каждая полученная область является областью определения отдельного класса. Число таких классов для одной НС персептронного типа не превышает $2^m$, где $m$ – число выходов сети. Однако не все из них могут быть разделимы данной НС.

Например, однослойный персептрон, состоящий из одного нейрона с двумя входами, представленный на рис. 4.6, не способен разделить плоскость (двумерное гиперпространство) на две полуплоскости так, чтобы осуществить классификацию входных сигналов по классам A и B.

Уравнение сети для этого случая

$$x_1 \cdot w_1 + x_2 \cdot w_2 = T \tag{4.13}$$

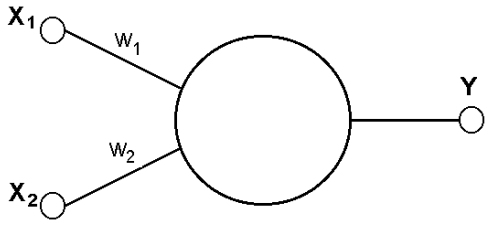



*Рис. 4.6. Однонейронный персептрон*

является уравнением прямой (одномерной гиперплоскости), которая ни при каких условиях не может разделить плоскость так, чтобы точки из множества входных сигналов, принадлежащие разным классам, оказались по разные стороны от прямой (см. рис. 4.7).

Если присмотреться к таблице, можно заметить, что данное разбиение на классы реализует логическую функцию исключающего ИЛИ для входных сигналов. Невозможность реализации однослойным перцептроном этой функции получила название проблемы исключающего ИЛИ.

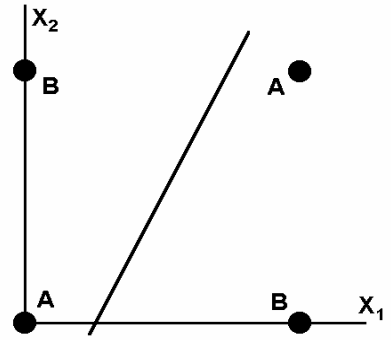



*Рис. 4.7. Визуальное представление работы НС с рис. 4.6*

Функции, которые не реализуются однослойной сетью, называются линейно неразделимыми. Решение задач, подпадающих под это ограничение, заключается в применении 2-х и более слойных сетей или сетей с нелинейными синапсами, однако и тогда существует вероятность, что корректное разделение некоторых входных сигналов на классы невозможно.

Рассмотрим алгоритм обучения с учителем.

1. Проинициализировать элементы весовой матрицы (обычно небольшими случайными значениями).

2. Подать на входы один из входных векторов, которые сеть должна научиться различать, и вычислить ее выход.

3. Если выход правильный, перейти на шаг 4.

Иначе вычислить разницу между идеальным и полученным значениями выхода: $\delta = Y_I - Y$

Модифицировать веса в соответствии с формулой:

$$w_{ij}(t+1) = w_{ij}(t) + \nu \cdot \delta \cdot x_i$$

где $t$ и $t$+1 – номера соответственно текущей и следующей итераций; $\nu$ – коэффициент скорости обучения, $0<\nu<1$; $i$ – номер входа; $j$ – номер нейрона в слое.

Очевидно, что если $Y_I > Y$ весовые коэффициенты будут увеличены и тем самым уменьшат ошибку. В противном случае они будут уменьшены, и $Y$ тоже уменьшится, приближаясь к $Y_I$.

4. Цикл с шага 2, пока сеть не перестанет ошибаться.

На втором шаге на разных итерациях поочередно в случайном порядке предъявляются все возможные входные вектора. К сожалению, нельзя заранее определить число итераций, которые потребуется выполнить, а в некоторых случаях и гарантировать полный успех.

## 4.2. Алгоритм обратного распространения

Среди различных структур нейронных сетей (НС) одной из наиболее известных является многослойная структура, в которой каждый нейрон произвольного слоя связан со всеми аксонами нейронов предыдущего слоя или, в случае первого слоя, со всеми входами НС. Такие НС называются полносвязными. Когда в сети только один слой, алгоритм ее обучения с учителем довольно очевиден, так как правильные выходные состояния нейронов единственного слоя заведомо известны, и подстройка синаптических связей идет в направлении, минимизирующем ошибку на выходе сети. По этому принципу строится, напри-

мер, алгоритм обучения однослойного персептрона. В многослойных же сетях оптимальные выходные значения нейронов всех слоев, кроме последнего, как правило, не известны, и двух или более слойный персептрон уже невозможно обучить, руководствуясь только величинами ошибок на выходах НС. Один из вариантов решения этой проблемы – разработка наборов выходных сигналов, соответствующих входным, для каждого слоя НС, что, конечно, является очень трудоемкой операцией и не всегда осуществимо. Второй вариант – динамическая подстройка весовых коэффициентов синапсов, в ходе которой выбираются, как правило, наиболее слабые связи и изменяются на малую величину в ту или иную сторону, а сохраняются только те изменения, которые повлекли уменьшение ошибки на выходе всей сети. Очевидно, что данный метод, несмотря на свою кажущуюся простоту, требует громоздких рутинных вычислений. И, наконец, третий, более приемлемый вариант – распространение сигналов ошибки от выходов НС к ее входам, в направлении, обратном прямому распространению сигналов в обычном режиме работы. Этот алгоритм обучения НС получил название процедуры обратного распространения.

Согласно методу наименьших квадратов, минимизируемой целевой функцией ошибки НС является величина

$$E(w) = \frac{1}{2}\sum_{j,p}(y_{j,p}^{(N)} - d_{j,p})^2 \ . \qquad (4.14)$$

где $y_{j,p}^{(N)}$ – реальное выходное состояние нейрона $j$ выходного слоя $N$ нейронной сети при подаче на ее входы p-го образа; $d_{jp}$ – идеальное (желаемое) выходное состояние этого нейрона.

Суммирование ведется по всем нейронам выходного слоя и по всем обрабатываемым сетью образам. Минимизация ведется методом градиентного спуска, что означает подстройку весовых коэффициентов следующим образом:

$$\Delta w_{ij}^{(n)} = -\eta \cdot \frac{\partial E}{\partial w_{ij}} \ . \qquad (4.15)$$

Здесь $w_{ij}$ – весовой коэффициент синаптической связи, соединяющей $i$-ый нейрон слоя $n$-1 с $j$-ым нейроном слоя $n$, $\eta$ – коэффициент скорости обучения, $0<\eta<1$.

Можно показать, что

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial y_j} \cdot \frac{dy_j}{ds_j} \cdot \frac{\partial s_j}{\partial w_{ij}} . \tag{4.16}$$

Здесь под $y_j$, как и раньше, подразумевается выход нейрона $j$, а под $s_j$ – взвешенная сумма его входных сигналов, то есть аргумент активационной функции. Так как множитель $dy_j/ds_j$ является производной этой функции по ее аргументу, из этого следует, что производная активационной функция должна быть определена на всей оси абсцисс. В связи с этим функция единичного скачка и прочие активационные функции с неоднородностями не подходят для рассматриваемых НС. В них применяются такие гладкие функции, как гиперболический тангенс или классический сигмоид с экспонентой. В случае гиперболического тангенса

$$\frac{dy}{ds} = 1 - s^2 . \tag{4.17}$$

Третий множитель $\partial s_j/\partial w_{ij}$, очевидно, равен выходу нейрона предыдущего слоя $y_i^{(n-1)}$.

Что касается первого множителя в (3), он легко раскладывается следующим образом:

$$\frac{\partial E}{\partial y_j} = \sum_k \frac{\partial E}{\partial y_k} \cdot \frac{dy_k}{ds_k} \cdot \frac{\partial s_k}{\partial y_j} = \sum_k \frac{\partial E}{\partial y_k} \cdot \frac{dy_k}{ds_k} \cdot w_{jk}^{(n+1)} . \tag{4.18}$$

Здесь суммирование по $k$ выполняется среди нейронов слоя $n$+1.

Введя новую переменную

$$\delta_j^{(n)} = \frac{\partial E}{\partial y_j} \cdot \frac{dy_j}{ds_j} , \tag{4.19}$$

мы получим рекурсивную формулу для расчетов величин $\delta_j^{(n)}$ слоя n из величин $\delta_k^{(n+1)}$ более старшего слоя $n$+1.

$$\delta_j^{(n)} = \left[ \sum_k \delta_k^{(n+1)} \cdot w_{jk}^{(n+1)} \right] \cdot \frac{dy_j}{ds_j} . \qquad (4.20)$$

Для выходного же слоя

$$\delta_l^{(N)} = (y_l^{(N)} - d_l) \cdot \frac{dy_l}{ds_l} . \qquad (4.21)$$

Теперь мы можем записать (4.15) в раскрытом виде:

$$\Delta w_{ij}^{(n)} = -\eta \cdot \delta_j^{(n)} \cdot y_i^{(n-1)} . \qquad (4.22)$$

Иногда для придания процессу коррекции весов некоторой инерционности, сглаживающей резкие скачки при перемещении по поверхности целевой функции, (9) дополняется значением изменения веса на предыдущей итерации

$$\Delta w_{ij}^{(n)}(t) = -\eta \cdot (\mu \cdot \Delta w_{ij}^{(n)}(t-1) + (1-\mu) \cdot \delta_j^{(n)} \cdot y_i^{(n-1)}) , \qquad (4.23)$$

где $\mu$ – коэффициент инерционности, $t$ – номер текущей итерации.

Таким образом, полный алгоритм обучения НС с помощью процедуры обратного распространения строится так:

1. Подать на входы сети один из возможных образов и в режиме обычного функционирования НС, когда сигналы распространяются от входов к выходам, рассчитать значения последних. Напомним, что

$$s_j^{(n)} = \sum_{i=0}^{M} y_i^{(n-1)} \cdot w_{ij}^{(n)} , \qquad (4.24)$$

где $M$ – число нейронов в слое $n$-1 с учетом нейрона с постоянным выходным состоянием +1, задающего смещение; $y_i^{(n-1)} = x_{ij}^{(n)}$ – $i$-ый вход нейрона $j$ слоя $n$.

$$y_j^{(n)} = f(s_j^{(n)}), \text{ где } f() \text{ – сигмоид.} \qquad (4.25)$$

$$y_q^{(0)} = I_q, . \qquad (4.26)$$

где $I_q$ – $q$-ая компонента вектора входного образа.

2. Рассчитать $\boldsymbol{\delta^{(N)}}$ для выходного слоя по формуле (4.21).

Рассчитать по формуле (4.22) или (4.23) изменения весов $\boldsymbol{\Delta w^{(N)}}$ слоя $N$.

3. Рассчитать по формулам (4.20) и (4.22) (или (4.21) и (4.23)) соответственно $\boldsymbol{\delta^{(n)}}$ и $\boldsymbol{\Delta w^{(n)}}$ для всех остальных слоев, $n$=$N$-1,...1.

4. Скорректировать все веса в НС

$$w_{ij}^{(n)}(t) = w_{ij}^{(n)}(t-1) + \Delta w_{ij}^{(n)}(t)\,. \qquad (4.27)$$

5. Если ошибка сети существенна, перейти на шаг 1. В противном случае – конец.

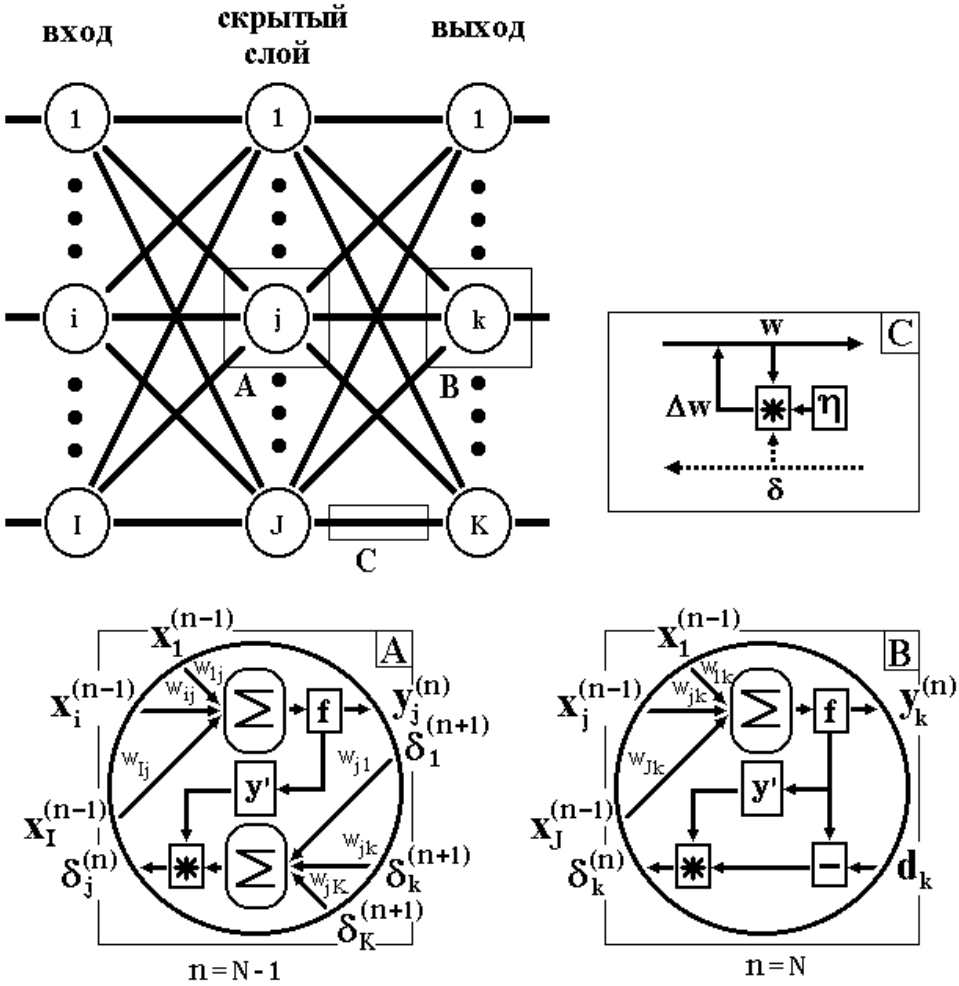



*Рис. 4.8. Диаграмма сигналов в сети при обучении по алгоритму обратного распространения*

Сети на шаге 1 попеременно в случайном порядке предъявляются все тренировочные образы, чтобы сеть, образно говоря, не забывала одни по мере запоминания других. Алгоритм иллюстрируется рисунком 4.8.

Из выражения (4.22) следует, что когда выходное значение $y_i^{(n-1)}$ стремится к нулю, эффективность обучения заметно снижается. При двоичных входных векторах в среднем половина весовых коэффициентов не будет корректироваться,

поэтому область возможных значений выходов нейронов [0,1] желательно сдвинуть в пределы [-0.5,+0.5], что достигается простыми модификациями логистических функций. Например, сигмоид с экспонентой преобразуется к виду

$$f(x) = -0.5 + \frac{1}{1 + e^{-\alpha \cdot x}}. \qquad (4.28)$$

Рассматриваемая НС имеет несколько «узких мест». Во-первых, в процессе обучения может возникнуть ситуация, когда большие положительные или отрицательные значения весовых коэффициентов сместят рабочую точку на сигмоидах многих нейронов в область насыщения. Малые величины производной от логистической функции приведут в соответствие с (7) и (8) к остановке обучения, что парализует НС. Во-вторых, применение метода градиентного спуска не гарантирует, что будет найден глобальный, а не локальный минимум целевой функции. Эта проблема связана еще с одной, а именно – с выбором величины скорости обучения. Доказательство сходимости обучения в процессе обратного распространения основано на производных, то есть приращения весов и, следовательно, скорость обучения должны быть бесконечно малыми, однако в этом случае обучение будет происходить неприемлемо медленно. С другой стороны, слишком большие коррекции весов могут привести к постоянной неустойчивости процесса обучения. Поэтому в качестве $\eta$ обычно выбирается число меньше 1, но не очень маленькое, например, 0.1, и оно, вообще говоря, может постепенно уменьшаться в процессе обучения. Кроме того, для исключения случайных попаданий в локальные минимумы иногда, после того как значения весовых коэффициентов стабилизируются, $\eta$ кратковременно сильно увеличивают, чтобы начать градиентный спуск из новой точки. Если повторение этой процедуры несколько раз приведет алгоритм в одно и то же состояние НС, можно более или менее уверенно сказать, что найден глобальный максимум, а не какой-то другой.

Существует и иной метод исключения локальных минимумов, а заодно и паралича НС, заключающийся в применении стохастических НС.

## 4.3. Нейронные сети: обучение без учителя

Рассмотренный алгоритм обучения нейронной сети с помощью процедуры обратного распространения подразумевает наличие некоего внешнего звена, предоставляющего сети кроме входных так же и целевые выходные образы. Алгоритмы, пользующиеся подобной концепцией, называются алгоритмами обучения с учителем. Для их успешного функционирования необходимо наличие экспертов, создающих на предварительном этапе для каждого входного образа эталонный выходной. Так как создание искусственного интеллекта движется по пути копирования природных прообразов, ученые не прекращают спор на тему, можно ли считать алгоритмы обучения с учителем натуральными или же они полностью искусственны. Например, обучение человеческого мозга, на первый взгляд, происходит без учителя: на зрительные, слуховые, тактильные и прочие рецепторы поступает информация извне, и внутри нервной системы происходит некая самоорганизация. Однако, нельзя отрицать и того, что в жизни человека не мало учителей – и в буквальном, и в переносном смысле, – которые координируют внешние воздействия.

Главная черта, делающая обучение без учителя привлекательным, – это его «самостоятельность». Процесс обучения, как и в случае обучения с учителем, заключается в подстраивании весов синапсов. Некоторые алгоритмы, правда, изменяют и структуру сети, то есть количество нейронов и их взаимосвязи, но такие преобразования правильнее назвать более широким термином – самоорганизацией, и в рамках данной статьи они рассматриваться не будут. Очевидно, что подстройка синапсов может проводиться только на основании информации, доступной в нейроне, то есть его состояния и уже имеющихся весовых коэффициентов. Исходя из этого соображения и, что более важно, по аналогии с известными принципами самоорганизации нервных клеток, построены алгоритмы обучения Хебба.

Сигнальный метод обучения Хебба заключается в изменении весов по следующему правилу:

$$w_{ij}(t) = w_{ij}(t-1) + \alpha \cdot y_i^{(n-1)} \cdot y_j^{(n)}, \qquad (4.29)$$

где $y_i^{(n-1)}$ – выходное значение нейрона i слоя (n-1), $y_j^{(n)}$ – выходное значение нейрона $j$ слоя n; $w_{ij}(t)$ и $w_{ij}(t\text{-}1)$ – весовой коэффициент синапса, соединяющего эти нейроны, на итерациях $t$ и $t$-1 соответственно; $\alpha$ – коэффициент скорости обучения. Здесь и далее, для общности, под n подразумевается произвольный слой сети. При обучении по данному методу усиливаются связи между возбужденными нейронами.

Существует также и дифференциальный метод обучения Хебба:

$$w_{ij}(t) = w_{ij}(t-1) + \alpha \cdot \left[y_i^{(n-1)}(t) - y_i^{(n-1)}(t-1)\right] \cdot \left[y_j^{(n)}(t) - y_j^{(n)}(t-1)\right]. \qquad (4.30)$$

Здесь $y_i^{(n-1)}(t)$ и $y_i^{(n-1)}(t\text{-}1)$ – выходное значение нейрона $i$ слоя $n$-1 соответственно на итерациях $t$ и $t$-1; $y_j^{(n)}(t)$ и $y_j^{(n)}(t\text{-}1)$ – то же самое для нейрона $j$ слоя $n$. Как видно из формулы (4.30), сильнее всего обучаются синапсы, соединяющие те нейроны, выходы которых наиболее динамично изменились в сторону увеличения.

Полный алгоритм обучения с применением вышеприведенных формул будет выглядеть так:

1. На стадии инициализации всем весовым коэффициентам присваиваются небольшие случайные значения.

2. На входы сети подается входной образ, и сигналы возбуждения распространяются по всем слоям согласно принципам классических прямопоточных (feedforward) сетей, то есть для каждого нейрона рассчитывается взвешенная сумма его входов, к которой затем применяется активационная (передаточная) функция нейрона, в результате чего получается его выходное значение $y_i^{(n)}$, $i$=0...$M_i$-1, где $M_i$ – число нейронов в слое $i$; $n$=0...$N$-1, а $N$ – число слоев в сети.

3. На основании полученных выходных значений нейронов по формуле (4.29) или (4.30) производится изменение весовых коэффициентов.

4. Цикл с шага 2, пока выходные значения сети не стабилизируются с заданной точностью. Применение этого нового способа определения завершения обучения, отличного от использовавшегося для сети обратного распространения, обусловлено тем, что подстраиваемые значения синапсов фактически не ограничены.

На втором шаге цикла попеременно предъявляются все образы из входного набора.

Следует отметить, что вид откликов на каждый класс входных образов не известен заранее и будет представлять собой произвольное сочетание состояний нейронов выходного слоя, обусловленное случайным распределением весов на стадии инициализации. Вместе с тем, сеть способна обобщать схожие образы, относя их к одному классу. Тестирование обученной сети позволяет определить топологию классов в выходном слое. Для приведения откликов обученной сети к удобному представлению можно дополнить сеть одним слоем, который, например, по алгоритму обучения однослойного перцептрона необходимо заставить отображать выходные реакции сети в требуемые образы.

Другой алгоритм обучения без учителя – алгоритм Кохонена – предусматривает подстройку синапсов на основании их значений от предыдущей итерации:

$$w_{ij}(t) = w_{ij}(t-1) + \alpha \cdot \left[ y_i^{(n-1)} - w_{ij}(t-1) \right]. \qquad (4.31)$$

Из вышеприведенной формулы видно, что обучение сводится к минимизации разницы между входными сигналами нейрона, поступающими с выходов нейронов предыдущего слоя $y_i^{(n-1)}$, и весовыми коэффициентами его синапсов.

Полный алгоритм обучения имеет примерно такую же структуру, как в методах Хебба, но на шаге 3 из всего слоя выбирается нейрон, значения синапсов которого максимально походят на входной образ, и подстройка весов по формуле (3) проводится только для него. Эта, так называемая, аккредитация может сопровождаться затормаживанием всех остальных нейронов слоя и введением выбранного нейрона в насыщение. Выбор такого нейрона может осуществляться, например, расчетом скалярного произведения вектора весовых коэффициентов с вектором входных значений. Максимальное произведение дает выигравший нейрон.

Другой вариант – расчет расстояния между этими векторами в p-мерном пространстве, где p – размер векторов.

$$D_j = \sqrt{\sum_{i=0}^{p-1} (y_i^{(n-1)} - w_{ij})^2}\ , \qquad (4.32)$$

где $j$ – индекс нейрона в слое $n$, $i$ – индекс суммирования по нейронам слоя ($n$-1), $w_{ij}$ – вес синапса, соединяющего нейроны; выходы нейронов слоя ($n$-1) являются входными значениями для слоя $n$. Корень в формуле (4.32) брать не обязательно, так как важна лишь относительная оценка различных $D_j$.

В данном случае, «побеждает» нейрон с наименьшим расстоянием. Иногда слишком часто получающие аккредитацию нейроны принудительно исключаются из рассмотрения, чтобы «уравнять права» всех нейронов слоя. Простейший вариант такого алгоритма заключается в торможении только что выигравшего нейрона.

При использовании обучения по алгоритму Кохонена существует практика нормализации входных образов, а так же – на стадии инициализации – и нормализации начальных значений весовых коэффициентов.

$$x_i = x_i \,/\, \sqrt{\sum_{j=0}^{n-1} x_j^2}\ , \qquad (4.33)$$

где $x_i$ – $i$-ая компонента вектора входного образа или вектора весовых коэффициентов, а $n$ – его размерность. Это позволяет сократить длительность процесса обучения.

Инициализация весовых коэффициентов случайными значениями может привести к тому, что различные классы, которым соответствуют плотно распределенные входные образы, сольются или, наоборот, раздробятся на дополнительные подклассы в случае близких образов одного и того же класса. Для избежания такой ситуации используется метод выпуклой комбинации. Суть его сводится к тому, что входные нормализованные образы подвергаются преобразованию:

$$x_i = \alpha(t) \cdot x_i + (1 - \alpha(t)) \cdot \frac{1}{\sqrt{n}}, \qquad (4.34)$$

где $x_i$ – $i$-ая компонента входного образа, $n$ – общее число его компонент, $\alpha(t)$ – коэффициент, изменяющийся в процессе обучения от нуля до единицы,

в результате чего вначале на входы сети подаются практически одинаковые образы, а с течением времени они все больше сходятся к исходным. Весовые коэффициенты устанавливаются на шаге инициализации равными величине

$$w_o = \frac{1}{\sqrt{n}}, \tag{4.35}$$

где $n$ – размерность вектора весов для нейронов инициализируемого слоя.

На основе рассмотренного выше метода строятся нейронные сети особого типа – так называемые самоорганизующиеся структуры). Для них после выбора из слоя $n$ нейрона $j$ с минимальным расстоянием $D_j$ (4.32) обучается по формуле (4.31) не только этот нейрон, но и его соседи, расположенные в окрестности $R$. Величина $R$ на первых итерациях очень большая, так что обучаются все нейроны, но с течением времени она уменьшается до нуля. Таким образом, чем ближе конец обучения, тем точнее определяется группа нейронов, отвечающих каждому классу образов.

Несмотря на некоторые сложности реализации, алгоритмы обучения без учителя находят обширное и успешное применение. По алгоритму обучения без учителя функционируют и наиболее сложные из известных на сегодняшний день искусственные нейронные сети – когнитрон и неокогнитрон, – максимально приблизившиеся в своем воплощении к структуре мозга.

## 4.4. Нейронные сети Хопфилда и Хэмминга

Среди различных конфигураций искусственных нейронных сетей (НС) встречаются такие, при классификации которых по принципу обучения, строго говоря, не подходят ни обучение с учителем, ни обучение без учителя. В таких сетях весовые коэффициенты синапсов рассчитываются только однажды перед началом функционирования сети на основе информации об обрабатываемых данных, и все обучение сети сводится именно к этому расчету. С одной стороны, предъявление априорной информации можно расценивать,

как помощь учителя, но с другой – сеть фактически просто запоминает образцы до того, как на ее вход поступают реальные данные, и не может изменять свое поведение, поэтому говорить о звене обратной связи с учителем не приходится. Из сетей с подобной логикой работы наиболее известны сеть Хопфилда и сеть Хэмминга, которые обычно используются для организации ассоциативной памяти. Далее речь пойдет именно о них.

Структурная схема сети Хопфилда приведена на рис. 4.9. Она состоит из единственного слоя нейронов, число которых является одновременно числом входов и выходов сети. Каждый нейрон связан синапсами со всеми остальными нейронами, а также имеет один входной синапс, через который осуществляется ввод сигнала. Выходные сигналы, как обычно, образуются на аксонах.

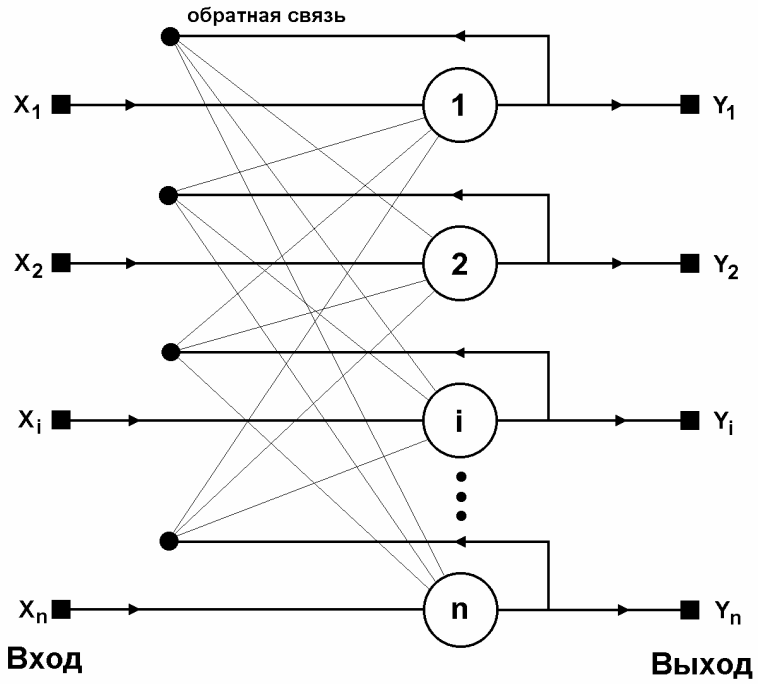



*Рис. 4.9. Структурная схема сети Хопфилда*

Задача, решаемая данной сетью в качестве ассоциативной памяти, как правило, формулируется следующим образом. Известен некоторый набор двоичных сигналов (изображений, звуковых оцифровок, прочих данных, описывающих некие объекты или характеристики процессов), которые считаются образцовыми. Сеть

должна уметь из произвольного неидеального сигнала, поданного на ее вход, выделить («вспомнить» по частичной информации) соответствующий образец (если такой есть) или «дать заключение» о том, что входные данные не соответствуют ни одному из образцов. В общем случае, любой сигнал может быть описан вектором $\mathbf{X} = \{ x_i: i=0...n-1\}$, $n$ – число нейронов в сети и размерность входных и выходных векторов. Каждый элемент $x_i$ равен либо +1, либо -1. Обозначим вектор, описывающий $k$-ый образец, через $\mathbf{X}^k$, а его компоненты, соответственно, $-x_i^k$, $k=0...m-1$, $m$ – число образцов. Когда сеть распознает (или «вспомнит») какой-либо образец на основе предъявленных ей данных, ее выходы будут содержать именно его, то есть $\mathbf{Y} = \mathbf{X}^k$, где $\mathbf{Y}$ – вектор выходных значений сети: $\mathbf{Y} = \{ y_i: i=0,...n-1\}$. В противном случае, выходной вектор не совпадет ни с одним образцовым.

Если, например, сигналы представляют собой некие изображения, то, отобразив в графическом виде данные с выхода сети, можно будет увидеть картинку, полностью совпадающую с одной из образцовых (в случае успеха) или же «вольную импровизацию» сети (в случае неудачи).

На стадии инициализации сети весовые коэффициенты синапсов устанавливаются следующим образом:

$$w_{ij} = \begin{cases} \sum_{k=0}^{m-1} x_i^k x_j^k, i \neq j \\ 0, \qquad i = j \end{cases}. \tag{4.36}$$

Здесь $i$ и $j$ – индексы, соответственно, предсинаптического и постсинаптического нейронов; $x_i^k$, $x_j^k$ – $i$-ый и $j$-ый элементы вектора $k$-ого образца.

Алгоритм функционирования сети следующий ($p$ – номер итерации):

1. На входы сети подается неизвестный сигнал. Фактически его ввод осуществляется непосредственной установкой значений аксонов:

$$s_j(p+1) = \sum_{i=0}^{n-1} w_{ij} y_i(p), \; y_i(0) = x_i, \; i = 0...n-1,, \tag{4.37}$$

поэтому обозначение на схеме сети входных синапсов в явном виде носит чисто условный характер. Ноль в скобке справа от $y_i$ означает нулевую итерацию в цикле работы сети.

2. Рассчитывается новое состояние нейронов

$$s_j(p+1) = \sum_{i=0}^{n-1} w_{ij} y_i(p)\,,\ j=0...n\text{-}1 \qquad (4.37)$$

и новые значения аксонов

$$y_j(p+1) = f\left[s_j(p+1)\right] \qquad (4.37)$$

где $f$ – активационная функция в виде скачка, приведенная на рис. 4.10,а.

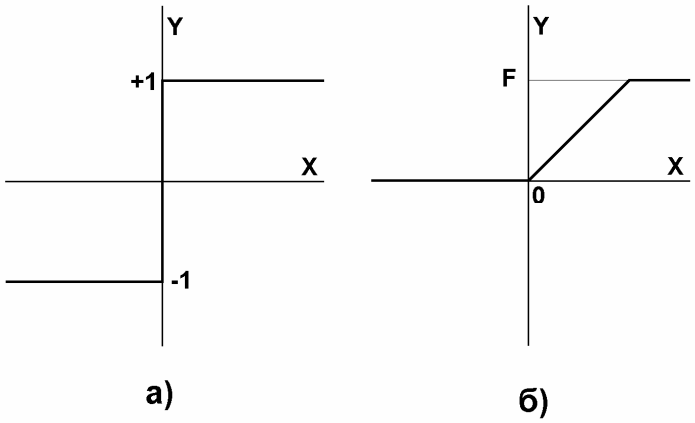



*Рис. 4.10. Активационные функции*

3. Проверка, изменились ли выходные значения аксонов за последнюю итерацию. Если да – переход к пункту 2, иначе (если выходы стабилизировались) – конец. При этом выходной вектор представляет собой образец, наилучшим образом сочетающийся с входными данными.

Как говорилось выше, иногда сеть не может провести распознавание и выдает на выходе несуществующий образ. Это связано с проблемой ограниченности возможностей сети. Для сети Хопфилда число запоминаемых образов m не должно превышать величины, примерно равной 0,15$n$. Кроме того, если два образа *А* и *Б* сильно похожи, они, возможно, будут вызывать у сети перекрестные ассоциации, то есть предъявление на входы сети вектора *А* приведет к появлению на ее выходах вектора *Б* и наоборот.

Когда нет необходимости, чтобы сеть в явном виде выдавала образец, то есть достаточно, скажем, получать номер образца, ассоциативную память ус-

пешно реализует сеть Хэмминга. Данная сеть характеризуется, по сравнению с сетью Хопфилда, меньшими затратами на память и объемом вычислений, что становится очевидным из ее структуры (рис. 4.12).

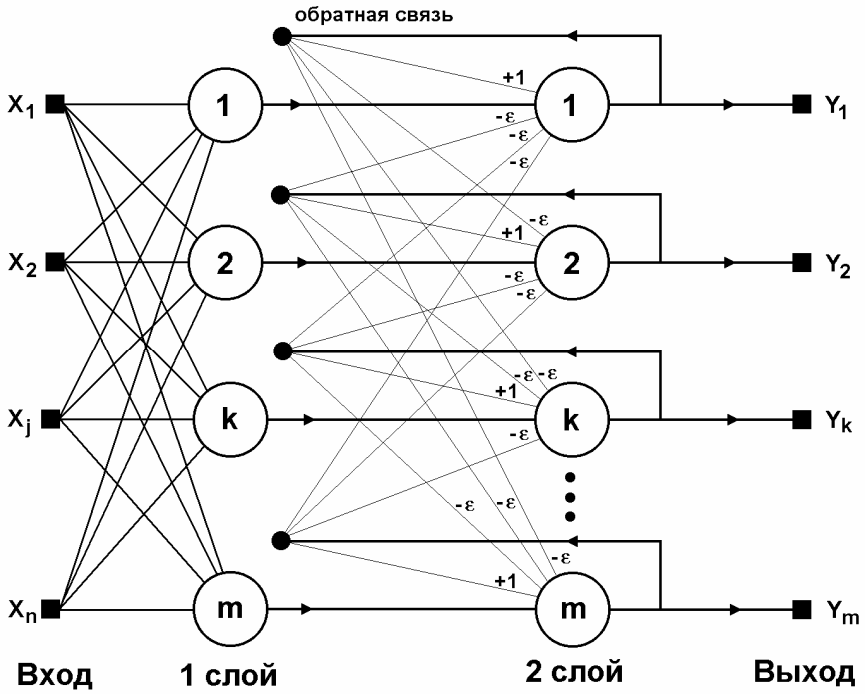



*Рис. 4.11. Структурная схема сети Хэмминга*

Сеть состоит из двух слоев. Первый и второй слои имеют по *m* нейронов, где *m* – число образцов. Нейроны первого слоя имеют по n синапсов, соединенных со входами сети (образующими фиктивный нулевой слой). Нейроны второго слоя связаны между собой ингибиторными (отрицательными обратными) синаптическими связями. Единственный синапс с положительной обратной связью для каждого нейрона соединен с его же аксоном.

Идея работы сети состоит в нахождении расстояния Хэмминга от тестируемого образа до всех образцов. Расстоянием Хэмминга называется число отличающихся битов в двух бинарных векторах. Сеть должна выбрать образец с минимальным расстоянием Хэмминга до неизвестного входного сигнала, в результате чего будет активизирован только один выход сети, соответствующий этому образцу.

На стадии инициализации весовым коэффициентам первого слоя и порогу активационной функции присваиваются следующие значения:

$$w_{ik} = \frac{x_i^k}{2},\ i=0...n-1,\ k=0...m-1, \tag{4.38}$$

$$T_k = n / 2,\ k = 0...m-1 \tag{4.39}$$

Здесь $x_i^k$ – $i$-ый элемент $k$-ого образца.

Весовые коэффициенты тормозящих синапсов во втором слое берут равными некоторой величине $0 < \varepsilon < 1/m$. Синапс нейрона, связанный с его же аксоном имеет вес +1.

Алгоритм функционирования сети Хэмминга следующий.

1. На входы сети подается неизвестный вектор $\mathbf{X} = \{x_i : i=0...n-1\}$, исходя из которого рассчитываются состояния нейронов первого слоя (верхний индекс в скобках указывает номер слоя):

$$y_j^{(1)} = s_j^{(1)} = \sum_{i=0}^{n-1} w_{ij} x_i + T_j,\ j=0...m-1 \tag{4.40}$$

После этого полученными значениями инициализируются значения аксонов второго слоя:

$$y_j^{(2)} = y_j^{(1)},\ j = 0...m-1. \tag{4.41}$$

2. Вычислить новые состояния нейронов второго слоя:

$$\boxed{\phantom{x}} \tag{4.42}$$

и значения их аксонов:

$$y_j^{(2)}(p+1) = f\left[s_j^{(2)}(p+1)\right],\ j = 0...m-1. \tag{4.43}$$

Активационная функция $f$ имеет вид порога (рис. 4.10,б), причем величина $F$ должна быть достаточно большой, чтобы любые возможные значения аргумента не приводили к насыщению.

3. Проверить, изменились ли выходы нейронов второго слоя за последнюю итерацию. Если да – перейди к шагу 2. Иначе – конец.

Из оценки алгоритма видно, что роль первого слоя весьма условна: воспользовавшись один раз на шаге 1 значениями его весовых коэффициентов,

сеть больше не обращается к нему, поэтому первый слой может быть вообще исключен из сети (заменен на матрицу весовых коэффициентов).

Обсуждение сетей, реализующих ассоциативную память, было бы неполным без хотя бы краткого упоминания о двунаправленной ассоциативной памяти (ДАП). Она является логичным развитием парадигмы сети Хопфилда, к которой для этого достаточно добавить второй слой. Структура ДАП представлена на рис. 4.12.

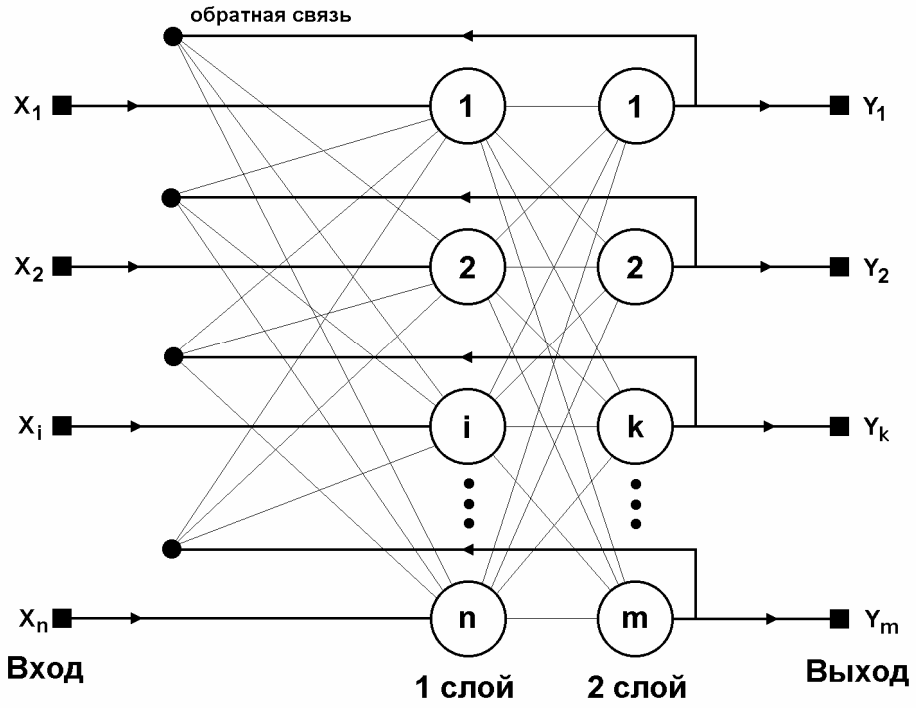



*Рис. 4.12. Двунаправленная ассоциативная память*

Сеть способна запоминать пары ассоциированных друг с другом образов. Пусть пары образов записываются в виде векторов $\mathbf{X^k} = \{x_i^k : i=0...n\text{-}1\}$ и $\mathbf{Y^k} = \{y_j^k : j=0...m\text{-}1\}$, $k=0...r\text{-}1$, где $r$ – число пар. Подача на вход первого слоя некоторого вектора $\mathbf{P} = \{p_i : i=0...n\text{-}1\}$ вызывает образование на входе второго слоя некоего другого вектора $\mathbf{Q} = \{q_j : j=0...m\text{-}1\}$, который затем снова поступает на вход первого слоя. При каждом таком цикле вектора на выходах обоих слоев приближаются к паре образцовых векторов, первый из которых – **X** – наиболее походит на **P**, который был подан на вход сети в самом начале, а второй –

**Y** – ассоциирован с ним. Ассоциации между векторами кодируются в весовой матрице $\mathbf{W^{(1)}}$ первого слоя. Весовая матрица второго слоя $\mathbf{W^{(2)}}$ равна транспонированной первой $(\mathbf{W^{(1)}})^{\mathrm{T}}$. Процесс обучения, также как и в случае сети Хопфилда, заключается в предварительном расчете элементов матрицы **W** (и соответственно $\mathbf{W^T}$) по формуле:

$$w_{ij} = \sum_k x_i y_j,\ i = 0...n-1,\ j = 0...m-1. \qquad (4.44)$$

Эта формула является развернутой записью матричного уравнения $\mathbf{W} = \sum_k \mathbf{X^T Y}$ для частного случая, когда образы записаны в виде векторов, при этом произведение двух матриц размером соответственно [$n$*1] и [1*$m$] приводит к (4.44).

В заключении можно сделать следующее обобщение. Сети Хопфилда, Хэмминга и ДАП позволяют просто и эффективно разрешить задачу воссоздания образов по неполной и искаженной информации. Невысокая емкость сетей (число запоминаемых образов) объясняется тем, что, сети не просто запоминают образы, а позволяют проводить их обощение, например, с помощью сети Хэмминга возможна классификация по критерию максимального правдоподобия. Вместе с тем, легкость построения программных и аппаратных моделей делают эти сети привлекательными для многих применений.

## 4.5. Пример использования правила обратного распространения

Для обучения обычно используется НС с функциями активации сигмоидного типа. Целью обучения по правилу обратного распространения является минимизация ошибки обучения, которая определяется как

$$E = \frac{1}{2}\sum_{i=1}^{N_{out}} \left(Y_i - Y_{жел.i}\right)^2 .$$

Для уменьшения ошибки веса изменяются по правилу

$$w_{ij} = w_{ij} - \nu \frac{\partial E}{\partial w_{ij}},$$

где ν – константа, характеризующая скорость обучения. Данная формула описывает процесс градиентного спуска в пространстве весов.

Алгоритм обратного распространением состоит из следующих шагов.

**Шаг 1.** На вход НС подается вектор **X** из обучающей выборки и вычисляются выходы всех нейронов $\mathbf{Y}_{ij}$.

**Шаг 2.** Определяется величина градиента ошибки *EI* для каждого нейрона выходного слоя:

$$EI_{Nj} = (Y_j - Y_{жел.j}) \cdot Y_j \cdot (1 - Y_j),$$

где $Y_j$ – выход *j*-го нейрона выходного слоя.

**Шаг 3.** Двигаясь от последнего слоя к первому определяются градиенты $EI_{ij}$ для каждого *j*-го нейрона каждого *i*-го слоя:

$$EI_{ij} = Y_{ij} \cdot (1 - Y_{ij}) \cdot \sum_{k=1}^{n(i+1)} EI_{i+1,k} \cdot w_{i+1,k}^{j},$$

где *k* – номер синапса, соединяющего нейрон $H_{ij}$ с нейроном $H_{i+1,k}$ следующего слоя.

**Шаг 4.** Коррекция весов синапсов:

$$w_{ij}^{k} = w_{ij}^{k} - \nu \cdot EI_{ij} \cdot Y_{i-1,k}.$$

Коррекция весов для входного слоя не производится.

**Шаг 5.** Если обучающая выборка не закончилась, то шаги 1 – 5 повторяются.

**Шаг 6.** Определяется величина ошибки E. Если она не удовлетворительна, то шаги 1 – 6 повторяются.

Из описанного алгоритма видно, что процесс обучения НС включает два вложенных цикла обучения: внутренний цикл (шаги 1 – 5) повторяется соответственно количеству примеров из обучающей выборки, внешний (шаги 1 – 6) – до тех пор, пока не будет достигнуто удовлетворительное (с точки зрения ошибки *E*) качество обучения.

После успешного обучения НС может быть протестирована на тестовой выборке. Если ошибка обучения на каждом примере из тестовой выборки

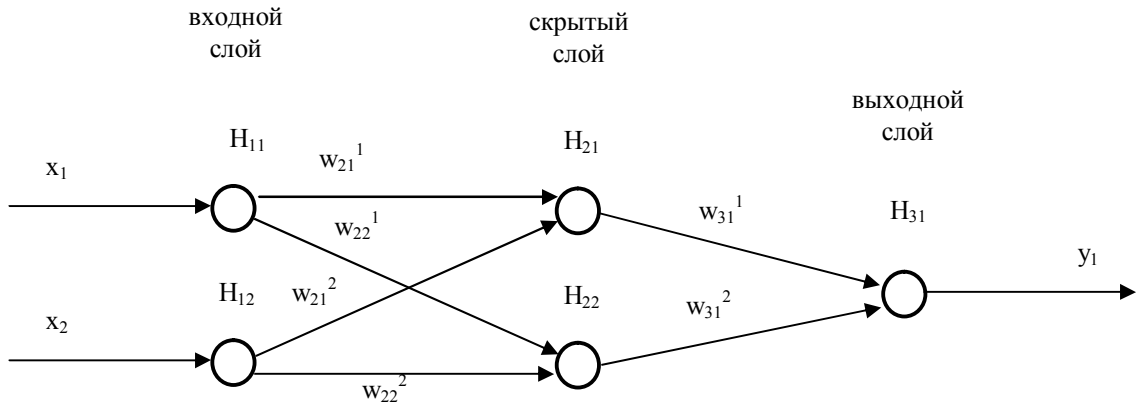
удовлетворительна, то НС можно считать обученной и приступать к ее эксплуатации.

В качестве примера возьмем обучающую выборку:

| $x_1$ | $x_2$ | y |
|---|---|---|
| 1 | 0 | 2 |
| 2 | 1 | 6 |
| 4 | 2 | 16 |

Здесь $x_1$ и $x_2$ – входные параметры НС, y – желаемый выходной параметр.

Поскольку у аппроксимируемой функции два входных параметра и один выходной, то выбирается НС с двумя нейронами во входном слое и одним в выходном. Количество нейронов скрытого слоя примем равным двум. То есть формируется сеть вида 2-2-1.




В качестве функции активации выбирается сигмоидная функция с коэффициентом $\alpha = 1$. Начальные значения весов синаптических связей принимаются равными 0,5:

$$w_{21}^1 = w_{22}^1 = w_{21}^2 = w_{22}^2 = w_{31}^1 = w_{31}^2 = 0{,}5.$$

Поскольку исходные значения $x_1$ , $x_2$ и y не лежат в пределах [0, 1], их необходимо пронормировать, поделив, например, $x_1$ на 4, $x_2$ на 2, а y на 16. В результате получена нормированная выборка:

| $x_1$ | $x_2$ | y |
|---|---|---|
| 0,25 | 0 | 0,125 |
| 0,5 | 0,5 | 0,375 |
| 1 | 1 | 1 |

Скорость обучения принимается равной $\nu = 0,2$.

После подготовки можно приступать к обучению.

**Шаг 1.** На входы НС подается первый вектор входных параметров: $x_1 = 0,25$ и $x_2 = 0$. При этом $y_{жел} = 0,125$.

Выходы нейронов входного слоя: $Y_{11} = 0,25$, $Y_{12} = 0$.

Для скрытого слоя:

$U_{21} = w_{21}^1 * Y_{11} + w_{21}^2 * Y_{12} = 0,5*0,25 + 0,5*0 = 0,125;$

$U_{22} = w_{22}^1 * Y_{11} + w_{22}^2 * Y_{12} = 0,5*0,25 + 0,5*0 = 0,125;$

$Y_{21} = 1 / (1 + \exp(-\alpha * U_{21})) = 1 / (1 + \exp(-0,125)) = 0,5312;$

$Y_{22} = 1 / (1 + \exp(-\alpha * U_{22})) = 1 / (1 + \exp(-0,125)) = 0,5312.$

Для выходного слоя:

$U_{31} = w_{31}^1 * Y_{21} + w_{31}^2 * Y_{22} = 0,5*0б5312 + 0,5*0б5312 = 0,5312;$

$Y_{31} = 1 / (1 + \exp(-\alpha * U_{31})) = 1 / (1 + \exp(-0,5312)) = 0,6298.$

**Шаг 2.** Величина градиента для выходного нейрона:

$EI_{31} = (Y_{31} - y_{жел}) * Y_{31} * (1 - Y_{31}) = (0,6298 - 0,125) * 0,6298 * (1 - 0,6298) = 0,1177.$

**Шаг 3**. Величины градиентов для скрытого слоя:

$EI_{21} = Y_{21} * (1 - Y_{21}) * [EI_{31} * w_{31}^1] = 0,5312 * (1 - 0,5312) * 0,1177 * 0,5 = 0,01466,$

$EI_{22} = Y_{22} * (1 - Y_{22}) * [EI_{31} * w_{31}^2] = 0,5312 * (1 - 0,5312) * 0,1177 * 0,5 = 0,01466.$

**Шаг 4.** Коррекция весов синапсов:

$w_{21}^1 = w_{21}^1 - \nu * Y_{11} * EI_{21} = 0,5 - 0,2 * 0,25 * 0,01466 = 0,4993,$

$w_{22}^1 = w_{22}^1 - \nu * Y_{11} * EI_{22} = 0,5 - 0,2 * 0,25 * 0,01466 = 0,4993,$

$w_{21}^2 = w_{21}^2 - \nu * Y_{12} * EI_{21} = 0,5 - 0,2 * 0 * 0,01466 = 0,5,$

$w_{22}^2 = w_{22}^2 - \nu * Y_{12} * EI_{22} = 0,5 - 0,2 * 0 * 0,01466 = 0,5,$

$w_{31}^1 = w_{31}^1 - \nu * Y_{21} * EI_{31} = 0,5 - 0,2 * 0,5312 * 0,1177 = 0,4875,$

$w_{31}^2 = w_{31}^2 - \nu * Y_{22} * EI_{31} = 0,5 - 0,2 * 0,5312 * 0,1177 = 0,4875.$

Если при полученных весах на вход НС подать тот же вектор входных параметров, то на выходе будет $y = 0,6267$, что уже ближе к желаемому $y_{жел} =$

0,125. То есть данный цикл обучения приблизил ответ НС к желаемому на величину $\Delta y = 0{,}6298 - 0{,}6267 = 0{,}0031$.

Поскольку обучающая выборка не закончилась, то шаги 1 – 4 повторяются аналогично для следующего вектора входных параметров. ♦

# 5. ИНСТРУМЕНТАЛЬНЫЕ СРЕДСТВА ПРОЕКТИРОВАНИЯ ИНТЕЛЛЕКТУАЛЬНЫХ СИСТЕМ

Программные средства инженерии знаний и реализации интеллектуальных информационных систем (ИИС) можно разделить на следующие группы: универсальные языки программирования (в том числе традиционные), универсальные языки представления знаний и оболочки.

Выше уже говорилось, что ИИС представляют собой некоторый программный комплекс, позволяющий решать производственный и экономические задачи на уровне человека – оператора или управленца (эксперта). Однако очевидно, что любую программу можно написать на машинно – ориентированном языке (ассемблере) или на универсальном языке высокого уровня (ПЛ/1, Си, Бейсик, Алгол, Ада, Фортран, Паскаль и т.д.). В этой связи возникает вполне справедливый вопрос: зачем рассматривать специализированные средства, для изучения которых требуется определенное время, если универсальным языком высокого уровня (либо языком ассемблера) владеет практически любой программист? Ответ на этот вопрос взят из практики: процесс программирования систем ИИ на специализированных средствах занимает в 2-3 раза меньше времени, чем на универсальных. Однако следует всегда помнить, что параметры эффективности (объем памяти и быстродействие) ИИС, реализованных на базе специализированных средств, в большинстве случаев ниже, чем при реализации ИИС на универсальных средствах.

Еще одним фактором, существенным для выбора ИИС инструментальных программных средств при разработке ИИС, является потенциальная возможность взаимодействия с программными средствами, используемыми на различных уровнях иерархии интегрированных корпоративных информационных систем.

В этой связи оптимальным решением задачи выбора программных средств для реализации ИИС следует, по – видимому, считать следующее: первый прототип (или прототипы: исследовательский, демонстрационный) реализует-

ся на специализированных средствах. В случае достаточной эффективности этих средств на них могут быть написаны действующий прототип, и даже промышленная система. Однако в большинстве случаев прототип и даже промышленная система. Однако в большинстве случаев прототип следует «переписать» на традиционных программных средствах.

Рассмотрим наиболее известные и широко применяемые программные средства интеллектуальных систем.

## 5.1. Анализ традиционных языков программирования и представления знаний

### *5.1.1. Специализированный язык LISP*

Одним из самых популярных языков программирования в системах ИИ является язык LISP. Этот язык был создан в 60-х годах американским ученым Дж. Маккарти и его учениками. На сегодняшний день существует около 20 диалектов этого языка. Наиболее известными являются INTERLISP, FRANZLISP, QLISP, COMMONLISP. В Советском Союзе разработаны также несколько версий языка LISP. На языке LISP написаны многие экспертные системы (MYCIN, INTERNIST, KEE и др.), системы естественно – языкового общения (MARGIE, SHRDLU, ДИЛОС и др.), интеллектуальные операционные системы (FLEX).

Популярность языка LISP в первую очередь объясняется тем, что он с помощью довольно простых конструкций позволяет писать сложные и изящные системы обработки символьной информации. К сожалению, почти все существующие LISP – системы имеют низкую вычислительную эффективность. Именно это не дает возможность языку LISP выйти за рамки «академических» экспериментальных систем. Однако бурное повышение производительности современных компьютеров, а также разработка LISP – машин типа C/330, SYMBOLICS и т.д. вселяет оптимизм в отношении будущего языка.

Язык LISP имеет очень простой синтаксис, поскольку возможны только две его конструкции: атом и список.

Атом – элементарная конструкция языка LISP, характеризуемая своим именем и значением. В некоторых LISP – системах с атомом связывается также определенный список свойств. Примерами атомов могут служить: A, B, A1, ВРЕМЯ ВЫЛЕТА, ВЫПУСК, АИ-93 и т.д.

Список – конструкция LISP, состоящая из множества атомов и подсписков. В LISP принята скобочная нотация описания списков. Примеры списков (A1, A2,…,AK), (A,B) (ВРЕМЯ ВЫЛЕТА, 15_40) (ВЫПУСК АИ-93).

Существенной особенностью языка LISP является то, что здесь «данные» и «программы» внешне ничем не отличаются друг от друга. Это дает возможность писать на LISP «программы», манипулирующие не только данными, но и «программами». Именно данное свойство позволяет LISP стать изящным средством программирования систем ИИ. Понятия «данные», и «программа» в LISP не используются, их заменяют такие понятия, как выражение и функция.

LISP – функциональный язык. Все процедуры обработки информации оформляются в виде функций. Благодаря стандартному набору системных функций, LISP может быть «расширен» за счет пользовательских функций. Системные функции делятся на арифметические, списковые функции, функции ввода – вывода, предикаты и др.

LISP – это рекурсивный язык, т.е. обеспечивает возможность определения функций с помощью самих себя.

Рекурсивность LISP удобна при решении очень популярной в искусственном интеллекте задачи «поиска по дереву», которая является довольно обобщенной и охватывает широкий класс конкретных задач, начиная шахматными и кончая задачами «принятия решений» или управления сложными объектами. В каждом конкретном случае вершины дуги будут иметь свою семантику. Например, при решении шахматной задачи вершинам могут соответствовать возможные позиции, а дугам те или иные ходы, приводящие к этим позициям. Последовательность ходов, обязательно приводящих к «выигранной» позиции, и будет решением данной задачи. Древовидная структура очень часто имеет также и сценарий

диалога человека с ЭВМ. В этом случае с вершинами соотносятся состояния (шаги) диалога, а с дугами – возможные переходы из одного состояния в другое.

### *5.1.2 Фрейм – ориентированный язык FRL*

Одним из известных языков представления знаний является язык FRL (Frame Representation Language), относящийся к классу фрейм – ориентированных. Основная единица знания в таких языках – фрейм, представляющий собой информационную модель (или описание) некоторой стереотипной ситуации. «Стереотипная ситуация», является обобщением таких понятий, как действия, процессы, события, объекты, свойства, модификаторы и т.д.

Фрейм в FRL – это совокупность поименованных, ассоциативных списков, содержащая до пяти уровней подструктур. Подструктурами фреймов могут быть слоты, аспекты, данные, комментарии и сообщения. Основной структурной единицей во фрейме являются слоты, отражающие взаимосвязи между понятиями предметной области. Слот характеризуется своим именем и значением. Имена слотов назначаются проектировщиками БЗ. Однако FRL имеет также и зарезервированные имена слотов: AKO, INSTANSE, CLAS SIFICATON. В качестве значения слотов могут выступать числа, символы, имена других фреймов, имена процедур.

Фреймы в FRL строятся с помощью процедуры FASSERT.

В FRL имеется семь зарезервированных аспектов: ¤VALUE, ¤DEFAULT, ¤IF-NEEDED, ¤IF-ADDED, ¤IF-REMOVED, ¤IF-INSTANTIAD, ¤REQUIRE. Данные из аспекта ¤VALUE интерпретируются как значение слота, а из аспекта ¤DEFAULT – как значение по умолчанию. Остальные пять аспектов связывают с фреймом процедуральные знания. Процедуры из аспекта ¤IF-ADDED активизируются в том случае, если в слот добавлено новое данные; из аспекта ¤IF-REMOVED – если из слота удаляется то или иное данные. Процедуры из аспекта ¤IF-NEEDED запускаются при создании экземпляров фрейма. Аспект ¤REQUIRE содержит процедуры, которые ограничивают значение слота.

Важным свойством FRL является наличие в нем встроенного механизма «наследования свойств». Суть этого механизма заключается в следующем. Все понятия предметной области в БЗ организовываются в виде иерархической классификационной системы, где каждое понятие связывается с помощью специальных отложений с более конкретными. Для реализации этих отложений существуют слоты АКО и INSTANSE. Слот АКО связывает понятие с более общим (родовым). Слот INSTANSE связывает понятие с более конкретным (видом). Свойства присущие всему классу, описывают только во фрейме класса, а остальные фреймы этого класса могут наследовать это свойство в случае надобности.

Процедуры обработки FRL подразделяются на независимые и присоединенные. Независимо от типа эти процедуры пишутся обычно на языке реализации самого FRL. На сегодняшний день большинство FRL – систем написаны на LISP.

### *5.1.3 Язык логического программирования PROLOG*

В последнее время к разработке ЭС все чаще стал привлекаться язык программирования Пролог. Свое наименование Пролог получил от сокращения «Программирование логики» (PRogramming in LOGic). Математической основой Пролога являются исчисление предикатов преимущественно первого порядка, метод резолюции Робинсона, теория рекурсивных функций.

Основной конструкцией языка (в форме, принятой для Пролога), является импликация:

$A \leftarrow B_1, B_2, \ldots, B_n$, называемая правилом, где $A_1, B_1, B_2 \ldots, B_n$ – предикаты.

Смысл ее таков «А истинно, если истинно $B_1$ и истинно $B_2$ и … и истинно $B_n$». Например;

```
РЕЖИМ_УСТАНОВКИ («РЕЖИМ_1») ТИП_НЕФТИ)
                    («ЗАПАДНО-СИБИРСКАЯ»)
                    КОЛ_НЕФТИ (X),
                    БОЛЬШЕ (X,0)
```

Смысл этого правила: «ЕСЛИ тип нефти – западно – сибирская и количество нефти больше нуля, то технологическая установка работает на первом режиме».

Из приведенной импликации следует два вырожденных случая:

1) $A\leftarrow$ – такая конструкция называется фактором и имеет смысл: «*A* истинно всегда». Например: ВХОДИТ («УСТАНОВКА – 26», «ЦЕХ № 2»), т.е. установка 26 входит в цех № 2.

2) $\leftarrow B_1, B_2, \ldots, B_n$ – *данная* конструкция носит название «вопрос» и означает: «Истинно ли $B_1 \wedge B_2 \ldots \wedge B_n$? (знак $\wedge$ – конъюнкция)». Например: ← РЕЖИМ_УСТАНОВКИ («РЕЖИМ_2»), т.е. следует проверить работает ли установка во втором режиме?

Предикаты Пролога имеют вид:

<ИМЯ предиката> (<аргумент 1>, … <аргумент *K*>)

Имя предиката можно рассматривать как наименование отношения на множестве аргументов. Аргументом являются либо константа (атом, атомическое выражение), либо переменная. Переменные используются для построения более сложных зависимостей:

ВХОДИТ (X, «ЗАВОД НПЗ1») ←ВХОДИТ (X, «ЦЕХ_№-2»)

Т.е. если некоторый объект (X) входит цех №2, то он входит в НПЗ1;

ВХОДИТ (X,У) ← ВХОДИТ (X, Z), ВХОДИТ (Z,Y).

Здесь X входит в У, если X входит в Z, а Z входит в У.

При использовании переменных «в вопросе» можно получить их значения в качестве результата. Например: РЕЖИМ_УСТАНОВКИ (X). Если режим установки известен и определен, то в качестве ответа получим логическое «да» или «истинно», а в качестве побочного эффекта получим значение переменной X, т.е. действующий режим работы установки, например X= «РЕЖИМ_2».

Неотъемлемым элементом Пролога является рекурсия. В частности, приведенное выше определение предиката ВХОДИТ является рекурсивным, т.е. имя предиката, стоящего в левой части, совпадает с именем хотя бы одного из предикатов, стоящих в правой части. С помощью рекурсии можно реализовать также циклические процессы вычислений.

Существующие системы программирования Пролога имеют большой набор «встроенных» предикатов (т.е. предикатов, понимаемых самим Прологом), которые обеспечивают выполнение арифметических операций, строковую (символьную) обработку, функции ввода-вывода и целый ряд специфических функций. За счет наличия встроенных предикатов язык Пролог можно отнести к универсальным языкам программирования и даже к языкам системного программирования.

Важнейшей особенностью языка Пролог является наличие реляционной базы данных, причем доступ и работа с реляционными отношениями погружены в сам Пролог. Для пользователя эти отношения существуют лишь в виде предикатов. Отмеченное свойство делает Пролог очень удобным средством для описания организационных и технологических структур. Так, на Прологе эффективно реализуются задачи подсистемы «Кадры», выдающие всевозможную информацию о кадровой структуре подразделений и о предприятии в целом. Удобно использовать Пролог и для описания технологической схемы производства со множеством взаимосвязей отдельных узлов (установок) по материальным, энергетическим, информационным и другим потокам.

Здесь не преследовалась цель дать полное описание языка Пролог, поэтому многие элементы языка (такие, как сопоставление, backtracking и др.) нами не рассматривались.

В настоящее время создано большое число различных по эффективности и мощности Пролог – систем, каждая из которых предлагает свой синтаксис языка и свой набор встроенных предикатов. Синтаксис языка определен формой записи Пролог – конструкций: фактов, правил, вопросов, предикатов, атомов, переменных, выражений и т.д.

### *5.1.4. Продукционный язык OPS*

Язык относится к числу продукционных. Являясь универсальным языком программирования, он в первую очередь предназначен для разработки систем

ИИ, и, в частности экспертных систем. Идеология языка OPS нашла отражение в целом ряде практических реализаций, достаточно сильно отличающихся друг от друга. Одной из первых и наиболее известной является реализация OPS-5, выполненная на одной из версий Лиспа (Franz LISP). Поэтому синтаксис OPS –5 максимально приближен к синтаксису Лиспа. На языке OPS – 5 создан ряд промышленно эксплуатируемых экспертных систем для фирмы DEC с объемом баз знаний от 1000 до 5000 правил. Одной из последних, но уже достаточно широко известной реализацией является OPS-83. Особенности этой реализации – наличие некоторых конструкций, характерных для процедурных языков программирования, а также сильная типизация данных.

Говоря об общих отличительных чертах семейства языков OPS, необходимо отметить наличие:

программного управления стратегий вывода решений;

развитой структуры данных и принципиальной эффективности реализации.

Язык OPS имеет типичную для продукционных систем архитектуру, включающую в себя базу правил, рабочую память и механизм вывода. База правил состоит из неупорядоченной совокупности правил, рабочая память – из дискретных объектов, называемых элементами рабочей памяти. Элемент рабочей памяти может быть добавлен в рабочую память, удален из нее или модифицирован. Механизм вывода является стандартным для системы продукций циклом управления. На первой фазе цикла выбираются все правила, левые части которых сопоставились с содержимым рабочей памяти. На второй фазе правило выполняется. Встроенный в OPS механизм вывода непосредственно поддерживает только прямой вывод, однако в языке имеются средства для организации обратного и смешанного выводов.

В языке OPS допускается использование внешних процедур, реализованных на других языках программирования. Эффективность в языке OPS достигается, во–первых, за счет использования специального алгоритма быстрого сопоставления, а во-вторых, за счет компилирующей схемы, применяемой

взамен более традиционной для продукционных языков интерпретирующей. Применение алгоритма быстрого сопоставления накладывает ряд ограничений, однако опыт эксплуатации языка OPS показал достаточно высокую адекватность средств языка для разработки экспертных систем.

Программа на языке OPS состоит из декларативной и продукционной частей. Вообще говоря, язык OPS очень прост: в нем всего три вида операторов: оператор описания типов данных TYPE, оператор описания классов CLASS и оператор описания правил RULE. Декларативная часть программы содержит описание типов данных и классов элементов рабочей памяти. Элемент рабочей памяти (класс) является единственно возможным представлением данных в OPS программе. Он представляет собой фиксированную структуру, состоящую из совокупности пар «атрибут – значение» вида:

Класс

Атрибут – 1

значение

Атрибут –2

значение

Например, элемент памяти «Установка 1 предприятия» имеет вид:

ОБЪЕКТ

Имя

Установка 1

Входит в

Цех 1

Состоит из

(колонна 1, колонна 2, колонна 3)

Получает сырье от

Резервуар 5

Тип сырья

Нефть

Вырабатывает

Керосин

Поставляет продукцию (установка 3, установка 4).

Элементарным объектом данных, имеющим значение является атрибут. Атрибуты локализованы в пределах одного класса, некоторые атрибуты могут не иметь значения. Каждый элемент рабочей памяти относится к определенному классу. Допустимые классы элементов рабочей памяти должны быть предварительно описаны в разделе определения классов. Определение класса задает его структуру, типы допустимых значений атрибутов, входящих в класс, и, если это необходимо, начальные значения атрибутов (значения по умолчанию).

Отметим, что все используемые типы данных (кроме встроенных) должны быть явно описаны в операторах TYPE или CLASS; особенно удобными при программировании экспертных систем являются произвольные типы данных, которые позволяют следить за правильностью вводимых пользователем данных и допустимых значений в процессе логического вывода.

Перейдем теперь к рассмотрению продукционной части OPS программы – раздела правил. Он записывается после декларативной части и представляет собой совокупность правил. Правило OPS состоит из заголовка и тела правила. Заголовок правила начинается со слова RULE, за которым следует имя правила и описание переменных (если они используются). Тело правила состоит из левой части, задающей условие применимости правила, и правой части, содержащей последовательность выполняемых действий. Левая часть начинается словами IF, разделителем между левой и правой частями служит слово THEN. Ниже приведен пример правила OPS:

RULE ОСТАНОВКА УСТАНОВКИ 1

IF

**ОБЪЕКТ**

ИМЯ УСТАНОВКА_1

СОСТОЯНИЕ ОСТАНОВИЛАСЬ

AND

ОБЪЕКТ

ИМЯ УСТАНОВКА_2

СОСТОЯНИЕ РАБОТАЕТ

AND

ОБЪЕКТ

ИМЯ УСТАНОВКА_3

СОСТОЯНИЕ РАБОТАЕТ

THEN

WRITE ∇ Установка 1 остановилась∇

WRITE ∇ Распределить ее сырье между установкой 2 и установкой 3 ∇

Левая часть правила состоит из одного или более условных элементов. Каждый условный элемент задает образец, который сопоставляется с содержащимися в рабочей памяти элементами. Левая часть считается сопоставленной, если одновременно сопоставились все входящие в нее условные элементы. Разделителем между условными элементами служит слово AND или ANDNUT. В левой части могут быть использованы следующие элементарные предикатное операторы: «не равно», «больше», «меньше», «не больше», «не меньше», «входит», «не входит в список», «имеет/не имеет значение» и др. Правая часть правила состоит из последовательности императивных утверждений, называемых действиями. Действия, имеющиеся в OPS, разделяются на две группы: элементарные, обеспечивающие вывод решения, и вспомогательные, обеспечивающие ввод и другие сервисные возможности. К элементарным действиям относятся:

MARE – создание нового элемента рабочей памяти;

REMOVE – удаление элемента из рабочей памяти;

MODIFY – изменение значений атрибутов, уже находящихся в рабочей памяти

К вспомогательным действиям относятся:

HALT – явное прекращение программы;

WRITE – выдача сообщения на терминал;

PRINT – печать сообщения на печатающем устройстве;

DISPLAY – вывод на экран терминала информации из библиотеки на магнитном диске;

BIND – вызов функций или модулей на других языках программирования;

SET – динамическое изменение стратегии вывода решений или подробности объяснений решений.

Рассмотрим современные разработки средств построения интеллектуальных систем.

## 5.2. Современные программные средства построения интеллектуальных систем

### *5.2.1. Объектно-ориентированный язык Visual Basic*

Visual Basic язык поддерживающий событийно-управляемое программирование (event-driven programming): визуальное проектирование и элементы объектно-ориентированного программирования,

*Возможности языка Visual Basic для создания ЭС*

В основу программного пакета было положено передовое архитектурное решение, позволяющее не писать, а проектировать программы, подобно инженеру-дизайнеру. Иначе говоря, в нем был одним из первых реализован популярный ныне стиль визуального программирования. И ключевым словом в названии языка является Visual -экранные формы и множество встроенных компонент (текстовые, графические окна, кнопки, диалоги и т. п.) избавляют от сложностей, связанней с выводом, обработкой, обновлением всех этих элементов, что особенно важно при разработке экспертной системы, для которой легкая модифицируемость является чуть ли не наиболее важной отличительной чертой.

В Visual Basic реализовано «управление от событий», как уже отмечалось ранее, что позволяет разработчику спроектировать интерфейс пользователя максимально удобно для пользователя. После этапа визуального проектирования программист просто пишет программный код для обработки связанных с

объектом событий, VB предоставляет удобную среду для разработки приложений, тестирования их работы и нахождения и исправления ошибок.

Рассмотрим основные принципы, положенные в основу Visual Basic и позволяющие максимально облегчить труд разработчика и сделать работу с написанным приложением максимально комфортным.

Объект одно из основных понятий Visual Basic и объектно-ориентированного программирования. Объектом называется некая сущность, которая, во-первых, чётко проявляет своё поведение, а во-вторых, является представителем некоторого класса подобных объектов.

Классом объектов в объектно-ориентированных языках называется общее описание таких объектов, для которых характерно наличие множества общих свойств и действий.

СОБЫТИЕ – действие или ситуация, связанная с объектом, например: щелчок мыши или нажатие клавиши. События могут инициироваться в программном коде приложения или непосредственно в системной среде.

СВОЙСТВА – определяют представление, поведение и другие черты объекта. Цвет фона формы, строка соединения (в сеансе БД), размер элементов и т.п. – все это свойства тех или иных объектов.

Свойства: Caption – имя, Enabled – доступ, Height – высота.

МЕТОДЫ – программные процедуры, которые выполняют некоторую обработку, связанную с объектом, определенные действия над объектом.

Объектно-ориентированная модель аналогична во многом фреймовой модели, реализуя обмен сообщениями между объектами, в большей степени ориентирована на решение динамических задач и отражение поведенческой модели. Отличия от фреймовой модели заключаются в чётком понятии класс объектов и экземпляр объекта, а также в способе активации процедур объектами.

Перечислим и дадим краткую характеристику тем средствам VB которые помогают в разработке профессиональных коммерческих приложений вообще и экспертных систем в частности.

Пакет Microsoft Visual Basic представляет собой идеальную платформу для создания интерфейсов с локальными базами данных и базами данных типа клиент/сервер в среде Windows.

Visual Basic позволяет создавать различные приложения для работы с базами данных – от простейших локальных баз данных до многоуровневой архитектуры клиент/сервер, а также приложений для работы в intranet и Internet с использованием таких передовых технологий, как DHTML, XML и ASP.

Также очень важны вопросы проектирования реляционных баз данных и использования языка SQL. Язык структурированных запросов является стандартным средством для работы с базами данных и может использоваться как для интерактивной работы с базами данных, так и включаться в языки программирования. Применительно к Visual Basic SQL позволяет:

создавать, модифицировать или удалять таблицы в базе данных Access;

вставлять, удалять или модифицировать записи таблиц;

получать сводную информацию о данных в таблице;

поиск данных в одной или более таблицах по запросу.

Экспертная система, как никакая другая, должна предоставлять пользователю максимально «дружелюбный» интерфейс, поскольку она в большинстве случаев является диалоговой и требует от пользователя максимального взаимодействия. Это справедливо в большой степени для так называемых оболочек экспертных систем, правильность настройки и заполнения которых является ключевым фактором в построении адекватно реагирующей экспертной системы. В данной ситуации многое зависит от логики представления информации и управляющих элементов в программе, наличия пользовательского меню и развитой справочной системой.

Visual Basic предоставляет разработчику все средства для создания «графического интерфейса пользователя»:

возможность построения много документального интерфейса;

создание пользовательского меню;

травления ACTIVE-X и многое др.

Приведем некоторые подробности реализации механизма вывода с помощью деревьев.

В ЭС принято представлять процесс логического вывода в виде схемы, которая показывается деревом вывода. В системах, база знаний которых насчитывает сотни правил, весьма желательным является использование какой-либо стратегии управления выводом, позволяющей минимизировать время поиска решения и тем самым повысить эффективность вывода. К числу таких стратегий относятся поиск в глубину, поиск в ширину, разбиение на подзадачи.

Visual Basic позволяет разработчику создавать деревья и осуществлять поиск по ним по любому из вышеперечисленных алгоритмов.

Рассмотрим некоторые общие способы реализации деревьев на языке Visual Basic, Один из способов – создать отдельный класс для каждого типа узлов дерева. Для построения дерева порядка 3 можно определять структуры данных для узлов, которые имеют ноль, один, два или три дочерних узла. Этот подход был бы довольно неудобным. Кроме того, что нужно было бы управлять четырьмя различными классами, в классах потребовались бы какие-то флаги, которые бы указывали тип дочерних узлов. Алгоритмы, которые оперировали бы этими деревьями, должны были бы уметь работать со всеми различными типами деревьев.

В качестве простого решения можно определить один тип узлов, который содержит остаточное число указателей на потомков для представления всех нужных узлов. Назовем это методом *полных узлов,* так как некоторые узлы могут быть большего размера, чем необходимо па самом деле.

Для построения дерева порядка 3 с использованием метода полных узлов, требуется определить единственный класс, который содержит указатели на три дочерних узла.

Если порядки узлов в дереве сильно различаются, метод полных узлов приводит к напрасному расходованию большого количества памяти. Некоторые программы добавляют и удаляют узлы, изменяя порядок узлов в процессе выполнения. В этом случае метод полных узлов не будет работать. Такие дина-

мически изменяющиеся деревья можно представить, поместив дочерние узлы в списки. Есть несколько подходов, которые можно использовать для создания списков дочерних узлов. Наиболее очевидный подход заключается в создании в классе узла открытого массива дочерних узлов, как показано в следующем коде. Тогда для оперирования дочерними узлами можно использовать методы работы со списками на основе массивов.

Второй подход состоит в том, чтобы сохранять ссылки на дочерние узлы в связных списках. Каждый узел содержит ссылку на первого потомка. Он также содержит ссылку на следующего потомка на том же уровне дерева. Эти связи образуют связный список узлов одного уровня, поэтому я называю этот метод представлением в виде связного списка узлов одного уровня.

Третий подход заключается в том, чтобы определять в классе узла открытую коллекцию, которая будет содержать дочерние узлы.

Представление *нумерацией связей* позволяет компактно представить деревья, графы и сети при помощи массива. Для представления дерева нумерацией связей, в массиве FirstLink записывается индекс для первых ветвей, выходящих из каждого узла. В другой массив, ToNode, заносятся узлы, к которым ведет ветвь.

Сигнальная метка в конце массива FirstLink указывает на точку сразу после последнего элемента массива ToNode. Это позволяет определить, какие ветви выходят из каждого узла.

Используя представление нумерацией связей, можно быстро найти связи, выходящие из определенного узла. С другой стороны, очень сложно изменять структуру данных, представленных в таком виде.

Последовательное обращение ко всем узлам называется обходом дерева. Обход в глубину начинается с прохода вглубь дерева до тех пор, пока алгоритм не достигнет листьев. При возврате из рекурсивного вызова подпрограммы, алгоритм перемещается по дереву в обратном направлении, просматривая пути, которые он пропустил при движении вниз.

Другой метод перебора узлов дерева – это обход в ширину. Этот метод обращается ко всем узлам на заданном уровне дерева, перед тем, как перейти к

более глубоким уровням. Алгоритмы, которые проводят полный поиск по дереву, часто используют обход в ширину.

Детали реализации обхода зависят от того, как записано дерево. Для обхода дерева на основе коллекций дочерних узлов, программа должна использовать несколько другой алгоритм, чем для обхода дерева, записанного при помощи нумерации связей.

Особенно просто обходить полные деревья, записанные в массиве. Алгоритм обхода в ширину, который требует дополнительных усилий в других представлениях деревьев, для представлений на основе массива тривиален, так как узлы записаны в таком же порядке.

### *5.2.2. Язык логического программирования Visual Prolog*

В октябре 1981 г. была широко распространена информация о японском проекте создания ЭВМ пятого поколения. Многих специалистов удивило, что в основу методологии разработки программных средств было положено логическое программирование. Целью проекта декларировалось создание систем обработки информации, базирующихся на знаниях. Тогда же появляется множество коммерческих реализаций Пролога практически для всех типов компьютеров. К наиболее известным можно отнести CProlog, Quintus Prolog. Silogic Knowledge Workbench. Prolog-2, Arity Prolog, Prolog-86, Turbo Prolog и др.

Наибольшую популярность в нашей стране получила система программирования Turbo Prolog – коммерческая реализация языка для IBM-совместимых ПК. Его первая версия была разработана датской компанией Prolog Development Center (PDC) в содружестве с фирмой Borland International в 1986 г. Система создавалась с серьезными отступлениями от неофициального стандарта языка, самым существенным из которых было введение строгой типизации данных, но это позволило значительно ускорить трансляцию и выполнение программ. Новый компилятор сразу же был по достоинству оценен праграммистами-практиками, хотя и вызвал критику в академических кругах.

В 1988 г. вышла значительно более мощная версия Turbo Prolog 2.0, включающая усовершенствованную интегрированную среду разработки программ. быстрый компилятор и средства низкоуровневого программирования. Кроме того, она предоставляла возможность работы с собственными внешними БД, dBase 111 и Reflex, интегрированным пакетом Lotus 1-2-3. графическим пакетом Paint Brush и другими приложениями. Фирма Borland распространяла эту версию до 1990 г., а затем компания PDC приобрела монопольное право на использование исходных текстов компилятора и дальнейшее продвижение системы программирования на рынок под названием PDC Prolog. В июне 1992 г. появилась версия 3.31 -эффективный универсальный инструмент профессиональных программистов, который вскоре стал одним из наиболее широко используемых. PDC Prolog 3.31 работал в среде MS DOS, OS/2, UNIX, XENIX, PharLap DOS Extender, MS Windows. Эта версия была хорошо совместима с традиционными языками программирования, в первую очередь с Си. В ней были расширены возможности создания приложений с интерфейсом GUI (Graphical User Interface), принятым в MS Windows и OS/2.

Хотя версия PDC Prolog 3.31 уже включала средства для написания программ, работающих под управлением графических операционных систем, процесс разработки подобных приложений все еще носил рутинный характер. Для того чтобы сделать более простыми, удобными и быстрыми процессы написания, тестирования и модификации приложений на языке PDC Prolog, специалисты Prolog Development Center создали систему программирования под названием Visual Prolog 4.0, выпущенную 7 января 1996 г.

При разработке приложений в среде Visual Prolog используется подход, получивший название «визуальное программирование», при котором внешний вид и поведение программ определяются с помощью специальных графических средств проектирования без традиционного программирования на алгоритмическом языке. В результате получили систему программирования, отличающуюся исключительной логичностью, простотой и эффективностью.

В Visual Prolog входят различные элементы: прежде всего, интерактивная среда визуальной разработки (VDE – Visual Develop Environment), которая включает текстовый и различные графические редакторы, инструментальные средства генерации кода, конструирующие управляющую логику (Experts), а также являющийся расширением языка интерфейс визуального программирования (VPI – Visual Programming Interface), Пролог-компилятор, набор различных подключаемых файлов и библиотек, редактор связей, файлы, содержащие примеры и помощь.

Visual Prolog поддерживается различными ОС, в том числе MS-DOS Phar-Lap-Extended DOS. всеми версиями Windows. 16- и 32-битовыми целевыми платформами OS/2, а также некоторыми другими системами, требующими графического пользовательского интерфейса.

В зависимости от выбранного интерфейса разработчику обеспечивается доступ к множеству генераторов кода (Code Expert), всевозможным ресурсным редакторам и особым дополнительным VPI-предикатам. определениям и библиотекам. Ресурсные редакторы применяются для создания, компоновки и редактирования окон, диалогов, меню, панелей инструментов, строк помощи, строковых таблиц, ярлыков, курсоров, битовых карт и оперативной помощи. Генераторы кода на основе подобных структур создают необходимый первичный Prolog-код. В результате появляется первичный код («скелет»), готовый для компиляции, редактирования связей и выполнения.

По желанию программиста генераторы кода могут отобразить любую часть первичного текста программы в окне редактора для просмотра и дополнения в соответствии с логикой приложения, а также для преобразования «скелета» в полноценное приложение. Этот процесс реализуется с помощью различных функций: редактирования, выбора, поиска, перемещения и вставки.

Прикладной программный интерфейс высокого уровня облегчает проектирование Пролог-приложений с утонченным видовым пользовательским решением, использующим графические возможности современных ОС и аппаратных средств отображения информации. Ресурсы и инструментальные средст-

ва, требующиеся для таких приложений (окна, меню, диалоги, органы управления, перья, кисти, курсоры мыши, графические курсоры, рисунки и т. п.), представляются в виде несложных Пролог-структур.

С помощью VPI можно создать мобильный исходный текст, а затем перекомпилировать его для работы как в 16-битовом режиме под управлением MS DOS или Windows, так и в 32-битовом режиме под управлением Windows NT, OS/2 PM и других ОС.

В декабре 1997 г. фирма PDC выпустила Visual Prolog 5.0, а с января 1999 г. приступила к распространению версии 5.1.

Пролог – это язык, который до сих пор находится в развитии. Область его применения постоянно расширяется, в него вносятся новые дополнительные функциональные возможности, призванные удовлетворить возрастающие потребности пользователей,

Visual Prolog является Prolog-системой со 100% оболочкой, выполненной в идеологии Visual, упрощающей разработку программ для систем Windows 3.x. Windows 95/98/2000, Windows NT. Среда разработки приложений системы Visual Prolog включает удобный текстовый редактор, различные редакторы ресурсов, средства разработки гипертекстовых Help-систем, оптимизирующий компилятор. Visual Prolog автоматизирует труд программиста по построению сложных процедур. С его помощью проектирование пользовательского интерфейса и связанных с ним окон, диалогов, меню, кнопок, строк состояния и т.п. производится в графической среде. С созданными объектами сразу же могут работать различные Кодовые Эксперты, генерирующие программные коды на языке Пролог.

Мощность языка Пролог и визуальной среды разработки программ делает простой и интуитивно понятной разработку экспертных систем, основанных на знаниях, систем поддержки принятия решений, планирующих программ, развитых систем управления базами данных и др., а также обеспечивает повышение скорости разработки приложений.

### *5.2.3. Интегрированная инструментальная среда GURU*

В инструментальной среде построения ЭС GURU, разработанной фирмой Micro Data Base Systems, Inc., методы экспертных систем сочетаются с такими средствами обработки данных, как составление электронных ведомостей, управление базой данных и деловой графикой, и таким образом формируется уникальная среда для поддержки принятия решений и разработки прикладных интеллектуальных систем.

Система GURU легка в употреблении для новичков и в то же время является достаточно эффективной и гибкой системой для профессионалов – разработчиков.

В обычных «интегрированных» программных продуктах или несколько отдельных программ помещены в операционную среду, или несколько, второстепенных компонентов вкладываются в главный компонент (как, например, программа обработки электронных ведомостей или текстовый редактор). Трудности, с которыми сталкиваются при таких стилях «интеграции», хорошо известны. Первая трудность заключается в том, что пользователь вынужден переходить назад и вперед по отдельным программам и передавать данные между ними. Метод вложений заставляет пользователя выполнить всю обработку в пределах главного компонента, и в результате получаются относительно слабые вторичные компоненты.

Метод Интеграции, используемый в системе GURU, совершенно отличается от вышеупомянутых. Он основывается на принципе синергизма. При этом под «синергизмом» здесь понимается следующее. В системе GURU все средства всегда доступны. Многочисленные компоненты можно соединять по желанию в пределах одной операции, а это характеризует систему как гибкую и удобную в использовании. Например:

в посылке любого правила ЭС можно делать прямые ссылки на поля базы данных, на ячейки электронных ведомостей, на статические переменные, на программные переменные и массивы;

вывод любого правила ЭС может включать в себя операции управления базой данных, запросы на языке SQL (языке структурированных запросов), операции

обработки электронных ведомостей, генерацию статистических данных, дистанционную передачу данных, выполнение процедур, генерацию деловой графики

поскольку ЭС обосновывает задачу, она может брать консультации у других ЭС, выполнять процедурные модели, просматривать базы данных, составлять электронные ведомости или проводить статистический анализ;

любую ячейку электронной ведомости можно определить в терминах поиска в базе данных или в терминах всей программы, или в терминах консультации с ЭС.

Взаимодействовать с системой можно любым из четырех различных способов: с помощью меню: на ограниченном естественном языке, в режиме команд или через специально разработанные интерфейсы. Каждый тип интерфейса системы GURU предназначен для удовлетворения потребностей и вкусов различных классов пользователей. Всеми четырьмя интерфейсами можно пользоваться во время одного и того же сеанса взаимодействия с системой GURU.

Как и в большинстве оболочек, в GURU используется продукционная модель представления знаний в виде совокупности «If-then» правил с обратной стратегией вывода в качестве основной имеется возможность моделирования нечетких и неточных рассуждений. Кроме посылок и заключения в правила можно включать команды, которые будут выполняться перед проверкой условия, а также пояснительный текст для генерации объяснений. Правила также включают необязательные параметры цены и приоритета, позволяющие управлять процессом выбора из совокупности, готовых, к выполнению правил очередного. С каждым правилом можно также связать число, определяющее, сколько раз это правило может выполняться в процессе консультации.

Правила, относящиеся к решению некоторой общей задачи, образуют базу знаний, или набор правил. В этот набор кроме: собственно правил включаются две специальные процедуры: инициализация и завершение, которые должны выполняться до и после выполнения правил. В набор правил также включаются описания переменных, участвующих в правилах, содержащие спецификации типа, точности и т. п.

По умолчанию в GURU принята стратегия обратного вывода, однако, можно использовать чисто прямой вывод, а также комбинировать его с обратным в рамках одного набора правил. Как стратегиями вывода, так и целевыми переменными можно управлять динамически в процессе консультации.

GURU обеспечивает мощные средства управления обработкой факторов уверенности, отражающих степень неточности и нечеткости выраженных в правилах эвристических знаний. Для предоставления такой нечеткости в GURU с каждым значением переменной может быть связан числовой коэффициент от 0 до 100. Система предоставляет разработчику выбор более чем из 30 различных формул, позволяющих управлять обработкой факторов уверенности во время вывода.

Полезными являются такие дополнительные средства управления логическим выводом, как установка степени «точности» вывода значения для некоторой переменной, изменение принятого по умолчанию порядка просмотра правил.

Эффективность машины логического вывода во многом зависит от того, как она осуществляет поиск в наборе правил, когда ищет правила, которые можно выполнять. В отличие от традиционного программного обеспечения, использующего принципы искусственного интеллекта, система GURU предоставляет расширенные средства управления настройкой, в частности поддерживает до 50 различных стратегий поиска. Эффективность также зависит от количества и состава правил в наборе правил. Поскольку система GURU предоставляет разнообразные возможности создания наборов правил, то можно значительно сократить количество правил, необходимых для охвата всех знаний и опыта в конкретной проблемной области. Это приводит к ускорению процесса получения логических выводов, а также к упрощению управления этими правилами.

Интегрированная система GURU пытается превратить потенциальные преимущества ЭС в реальность, облегчить пользователю процесс создания ЭС, сделать его прямым, эффективным и естественным.

### *5.2.4. Интегрированная инструментальная среда G2 для создания интеллектуальных систем реального времени*

В 1986 г. фирма Gensym вышла на рынок с инструментальным средством G2, версия 1.0. В настоящее время функционирует уже версия 5.2.

Основное предназначение программных продуктов фирмы Gensym (США) помочь предприятиям сохранять и использовать знания и опыт наиболее квалифицированных сотрудников в интеллектуальных системах реального времени, I повышающих качество продукции, надежность и безопасность производства и снижающих производственные издержки.

Классы задач, для которых предназначена G2 и подобные ей системы:

мониторинг в реальном масштабе времени;

системы управления верхнего уровня;

системы обнаружения неисправностей;

диагностика;

составление расписаний;

планирование;

оптимизация;

системы – советчики оператора;

системы проектирования.

Инструментальный комплекс G2 является эволюционным шагом в развитии традиционных экспертных систем от статических предметных областей к динамическим.

Основные принципы, которые заложены в G2:

проблемно/предметная ориентация;

следование стандартам;

независимость от вычислительной платформы;

универсальные возможности, не зависимые от решаемой задачи;

обеспечение технологической основы для прикладных систем;

комфортная среда разработки;

распределенная архитектура клиент-сервер;

высокая производительность.

Основным достоинством оболочки экспертных систем G2 является возможность применять ее как интегрирующий компонент, позволяющий за счет открытости интерфейсов и поддержки широкого спектра вычислительных платформ объединить уже существующие, средства автоматизации в единую комплексную систему управления, охватывающую все -аспекты производственной деятельности – от формирования портфеля заказов до управления технологическим процессом и отгрузки готовой продукции.

На основе базового средства G2– фирма Gensym разработала комплекс проблемно/предметно-ориентированных расширений для быстрой реализации сложных динамических систем на основе специализированных графических языков, включающих параметризуемые операторные блоки для представления элементов технологического процесса и типовых задач обработки информации. Набор инструментальных сред**,** сгруппированный по проблемной ориентации, охватывает все стадии производственного процесса и выглядит следующим образом:

интеллектуальное управление производством – G2 Diagnostic Assistant (GDA), Statistical Process Control (SPC).

оперативное планирование – G2 Scheduling Toolkit (GST), Dynamic Scheduling Packadge (DSP);

разработка и моделирование производственных процессов – G2, ReThink.

управление операциями и корпоративными сетями – Fault Expert.

G2 динамическая система в полном смысле этого слова. Это объектно-ориентированная интегрированная среда для разработки и сопровождения приложений реального времени, использующих базы знаний. G2 функционирует на большинстве существующих платформ: Solaris 1 and 2, Unix, OpenVMS, Windows NT / 2000 Professional / XP. База знаний G2 сохраняется в обычном ASCII-файле, который однозначно интерпретируется на любой из поддерживаемых платформ. Перенос приложения не требует его перекомпиляции и заключается в простом переписывание файлов. Функциональные возможности и внешний вид приложения не претерпевают при этом никаких изменений.

G2 – среда для разработки и развертывания интеллектуальных динамических систем управления. Прикладные программы, написанные в среде G2, могут существенно повысить эффективность выполнения операций, благодаря следующим факторам

непрерывному контролю над потенциальными проблемами прежде, чем они проявят неблагоприятное воздействие;

принятие комплексных оперативных решений на основе информации, полученной посредством рассуждений и анализа данных, содержащихся в интеллектуальной модели процесса;

диагностирование основных случаев возникновения проблем, критичных ко времени выполнения и выработки последовательности правильных действий;

поддержание оптимальных рабочих условий;

координирование действий и информации в выполнении сложных оперативных процессах.

*Использование мощности объектно-ориентированного программирования*

Объекты в G2 – это интуитивный способ представления материальных и абстрактных сущностей в прикладных программах. Мощность объектно-ориентированного подхода к разработке программ, позволяет быстро и легко:

встроить модули и объекты из других прикладных программ;

графически определить объекты, их свойства и действия;

создать новые образцы объектов, имитируя существующие объекты.

Объекты или класс объектов определенные один раз могут использоваться многократно. Любой объект или группа объектов могут иметь несколько экземпляров. Каждый экземпляр наследует все свойства и поведение первоначального объекта (ов). Объекты, правила, и процедуры можно группировать в библиотеки, которые будут общими для всех прикладных программ. Для моделирования широкого разнообразия действий типа производственных процессов, сетевых топологий, информационных маршрутизации, или логических потоков объекты можно объединять графически на экране дисплея.

*Представление знаний, правила, процедуры и модели.*

В G2 можно эффективно создавать и применять общие знания, создавая универсальные правила, процедуры, формулы и зависимости, которые являются применимыми для полных классов объектов. В результате сокращается время на разработку и увеличивается эффективность приложений.

Для представления знаний используется структурный естественный язык, что позволяет облегчить чтение, редактирование и поддержку баз знаний. Это облегчает использование и редактирование приложений пользователем непрограммистом. Для создания и редактирования баз знаний используется Редактор Баз Знаний.

Для представления знаний эксперта о проблемной области используются правила. Правила могут быть как общими, то есть относящимся ко всему классу, так и специфическим, относящимся к конкретным экземплярам класса. Правила возбуждаются автоматически в следующих случаях:

появились новые данные (вывод от фактов к цели);

требуется найти данные (вывод от цели к фактам) для автоматического вызова других правил, процедур, или формул;

требуется определить значения переменных;

каждые *n* секунд для оценивания правила в указанном интервале времени.

Подобно правилам, процедуры выполняются в реальном времени. Процедуры, правила, и модели выполняются одновременно согласно их приоритетам. Процедуры могут использоваться для эффективного представления поведения объектов. Для процедур можно определить состояния ожидания и выполнять обработку информации параллельно. В результате это позволяет формировать мощные прикладные системы реального времени проще и быстрее, чем с помощью традиционных инструментальных средств программирования.

*Работа в Реальном времени*

Работа в реальном времени, операционные решения и реакции зачастую должны быть выполнены мгновенно. Прикладные программы в G2 могут одновременно выполнять рассуждения относительно многократно выполняемых действий в реальном масштабе времени, перерабатывая тысячи правил, вы-

полняя процедуры и модели согласно их приоритетов. Для хранения хронологий данных и событий и для рассуждения относительно поведения через какое-то время используются переменные типа время.

G2 графика может моделировать знание, представляя объекты, связи и зависимости между объектами. Она может рассуждать в терминах связи, следуя сети связанных объектов для определения причин и результатов. Графическая связность объектов позволяет расширить прикладную программу используя графическое объединение аналогов. Графика включает встроенные диаграммы **(**графики**),** таблицы и рисунки и т.д.

G2 также работает с утилитами графического интерфейса Windows. Эти утилиты используют все преимущества объектно-ориентированных возможностей G2.

*Динамическое моделирование и моделирование для анализа* «*что – если*»

G2 позволяют динамически моделировать системы и процессы, используя объекты, правила, процедуры и формулы. Во время разработки модели используются вместо объектов реального мира, что позволяет непрерывно проверять прикладные программы в течение их разработки. Модели могут использоваться на этапе эксплуатации как часть прикладной программы G2 для сравнения фактических и модельных знаний.

Модели можно также использовать для проведения анализа и ответа на вопросы "ЧТО- ЕСЛИ", определения, например, лучших рабочих условий или лучших проектов. Модели можно также использовать для предсказания важных параметров действий в реальном времени, например, качество изделия или затрат.

G2 объединяет в себе как универсальные технологии построения современных информационных систем (стандарты открытых систем, архитектура клиент/сервер, объектно-ориентированное программирование, использование ОС, обеспечивающих параллельное выполнение в реальном времени многих независимых процессов), так и специализированные методы (рассуждения, основанные на правилах, рассуждения, основанные на динамических моделях,

или имитационное моделирование, процедурные рассуждения, активная объектная графика, структурированный естественный язык для представления базы знаний), а также интегрирует технологии систем, основанных на знаниях с технологией традиционного программирования (с пакетами программ, с СУБД, с контроллерами и концентраторами данных и т.д.).

Все это позволяет с помощью данной оболочки создавать практически любые большие приложения значительно быстрее, чем с использованием традиционных методов программирования, и снизить трудозатраты на сопровождение готовых приложений и их перенос на другие платформы.

# 6. ИНТЕЛЛЕКТУАЛЬНЫЕ МУЛЬТИАГЕНТНЫЕ СИСТЕМЫ

Примерно к концу 70-х годов прошлого столетия сформировалась идеология интеллектуальных мультиагентных систем (МАС). Она была основана на результатах работ по исследованию «распределенного искусственного интеллекта». Возникновение же самой идеи распределенного искусственного интеллекта было связано с исследованиями коллективного поведения автоматических устройств (роботов), решающих задачи поведенческого характера.

Быстрое развитие и огромные, дотоле невиданные возможности, которые открывались в связи с созданием, распространением и совершенствованием средств и технологий мировой телекоммуникационной системы Internet, дали импульс лавинообразному развитию информационных технологий. Именно информационный аспект стал определяющим направлением в развитии ИИС.

Если говорить о перспективах, которые вносят в развитие ИИС сетевые телекоммуникационные технологии, то наиболее реальными следует считать те, что связаны с виртуальной гибридизацией ИИС. Под этим следует понимать более высокий уровень интеграции моделей интеллектуальных возможностей человека, чем простое объединение в одной системе, например, двух ИИС, одна из которых представляет собой экспертную систему, а другая – искусственную нейронную сеть. По сути дела, современные сетевые технологии открывают возможность создания сетей распределенных ИИС.

Мультиагентной системой можно считать сетевую совокупность в разной степени «самодостаточных» (автономных) объектов (или агентов), способных получать, хранить, обрабатывать и передавать информацию как в интересах решения собственных, так и корпоративных (общих для группы объектов) задач анализа и синтеза информации.

Агентом такой системы может быть аппаратная, программная, или аппаратно-программная сущность, способная действовать как в «собственных»,

так и в общесистемных интересах. Уровень «интеллекта» агента определяется как возможностями его программы, так и способностью его программы воспринимать (в широком смысле этого понятия) и совокупно целенаправленно использовать интеллектуальные возможности других агентов системы в режиме динамического виртуального общения. Одной из существенных и в настоящее время актуальных областей применения МАС-технологий является создание виртуальных моделей различного класса – эволюционных, имитационных, математических, экономических и т.п.

В этой области сформировались два класса задач – задачи распределенного управления различного рода системами и объектами и задачи планирования целей развитии сложных систем, состоящих из подсистем, цели которых могут характеризоваться относительной независимостью и в тоже время быть взаимосвязанными общей целью. Понятно, что для решения задач обоих классов может потребоваться глобальная кооперация интеллектуальных и ресурсных возможностей всех агентов сети.

В рамках идеологии МАС «интеллектуальные» агенты должны обладать следующими

основными свойствами:

автономность, которая трактуется как способность действовать без вмешательства извне и осуществлять «объективный» самоконтроль своего состояния и результатов своих действий;

целенаправленность и осознанность (за счет объективного самоконтроля) взаимодействия с другими агентами (иногда это свойство принято трактовать как социальное поведение);

адекватная (в широком смысле) реактивность на воздействие внешних факторов;

активность, трактуемая как способность корректировать (уточнять в соответствии реальными условиями) поставленную задачу или сформулировать альтернативную (новую), если поставленная задача уже выполнена или объективно невыполнима;

базовая информированность (агент всегда должен иметь – или обладать возможностью получить – базовые знания о себе, о среде функционирования и о других агентах системы, необходимые и достаточные для соответствия своему предназначению в МАС);

ситуативная информированность (знания, полученные агентом для выполнения конкретных задач и хранящиеся в его памяти, могут со временем потерять актуальность, но агент может об этом не знать и, обладая ими, должен быть убежден в своей готовности решать на основе этих знаний новые задачи);

осознанность текущей цели (полная ясность в отношении совокупности задач или функций, на выполнение которых направлено текущее поведение агента);

способность проявлять (иметь) желание (осознаваемая постоянно или в течение некоторого времени совокупность мотиваций к достижению определенного состояния или совершению определенных действий);

способность осознавать свою обязанность по отношению к другим агентам МАС (задачи, которые получены от других агентов и по разным причинам еще не выполнены, воспринимаются как обязательства, которые должны быть выполнены при первой возможности);

способность проявлять намерения (наличие возможности и умения выстраивать систему приоритетов в отношении очередности выполнения обязательств перед другими агентами).

Иногда к этим свойствам добавляют и другие свойства, которые могут быть обеспечены повышенной сложностью алгоритмов операционного, аналитического и поведенческого толка, реализуемых программами конкретного типа агентов. При описании таких свойств хотелось бы иметь более ясную и недвусмысленную их трактовку в современной литературе по МАС.

Различают три базовых типа архитектуры МАС:

архитектура, основанная на принципах и методах работы со знаниями;

архитектура, основанная на поведенческих моделях;

гибридные построения, сочетающие в различных соотношениях особенности двух первых архитектур.

Системы, построенные в соответствии с методами и принципами обработки знаний, используют традиционные для ИИС модели преставления знаний и методы их обработки (анализа и синтеза). Правда, если вдуматься в приведенную трактовку основных свойств интеллектуальных агентов, то станет ясно, что выразительные возможности логики предикатов первого порядка могут оказаться недостаточными для их описания. В связи с этим и другими обстоятельствами, в МАС данного типа архитектуры применяются более сложные, чем одноместные предикаты, логические выражения и структуры. Хотя это и не избавляет от недостатка, присущего продукционным моделям, – отсутствие возможности эффективного обучения. Агенты систем данного типа архитектуры работают с использованием устоявшейся модели окружающей среды, составленной в символьной форме, а решения принимают на основе формальных рассуждений и по аналогии с известными примерами и образами. В теории МАС такие агенты называются делиберативными.

В системах, построенных на поведенческих моделях, главным образующим понятием является «ситуация», в которой оказался агент, и «побудительный мотив» к выработке адекватного этой ситуации действия или поведения («реакция») агента. Поведенческие модели чаще всего строятся на правилах выработки адекватных реакций. Или, если разнообразие ситуаций не катастрофично – применяются алгоритмы, позволяющие создать конечный автомат с ограниченными возможностями. Агенты МАС данного типа называются реактивными, а их реакция на тот или иной побудительный мотив всегда рассматривается как результат соотнесения состояния внешней среды и внутреннего состояния агента.

Наиболее перспективны гибридные МАС (это весьма похоже на ситуацию с выбором типа ИИС), сочетающие «лучшие» качества поведенческих и продукционных моделей.

Гибридные МАС часто используют принцип специализации своих агентов. Примечательно, что БЗ агентов таких систем имеет знания трех уровней – о предметной области, о возможных взаимодействиях с другими агентами (если

своих недостаточно) и специальные знания, обеспечивающие оптимальное управление системой 218

На развитие МАС существенное влияние оказала теория и практика открытых телекоммуникационных компьютерных систем типа Internet, которым свойственны:

возможность расширения состава функций и способов организации;

простота переноса на другие программно-аппаратные платформы;

способность обмениваться данными и форматами с другими системами;

дружелюбность к пользователю, удобство общения и управления системой.

Одним из важных результатов внедрения концепции открытых систем в технологию создания МАС стало распространение архитектуры «клиент – сервер», представленной двумя моделями сетевого взаимодействия:

«мощный клиент» – «слабый сервер» (сервер организует только доступ к ресурсам, а все функции поиска и обработки данных реализует клиент);

«слабый клиент» – «мощный сервер» (клиент обеспечивает только реализацию интерфейса, а сервер – поиск и все остальные функции сети).

Мобильными агентами в сетевых компьютерных и телекоммуникационных структурах называются программы, которые могут перемещаться по сети. Они покидают компьютер клиента и переходят на удаленный сервер для выполнения задачи, а выполнив, возвращаются обратно.

Мобильные агенты являются перспективными для МАС компонентами, но в настоящее время еще не выработаны стандарты дл разработки таких агентов и не решены проблемы, связанные с перемещением их по сети, поскольку легальные способы перемещения, идентификация агентов, вопросы разграничения доступа (конфиденциальности) и безопасности информации не гарантируют общей надежности сети. Немаловажной проблемой является также «перенаселенность» всемирной сети.

МАС в типовом составе представляет собой структуру, включающую:

множество системных агентов, в котором выделены (организованы) подмножества активных агентов, манипулирующих подмножествами пассивных агентов;

среду, представляющую собой пространство, в котором существуют агенты и объекты;

множество задач (функций, ролей), решение которых поручается агентам;

множество отношений, допустимых между агентами;

множество структур, которые могут быть созданы в системе из множества агентов;

множество действий, доступных агентам.

Современная теория и практика МАС предполагает три типа структурной организации систем:

распределенный искусственный интеллект (РИИ);

децентрализованный искусственный интеллект (ДИИ);

искусственная жизнь (ИЖ).

Структура РИИ создается для решения конкретной интеллектуальной задачи и управляется централизованно, с четкой координацией действий всех агентов, исходя из интересов общей цели – решить задачу с максимальной эффективностью. Под этим обычно понимается одна из двух постановок задач – прямая или обратная. При централизованном управлении системой РИИ задача должна быть оптимально разбита на составляющие ее подзадачи с учетом интеллектуальных возможностей контингента агентов.

Структура ДИИ создается в виде системы управления деятельностью автономных агентов в динамической многоагентной среде, ориентированной на решение нескольких классов интеллектуальных задач.

Структура ИЖ создается с целью исследования и моделирования процессов децентрализованного управления, эволюции, адаптации и кооперации (взаимодействия) структуры, состоящей из большого числа реактивных агентов.

Основными структурами МАС являются РИИ и ИЖ. Главная проблема, с которой сталкиваются создатели РИИ, является разработка оптимального по возможностям состава группы интеллектуальных агентов, способных решить задачу путем рассуждений и связанной с ними обработкой символьной информации. Групповое интеллектуальное поведение образуется на основе ин-

теллектуального поведения и действий индивидуумов. В связи с этим необходимо качественное согласование целей, стратегий и интересов отдельных агентов, концентрация и синхронизация их действий и разрешение конфликтных ситуаций на основе консенсуса.

Структуры РИИ, создаваемые под конкретную задачу, решают три организационных проблемы:

как наилучшим образом распределить задачу на подзадачи с учетом возможностей агентов и выбранного критерия оценки качества полученного решения;

как оптимально распределить властные полномочия между агентами;

как организовать коммуникации по обмену данными и ресурсами в процессе решения задачи.

Структура ИЖ решают проблему интеллектуального поведения в аспекте выживания популяций агентов, сходных по функциям или возможностям, способности адаптироваться к условиям среды обитания, способам самоорганизации и пр. Глобальное интеллектуальное поведение всей системы рассматривается как результат оптимального взаимодействия большого числа простых реактивных агентов.

Основная парадигма структуры ИЖ состоит из следующих посылок:

«искусственная жизнь» – это популяция простых и взаимосвязанных единством цели агентов;

каждый агент самостоятельно определяет свои реакции на воздействие факторов той или иной ситуации в среде обитания и способ взаимодействия с другими агентами;

не существует заранее установленных точных правил, определяющих глобальное поведение популяции агентов;

поведение, свойства и структуры коллективов агентов определяются только локальными взаимодействиями агентов.

Искусственная жизнь в узком смысле – это раздел теории и приложений МАС, изучающий вопросы возникновения интеллектуального поведения на ос-

нове локальных взаимодействий агентов. При этом сами агенты могут и не быть интеллектуальными (например, программы аппроксимации функций). В искусственной жизни могут образоваться искусственные сообщества (их называют роями, наподобие сообщества пчел). Искусственный рой – это множество интеллектуальных агентов, способных взаимодействовать друг с другом, формировать и перестраивать функциональные образования (целевые группы) и совместно решать различные задачи путем параллельных действий. Иначе говоря, искусственный рой представляет собой динамическую сетевую структуру взаимодействующих агентов, в которой происходит согласованное восприятие управляющих и оповещающих сигналов и согласованное воздействие (реагирование) на проявления факторов среды обитания с целью выживания или укрепления своего жизненного статуса, как сообщества. Основными принципами формирование интеллекта роя являются: совместное проживание (соседство), безошибочное определение параметров качества среды обитания, разнообразие ответных реакций, устойчивость организации роя, адаптация к факторам среды обитания.

Искусственная жизнь в широком смысле – это обширная междисциплинарная научно-техническая область, в которой исследуются проблемы создания и исследования искусственных организмов и систем, реализующих принципы и механизмы эволюции систем новых организмов.

Что касается структуры ДИИ, то в ней управление осуществляется только на уровне локальных взаимодействий агентов. Наряду с распределенными знаниями и ресурсами должны быть описаны и локальные задачи отдельных агентов, решаемые на базе локальных концептуальных моделей и локальных критериев. 220

Стоит особо выделить особенности мобильных агентов. Задачи мобильных интеллектуальных агентов в МАС любой структуры могут меняться и быть как локальными, так и глобальными по своему масштабу, а сами мобильные агенты могут иметь статус и «узких специалистов» или «универсалов», то есть быть полностью зависимыми или полностью свободными (автономными) в выборе задач. В этом аспекте можно определить следующие ситуации, характерные для взаимодействия только мобильных агентов МАС любой структуры:

сосуществование (агенты автономны, задачи индивидуальны по постановкам);

сотрудничество ( агенты автономны, задача общая);

совместное выполнение частей общей задачи (агенты специальные, задача общая);

распределенная работа (агенты автономные, задачи индивидуальные).

Возникновение МАС, ее структура и динамика, как сообщества агентов, определяются ситуацией взаимодействия этих агентов, которая, в свою очередь, зависит от совместимости целей, уровня и характера взаимных обязательств, меры ответственности агентов за свои обязательства и от ограниченности ресурсов системы (как частных, так и общих).

Главным принципом организации МАС любой структуры и типа архитектуры является обеспечение эффективного взаимодействия между агентами – установление устойчивых двусторонних динамически настраиваемых отношений между агентами.

Современная практика создания структур МАС предполагает рассматривать взаимодействие между агентами системы не как самоцель, а как средство взаимообразного превращения, преобразования как самих агентов, так и отношений между ними.

Между агентами МАС возможны следующие виды отношений:

кооперация (сотрудничество на основе общности цели, общности намерений или общности желаний);

конкуренция (на основе противоположности или несовпадения цели, намерений, обязательств, желаний, ресурсных потребностей и т.п.);

компромисс (готовность пойти на взаимные уступки в ресурсах или очередности решения задач, совпадающих по времени готовности результатов);

конформизм (отказ от собственных интересов);

негативизм (отказ от общения и взаимодействия в любом вопросе, не предполагая никакой формы противодействия).

Моделирование взаимодействия агентов в МАС характеризуется определенными признаками естественных систем. Эти признаки следует учитывать

при создании моделей различных виртуальных структур таких систем. Основными из этих признаков являются:

продолжительность жизни агента с ее зависимостью от характера и объема выполненной работы и от количества доступных ему жизненных ресурсов;

механизм биологического (естественного) отбора;

принцип организации сообщества агентов, особенно в структурах, отличающихся выраженным социальным характером отношений членов сообщества.

От того, насколько полно, а главное – правильно, будут учтены в модели эти признаки, существенно зависит информативность моделирования.

Модели, создаваемые на основе технологии МАС, довольно быстро завоевали популярность у специалистов различных областей общественной, производственной, экономической и финансовой деятельности. Наиболее известными из них являются такие, в которых координация поведения агентов выглядит естественной для той или иной прикладной задачи. То есть, решение задачи происходит с учетом «естественной» системы ограничений и допущений. Вот краткая характеристика таких моделей.

Игровые модели. Эти модели, как уже было отмечено в теме 4, предполагают конфликтную ситуацию с противоположными интересами в условиях неопределенности и риска. На основе модификации некоторых положений классической теории стратегических игр к постановкам задачи коллективного поведения в МАС, возникла идея, в соответствии с которой игровая модель может адекватно интерпретировать правила переговоров и соглашений между агентами, придерживаясь которых они могут ко взаимному (равновесному) минимаксному или максиминному удовлетворению разрешить любую (или почти любую) из ситуаций: кооперацию, компромисс или конфликт.

Модели планирования оптимального коллективного поведения. Эти модели применяются для случая децентрализованного планирования, когда агенты вправе выбирать наилучшие действия в процессе коллективной координации всех индивидуальных планов.

Это как раз тот момент, когда фактически решается задача «оптимальной» централизации, ибо при возникновении конфликтов индивидуальных планов они могут быть изменены не условиях компромисса.

Модели аукционных торгов. Они позволяют разрешить конкурентные ситуации, возникающие между участниками разных типов аукционов (открытых, когда цена лота известна всем; закрытых, когда цена известна только распорядителю) или, кооперируясь в группу, перебить конкурента и (уже на внутреннем аукционе группы) приобрести лот по более низкой цене.

Модели на основе МАС архитектуры BDI (Belief-Desire-Intention). Эти модели сочетают игровые стратегии с логикой здравых рассуждений человека, использующего понятийные категории «желание», «намерение» и «убеждение». Поведение агентов координируется путем согласования связей, которые каждый агент сформировал в своей системе взаимных предпочтений между этими понятиями, с их выводами о текущем состоянии театра действий. Модели этого типа характеризуются большими вычислительными трудностями, хотя интересны с точки зрения учета психологии игрока.

Интересны и другие направления в моделировании координации поведения агентов, например с использованием положений теории рефлексивных игр и рефлексивного управления, в соответствии с которыми агент или группа агентов, обладающих более высоким рангом рефлексии (способностью просчитать большее число циклов «ход»-«ответ» вперед), имеет более надежные шансы на успех.

Перспективы применения МАС-технологий многие ученые связывают со структурами «искусственная жизнь». Для этого, по мнению Д.А.Поспелова (16), необходимо сосредоточить усилия на ряде многообещающих направлений, к которым относятся:

развитие теории МАС на основе принципов поиска равновесного направления в разрешении конфликтных ситуаций;

исследования путей, механизмов и методов повышения качества (актуальности,

действенности и полезности) знаний, часто утрачиваемого в погоне за их количеством;

использование генетических алгоритмов и алгоритмов групповой (массовой) селекции;

использование возможностей методов рефлексивного управления;

усиления акцентов этического плана в моделях поступков и поведения агентов.

# ЗАКЛЮЧЕНИЕ

Ключевым фактором, определяющим сегодня развитие ИИ-технологий, считается темп роста вычислительной мощности компьютеров, так как принципы работы человеческой психики по-прежнему остаются неясными (на доступном для моделирования уровне детализации). Поэтому тематика ИИ-конференций выглядит достаточно стандартно и по составу почти не меняется уже довольно давно. Но рост производительности современных компьютеров в сочетании с повышением качества алгоритмов периодически делает возможным применение различных научных методов на практике. Так случилось с интеллектуальными игрушками, так происходит с домашними роботами.

Снова будут интенсивно развиваться временно забытые методы простого перебора вариантов (как в шахматных программах), обходящиеся крайне упрощенным описанием объектов. Но с помощью такого подхода (главный ресурс для его успешного применения – производительность) удастся решить, как ожидается, множество самых разных задач (например, из области криптографии). Уверенно действовать автономным устройствам в сложном мире помогут достаточно простые, но ресурсоемкие алгоритмы адаптивного поведения. При этом ставится цель разрабатывать системы, не внешне похожие на человека, а действующие, как человек.

Ученые пытаются заглянуть и в более отдаленное будущее. Можно ли создать автономные устройства, способные при необходимости самостоятельно собирать себе подобные копии (размножаться)? Способна ли наука создать соответствующие алгоритмы? Сможем ли мы контролировать такие машины? Ответов на эти вопросы пока нет.

Продолжится активное внедрение формальной логики в прикладные системы представления и обработки знаний. В то же время такая логика не способна полноценно отразить реальную жизнь, и произойдет интеграция различных систем логического вывода в единых оболочках. При этом, возможно,

удастся перейти от концепции детального представления информации об объектах и приемов манипулирования этой информацией к более абстрактным формальным описаниям и применению универсальных механизмов вывода, а сами объекты будут характеризоваться небольшим массивом данных, основанных на вероятностных распределениях характеристик.

Сфера ИИ, ставшая зрелой наукой, развивается постепенно – медленно, но неуклонно продвигаясь вперед. Поэтому результаты достаточно хорошо прогнозируемы, хотя на этом пути не исключены и внезапные прорывы, связанные со стратегическими инициативами. Например, в 80-х годах национальная компьютерная инициатива США вывела немало направлений ИИ из лабораторий и оказала существенное влияние на развитие теории высокопроизводительных вычислений и ее применение во множестве прикладных проектов. Такие инициативы будут появляться скорее всего на стыках разных математических дисциплин – теории вероятности, нейронных сетей, нечеткой логики.

# ЛИТЕРАТУРА

1. Андрейчиков А.В., Андрейчикова О.Н. Интеллектуальные информационные системы. Учебник. – М.: Финансы и статистика, 2004.-424с.

2. Арсеньев С.Н., Шелобов С.И., Давыдова Т.Ю. Принятие решений. Интегрированные информационные системы. Учебное пособие для вузов. –М.: ЮНИТИ-ДАНА, 2003. -270 с.

3. Гаврилов Т.Л. Хорошевская В.Ф. Базы знаний интеллектуальных систем. – СПб.: Питер, 2001. – 384с.

4. Джексон П. Введение в экспертные системы / Учебное пособие. – М.: «Вильямс», 2001 – 624 с.

5. Одинцов Б.Е. Проектирование экономических экспертных систем / Учебное пособие для вузов. – М.: Компьютер, ЮНИТИ, – М.: 1996. – 166с.

6. Леденева Т.М., Подвальный С.Л. Системы искусственного интеллекта и принятие решений; Учебное пособие; Уфа: УГАТУ, 2005.- 246с.

7. Матвеев М.Г. Модели и методы искусственного интеллекта. Применение в экономике: Учеб. пособие / М.Г. Матвеев, А.С. Свиридов, Н.А. Алейникова. – М.: Финансы и статистика; ИНФА-М, 2008.

8. Нейроинформатика. / А.Н.Горбань, В.Л.Дунин-Барковский, А.Н.Кирдин и др. – Новосибирск: Наука, Сибирская издательская фирма РАН, 1998. – 234с.

9. Путькина Л.В., Пискунова Т.Г. Интеллектуальные информационные системы: учеб. пособие. – СПб: Изд-во СПбГУП, 2008.

10. Представление и использование знаний: Пер. с англ. / Под ред. Х. Уэно, М. Исидзука. – М.: Мир, 1989.

11. Смолин Д.В. Введение в искусственный интеллект: конспект лекций. – 2-е изд, перераб. – М.: М. ФИЗМАТЛИТ, 2007.

12. Статические и динамические экспертные системы. Учебное пособие / Э.В.Попов и др. – М.: Финансы и статистика, 1996. -194с.

13. Ясницкий Л.Н. Введение в искусственный интеллект: Учеб. пособие для студ. высш. учеб. заведений – М.: Издательский центр "Академия", 2005.

# Оглавление

ВВЕДЕНИЕ .....3

1. МОДЕЛИ ПРЕДСТАВЛЕНИЯ ЗНАНИЙ, ОСНОВАННЫЕ НА ПРАВИЛАХ .....4

1.1. Основные понятия инженерии знания .....5

1.2. Методы представления знаний .....11

1.2.1. *Логические модели* .....11

1.2.2. *Продукционная модель представления знаний* .....17

1.2.3. *Фреймовая модель представления знаний* .....22

1.2.4. *Сетевые модели представления знаний* .....32

1.2.5. *Сценарии* .....40

1.2.6. *Представления знаний на основе нечеткой логики* .....43

2. МЕТОДЫ ПОИСКА РЕШЕНИЙ .....53

2.1. Представление задачи в пространстве состояний .....53

2.2. Стратегии поиска в пространстве состояний .....56

2.3. Поиск в глубину и в ширину .....59

2.4. Слепой перебор и бектрекинг .....63

2.5. Эвристический поиск .....65

2.5.1. *Алгоритм эвристического поиска* .....66

2.5.2. *Алгоритм «подъема на холм»* .....70

2.6. Редукция задач на подзадачи .....72

2.7. Нечеткий логический вывод .....83

2.8. Алгоритм поиска пути на графе ..... 88

3. МЕТОД АНАЛИЗА ИЕРАРХИЙ ..... 91

4. ИСКУССТВЕННЫЕ НЕЙРОННЫЕ СЕТИ ..... 108

4.1. Возможности и основные положения искусственных нейронных сетей . 108

4.2. Алгоритм обратного распространения ..... 120

4.3. Нейронные сети: обучение без учителя ..... 126

4.4. Нейронные сети Хопфилда и Хэмминга ..... 130

4.5. Пример использования правила обратного распространения ..... 137

5. ИНСТРУМЕНТАЛЬНЫЕ СРЕДСТВА ПРОЕКТИРОВАНИЯ ИНТЕЛЛЕКТУАЛЬНЫХ СИСТЕМ. ..... 142

5.1. Анализ традиционных языков программирования и представления знаний. ..... 143

5.1.1. *Специализированный язык LISP.* ..... 143

5.1.2 *Фрейм – ориентированный язык FRL* ..... 145

5.1.3 *Язык логического программирования PROLOG* ..... 146

5.1.4. *Продукционный язык OPS* ..... 148

5.2. Современные программные средства построения интеллектуальных систем ..... 153

5.2.1. *Объектно-ориентированный язык Visual Basic* ..... 153

5.2.2. *Язык логического программирования Visual Prolog* ..... 158

5.2.3. *Интегрированная инструментальная среда GURU* ..... 162

5.2.4. *Интегрированная инструментальная среда G2 для создания интеллектуальных систем реального времени* ..... 165

6. ИНТЕЛЛЕКТУАЛЬНЫЕ МУЛЬТИАГЕНТНЫЕ СИСТЕМЫ ..... 171

ЗАКЛЮЧЕНИЕ ..... 183

ЛИТЕРАТУРА ..... 185

***Публикуется в авторской редакции***

Компьютерная верстка П.Г. Немашкалов

Подписано в печать 1.12.10

Формат 60x84 $^{1}/_{16}$ Усл.печ.л. 10,93 Уч.-изд.л. 8,68

Бумага офсетная Тираж 100 экз. Заказ 81

Отпечатано в ООО «РБК-Сервис».