О. В. Герман

ЭКСПЕРТНЫЕ СИСТЕМЫ

Рекомендовано УМО вузов Республики Беларусь по образованию
в области информатики и радиоэлектроники в качестве учебно-методического пособия
для студентов учреждений, обеспечивающих получение высшего образования
по специальности «Автоматизированные системы обработки информации»

Минск БГУИР 2008

УДК 681.3 (075.8)
ББК 32.973.202-018.2 я 7
Г 38

Р е ц е н з е н т ы:

профессор кафедры информационных систем и технологий БГТУ,
д-р техн. наук П. П. Урбанович;

заведующий кафедрой автоматизированных информационных систем
Минского института управления, д-р техн. наук, проф. В. И. Курмашев

Герман, О. В.
Г 38 Экспертные системы : учеб.-метод. пособие / О. В. Герман. – Минск : БГУИР, 2008. – 91 с. : ил.
ISBN 978-985-444-911-1

Пособие посвящено проблематике экспертных систем. Рассматриваются вопросы представления знаний, построения машины вывода для моделей знаний, использующих классические и неклассические логические формализации. Описаны вопросы, связанные с реализацией логического программирования в среде Visual Prolog, а также альтернативный подход к логическому программированию.

Пособие предназначено для студентов, аспирантов и инженеров, имеющих дело с теоретико-прикладными аспектами реализации экспертных систем.

УДК 681.3 (075.8)
ББК 32.973.202-018.2 я 7

ISBN 978-985-444-911-1

Содержание

1. ОСНОВЫ

1.1. ВВЕДЕНИЕ

Концепция экспертных систем была предложена американцами Э. Фейгенбаумом и Д. Мозесом. Эта концепция основана на использовании базы знаний для решения проблем. В прикладной математике сформировалось несколько подходов к решению проблем вообще. Впервые глобальную идею в 17 в. предложил немецкий математик и философ Г. Лейбниц, который поставил задачу создать универсальный решатель задач. Эта глобальная идея упирается в два пункта:

- язык для описания задач;
- универсальный алгоритм (решатель).

После создания немцем Г. Фреге языка предикатов в его современном варианте (конец 19 в.) полагали, что именно логика даст ключ к реализации идеи Лейбница. Суть этого подхода состоит в следующем. Всякая задача может быть сформулирована как теорема существования решения. Эта теорема подлежит доказательству стандартными средствами, например с помощью техники приведения к противоречию (reductio ad absurdum). После получения доказательства из него извлекается ответ на поставленную задачу.

Уязвимым местом этого подхода оказался результат, установленный А. Черчем, о невозможности построить алгоритм для доказательства (опровержения) произвольной теоремы в теории, не менее сильной, чем арифметика целых чисел. Данный результат известен как результат о неразрешимости логики первого порядка в широком смысле. Раньше Черча австрийский математик К. Гедель доказал существование алгоритмически неразрешимых проблем в любых формальных теориях, содержащих арифметику целых чисел. Результат Черча является частным по отношению к результату Геделя. Таким образом, в решении проблем нельзя преодолеть общую проблему алгоритмической неразрешимости (бесконечного, заметим, перечня задач), оставаясь на позициях существующих формальных средств.

Тем не менее, в связи с появлением ЭВМ, развитием технологической базы логические и аксиоматические идеи нашли реализацию в прикладной сфере, включая экспертные системы.

В 1965 г. Дж. Робинсон разработал принцип резолюций в логике – эффективный и удобный для вычислительной реализации метод логического вывода. Это обстоятельство создало предпосылки для практической реализации программ прикладной логики.

Концепцию логического программирования создал Р. Ковальский. Заслуга Ковальского в том, что он придумал процедурную интерпретацию логических формул (сами формулы рассматриваются как программные вызовы). Логическая программа состоит из наборов альтернативных вызовов.

Первая реализация языка логического программирования Пролог осуществлена в 70-х годах прошлого века в Марселе (Франция) на основе языка программирования Паскаль. Работа проведена под руководством Кольмероэра. В это же время широким фронтом ведутся работы в области **нечеткой логики** и теории нечетких множеств (создана Л. Заде в 60-е годы). Эти работы получили официальный статус в японской программе создания ЭВМ пятого поколения. Японская программа подстегнула исследовательские работы во многих странах, включая СССР. Хотя результаты ее оказались не столь значительными, как ожидалось в начале, данная программа позволила четко уяснить существующие возможности и ограничения.

Экспертные системы (ЭС) – это технологии извлечения решения задачи из базы знаний, представленных в удобном для машинной обработки виде.

Два основных механизма ЭС – база знаний и машина вывода – соответствуют двум составляющим системы решения задач – языку описания задачи и алгоритму решения задачи.

В настоящее время ЭС нашли широкое коммерческое применение, включая медицину, психологию, промышленное планирование, теорию изобретений, САПР и пр. Задачи, которые решают ЭС, относятся к категории *интеллектуальных*.

В пособии сделан акцент на логических моделях знаний и логической машине вывода. В литературе по экспертным системам логические исчисления представлены значительно меньше, чем, например, системы, использующие аппарат математической статистики. Однако роль логических исчислений для практики представляется весьма существенной.

1.2. ХАРАКТЕРИСТИКА ИНТЕЛЛЕКТУАЛЬНЫХ ЗАДАЧ

Особенности задач, решаемых в ЭС, в значительной мере определяют выбор методов и подходов к их решению. Можно указать следующие отличительные особенности таких задач, объединив их под общим названием «интеллектуальные»:

- задача является не полностью определенной;
- критерий, как правило, отсутствует;

- нет «хорошего» алгоритма решения, т.е. либо алгоритм является переборным, либо алгоритм не известен;
- задача характеризуется значительным пространством поиска;
- имеются эксперты по рассматриваемой проблематике.

Следует заметить, что для отнесения задачи к категории «интеллектуальных» не обязательно одновременное выполнение всех указанных условий.

Частичная определенность задачи связана с недостаточными знаниями о предметной области. Для некоторых задач это свойство является определяющим. Сама задача построения логического вывода, а также многие оптимизационные задачи комбинаторной и дискретной математики обладают свойством частичной определенности, поскольку не для всякой формулы разрешим вопрос о ее истинности. Важной особенностью любой технологии, связанной с решением интеллектуальных задач, является использование **слабых методов,** к которым относятся: ограниченный направленный перебор, исключение и отсечение, эвристики, индукция.

Основная проблема применения слабых методов поиска решения состоит в том, чтобы на их основе построить точный или статистически точный метод.

В экспертных системах особенно с логическими моделями знаний велика роль **человека-решателя**. В настоящее время это обусловлено его незаменимостью в части генерации гипотез и обработки неформализованных знаний. Технология решения задач человеком отличается от машинной арифметики. Интуитивная составляющая механизма угадывания имеет тем большее значение, чем менее формализована задача.

Таким образом, представление задачи играет важную роль в плане разделения ролей человека и машины в процессе решения. Общую парадигму ЭС с учетом сказанного можно представить следующим образом:

ЭС= База Знаний + Машина Вывода + Человек-решатель (1.1)

1.3. МОДЕЛИ ЗНАНИЙ

Имеется несколько хорошо разработанных моделей знаний, обладающих своими достоинствами и недостатками. Это

- логические модели;
- продукционные модели;
- фреймы;
- спецификации и семантические сети.

1.3.1. Логическая модель знаний

Для логических моделей имеется мощный математический аппарат. Логическая модель представляет конечное или бесконечное множество логических формул. Для записи формул используется язык, содержащий алфавит (набор символов) и множество правил образования правильно построенных формул. Символами логического языка являются:

- константы (например, 0,1,2,… или $a,b,c,\ldots$);
- переменные: $x,y,\ldots, z$;
- функциональные символы: $f,g,h, \ldots$ (функциональные символы называются также функторами);
- символы предикатов (отношений) $P,Q,R,..$;
- кванторы всеобщности $\forall$ и существования $\exists$;
- логические операции (связки): & («и»), v («или»), $\neg$ («не»), $\rightarrow$ («следует»), $\leftrightarrow$ («эквивалентно»). & называют операцией конъюнкции, v – дизъюнкции, $\neg$ – отрицания, $\rightarrow$ – импликацией и $\leftrightarrow$ – эквиваленцией.

Кроме символов языка рассматриваются синтаксические правила записи предложений (формул) логического языка. Отправным элементом является простейшая формула – **предикат** (в логике предикатов) и булевская переменная (в логике высказываний).

Предикат – это простейшая (атомарная) формула (с отрицанием или без него) вида $P(\ldots)$, где в скобках указываются аргументы формулы – переменные, константы или функторы – либо вовсе не указываются аргументы. Предикат принимает одно из двух возможных значений: истина или ложь. Аргументы предиката имеют произвольную природу. Пусть P, Q – любые две формулы. Тогда формулами также являются следующие выражения:

1. $\neg P$, $\neg Q$;
2. P v Q;
3. P & Q;
4. $P \rightarrow Q$; (**1.2**)
5. $P \leftrightarrow Q$;
6. $\forall x\, P(x)$;
7. $\exists\, x\, P(x)$

(в пп. 6 и 7 переменная x входит свободно в $P(\ldots)$, т.е. x не связана в P каким-либо квантором $\forall,\exists$). В дальнейшем знак &, как правило, будем опускать, а знак $\leftrightarrow$заменять на = или $\equiv$.

Определение 1.1

А. Дизъюнктом в логике высказываний называется дизъюнкция литер (булевских переменных) с отрицанием или без, например $x \vee y \vee \neg z$.

В. Дизъюнктом в логике предикатов называется дизъюнкция литералов (атомарных формул), взятых с отрицанием или без него, например $P(a,z) \vee \neg Q(z,f(x))$.

Определение 1.2. Конъюнктивной нормальной формой (КНФ) называется конъюнкция дизъюнктов.

Определение 1.3. Выполняющей интерпретацией I для заданной КНФ называется множество значений переменных этой КНФ, при которых она истинна. Например, КНФ $(x_1 \vee \neg x_2)(\neg x_1 \vee x_2)(\neg x_2 \vee \neg x_3)(x_1 \vee x_3)$ истинна в интерпретации $I=<x_1=1,x_2=1,x_3=0>$ либо в интерпретации $<x_1=0, x_2=0, x_3=1>$.

Число различных интерпретаций для дизъюнкта с $n>0$ переменными равно 2^n. Однако не все они являются в общем случае выполняющими интерпретациями.

Определение 1.4. Задача ВЫПОЛНИМОСТЬ (коротко – ВЫП) формулируется так: для данной КНФ установить, имеется ли для нее хотя бы одна выполняющая интерпретация.

Для задачи ВЫП известно, что она является полиномиально универсальной в классе задач, разрешимых за полиномиальное время на недетерминированной машине Тьюринга, однако найти для нее полиномиальный алгоритм для детерминированной машины Тьюринга до сих пор не удалось. Следовательно, ВЫП можно отнести к категории интеллектуальных задач.

Определение 1.5. Формула B следует из формулы A, если в любой интерпретации, где A истинна, B также истинна. Пишем $A \rightarrow B$ в этом случае.

Определение 1.6. Формула A тождественно истинна, если она истинна в любой интерпретации. Тождественно истинная формула называется также тавтологией.

Определение 1.7. Правилом вывода R называется такое соотношение между формулами $A_1, A_2, \ldots, A_n$ и B, которое устанавливает истинность формулы B всякий раз, когда выполняется заданное соотношение.

Определение 1.8. Формула B выводима из формул $A_1, A_2, \ldots, A_n$, если имеется конечная последовательность формул $П$, начинающаяся с любой из формул A_i , такая, что каждая очередная формула этой последовательности либо выводима по некоторому правилу вывода из предшествующих членов (или их части), либо совпадает с какой-то из формул A_i .

Теорема 1.1 (о полноте в узком смысле логики первого порядка и логики высказываний). Всякая тождественно истинная формула выводима.

Задача логического вывода в логике высказываний может быть эффективно сведена к ВЫП. В самом деле, пусть даны дизъюнкты D_1, D_2, …, D_z. Спрашивается, выводим ли из них дизъюнкт R? (Т.е. требуется установить тождественную истинность формулы $D_1 \& D_2 \& \ldots \& D_z \rightarrow R$.) Умножим эту последнюю формулу справа и слева на $\neg R$. Получим:

$$D_1 \& D_2 \& \ldots \& D_z \& \neg R \rightarrow False.$$

Следовательно, если удастся показать выполнимость системы дизъюнктов $D_1 \& D_2 \& \ldots \& D_z \& \neg R$, то получим опровержение исходной формулы $D_1 \& D_2 \& \ldots \& D_z \rightarrow R$. Если докажем, что система не выполнима, то получим доказательство исходной формулы. Таким образом, задача логического вывода сводится к задаче ВЫП.

Рассмотрим описание задач на языке логики. Такое описание потребует формализовать некоторые наиболее часто встречаемые условия.

Прежде всего, таким условием является

$$\begin{aligned} & x_1 + x_2 + \ldots + x_n = 1, \\ & x_i \in \{0,1\}. \end{aligned} \qquad \textbf{(1.3)}$$

Это условие эквивалентно следующей системе логических формул:

$$\begin{cases} x_1 \vee x_2 \vee \ldots \vee x_n, \\ \neg x_1 \vee \neg x_2, \\ \neg x_1 \vee \neg x_3, \\ \ldots \\ \neg x_1 \vee \neg x_n, \\ \ldots \\ \neg x_2 \vee \neg x_n, \\ \ldots \\ \neg x_{n-1} \vee \neg x_n. \end{cases} \qquad \textbf{(1.4)}$$

Количество всех дизъюнктов равно C_n^2 +1. Число дизъюнктов можно уменьшить, если использовать дополнительные переменные и провести следующие преобразования.

Обозначим через $\boldsymbol{F}$ следующую систему формул:

$$\boldsymbol{F} \equiv \begin{cases} x_1 + x_2 + y_1 = 1, \\ x_3 + x_4 + y_2 = 1, \\ \ldots \\ x_{n-1} + x_n + y_{n/2} = 1. \end{cases}$$

Замечание. Если n – нечетное, то последняя формула заменяется на $x_n + y_{(n+1)/2} = 1$.

В $\boldsymbol{F}$ y_i – дополнительные переменные. Теперь система (**1.3**) оказывается эквивалентной системе

$$\boldsymbol{G} \equiv \begin{cases} F, \\ \neg y_1 + \neg y_2 + \ldots + \neg y_{n/2} = 1. \end{cases}$$

В свою очередь формулу $\neg y_1 + \neg y_2 + \ldots + \neg y_{n/2} = 1$ можно заменить эквивалентной системой $\boldsymbol{H}$:

$$\boldsymbol{H} \equiv \begin{cases} \neg y_1 + \neg y_2 + z_1 = 1, \\ \neg y_3 + \neg y_4 + z_2 = 1, \\ \neg y_{(n/2)\ 1} + \neg y_{n/2} + z_{n/4} = 1, \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ \neg z_1 + \neg z_2 + \ldots + \neg z_{n/4} = 1. \end{cases}$$

Следовательно, система (**1.4**) становится эквивалентной системе $\boldsymbol{J}$ с булевскими переменными

$$\boldsymbol{J} = \begin{cases} \boldsymbol{F} \\ \boldsymbol{H} \end{cases}. \qquad (\mathbf{1.5})$$

Описанные преобразования по аналогии применяем к системе (**1.5**) и т. д. до тех пор, пока не останутся только трехлитерные или двухлитерные уравнения вида $a+b+c=1$ или $d+e=1$. Каждое трехлитерное уравнение дает 4 дизъюнкта: $a \vee b \vee c$, $\neg a \vee \neg b$, $\neg a \vee \neg c$, $\neg b \vee \neg c$. Каждое двухлитерное уравнение $c+d=1$ дает 2 дизъюнкта: $c \vee d$, $\neg c \vee \neg d$. Следовательно, рассматриваемый способ построения эквивалентной системы дизъюнктов для (**1.3**) дает не выше $4(n–1)$ дизъюнктов (уточнение этой формулы требует принять во внимание четность/нечетность n). Приведем пример.

$$x_1 + x_2 + \ldots + x_{10} = 1,$$
$$x_i \in \{0,1\}.$$

Последовательно получаем эквивалентные системы:

$$\begin{cases} x_1 + x_2 + y_1 = 1, \\ x_3 + x_4 + y_2 = 1, \\ x_5 + x_6 + y_3 = 1, \\ x_7 + x_8 + y_4 = 1, \\ x_9 + x_{10} + y_5 = 1, \\ \neg y_1 + \neg y_2 + \neg y_3 + \neg y_4 + \neg y_5 = 1 \end{cases} \quad (\mathbf{1.6})$$

и

$$\begin{cases} x_1 + x_2 + y_1 = 1, \\ x_3 + x_4 + y_2 = 1, \\ x_5 + x_6 + y_3 = 1, \\ x_7 + x_8 + y_4 = 1, \\ x_9 + x_{10} + y_5 = 1, \\ \neg y_1 + \neg y_2 + z_1 = 1, \\ \neg y_2 + \neg y_3 + z_2 = 1, \\ \neg y_5 + z_3 = 1, \\ \neg z_1 + \neg z_2 + \neg z_3 = 1. \end{cases} \quad (\mathbf{1.7})$$

Теперь все уравнения (кроме одного) трехчленные. Каждое из них дает 4 дизъюнкта. Всего дизъюнктов 34, вместо $C_{10}^2 + 1 = 46$.
Рассмотрим другие важные условия.

$$x_1 + x_2 + \ldots + x_n \le 1, \quad (\mathbf{1.8})$$
$$x_i \in \{0,1\}.$$

Это условие трансформируется в эквивалентную систему двухлитерных дизъюнктов

$$\begin{cases} \neg x_1 \vee \neg x_2, \\ \neg x_1 \vee \neg x_3, \\ \quad \ldots \\ \neg x_1 \vee \neg x_n, \\ \quad \ldots \\ \neg x_2 \vee \neg x_n, \\ \quad \ldots \\ \neg x_{n-1} \vee \neg x_n. \end{cases}$$

На практике важно условие вида

$$x_1 + x_2 + \ldots + x_n = k\,, \qquad (\mathbf{1.9})$$
$$x_i \in \{0,1\},\ k > 1,\ k \le n,\ k - \text{целое}.$$

Прямое кодирование этого равенства при $k=n/2$ (n – четное) требует экспоненциально растущего от n числа дизъюнктов. Поэтому следует вводить новые переменные. В целях ясности обратимся к иллюстративному примеру.

$$x_1 + x_2 + \ldots + x_8 = 3. \qquad (\mathbf{1.10})$$

Равенство (1.10) можно заменить следующей эквивалентной системой равенств

$$\begin{cases} x_1 + y_4 + y_5 + y_6 + y_7 + y_8 = 1, \\ x_2 + z_4 + z_5 + z_6 + z_7 + z_8 = 1, \\ x_3 + w_4 + w_5 + w_6 + w_7 + w_8 = 1, \\ y_4 + z_4 + w_4 \le 1, \\ y_5 + z_5 + w_5 \le 1, \\ \ldots \\ y_8 + z_8 + w_8 \le 1, \\ x_4 \equiv y_4 \vee z_4 \vee w_4, \\ x_5 \equiv y_5 \vee z_5 \vee w_5, \\ \ldots \\ x_8 \equiv y_8 \vee z_8 \vee w_8. \end{cases}$$

При этом для k первых переменных из (**1.9**) записываем k равенств вида

$$\begin{cases} x_1 + y_{k+1} + y_{k+2} + y_{k+3} + \ldots + y_n = 1, \\ x_2 + z_{k+1} + z_{k+2} + z_{k+3} + \ldots + y_n = 1, \\ \ldots \\ x_k + w_{k+1} + w_{k+2} + w_{k+3} + \ldots + w_n = 1. \end{cases} \qquad (\mathbf{1.11})$$

Пусть $x_1 = x_2 = \ldots = x_k = 0$. Тогда, например, $y_a = z_b \ldots = w_c = 1$, причем $a \ne b$, $a \ne c$, …, $b \ne c$. Следовательно, в силу наших условий, некоторые k переменных из числа $x_{k+1}, x_{k+2}, \ldots, x_n$ будут иметь единичные значения, обеспечивая условие (**1.9**). Если, например, $x_1 = 1$, $x_2 = \ldots = x_k = 0$, то некоторые $(k - 1)$ переменных (и только они) из числа переменных $x_{k+1,}$

x_{k+2},…, x_n будут иметь единичные значения и т.п. Число дизъюнктов для (**1.11**) составит порядка $4k(n-1)$. Общее число дизъюнктов для (**1.9**) составит порядка $4k(n-1)+ C_{n-k}^{2}$ (эквивалентности

$$x_{k+1} \equiv y_{k+1} \vee z_{k+1} \vee \ldots \vee w_{k+1},$$
$$x_{k+2} \equiv y_{k+2} \vee z_{k+2} \vee \ldots \vee w_{k+2},$$
$$\ldots$$
$$x_n \equiv y_n \vee z_n \vee \ldots \vee w_n$$

в систему дизъюнктов преобразованной задачи нет необходимости включать).

Пример 1.1. В хищении подозреваются *A*, *B* и *C*.
1. Никто, кроме *A*, *B*, *C*, в хищении не замешан.
2. *A* никогда не ходит на дело без соучастников.
3. *C* не виновен, если виновен *B*.

Спрашивается, виновен ли *A*?

Нужно составить систему логических уравнений, введя необходимые булевские переменные. Обозначим: x_a – виновен *A*, x_b – виновен *B*, x_c – виновен *C*. Составим следующую базу знаний:

1. $x_a \vee x_b \vee x_c$,
2. $x_a \rightarrow x_b \vee x_c$,
3. $x_b \rightarrow \neg x_c$.

Эти логические формулы соответствуют условиям задачи. Можно воспользоваться арифметическим представлением логических формул. Этот прием осуществляет связь между логическими формулами и рассмотренными выше условиями формализации ((**1.3**), (**1.8**), (**1.9**) и др.). Арифметические представления таковы:

$$f \vee g \vee \ldots \vee h \equiv f + g + \ldots + h \geq 1,$$
$$\neg f \equiv (1- f), \qquad (\mathbf{1.12})$$
$$f \rightarrow g \equiv \neg f \vee g \equiv (1- f) + g \geq 1,$$
$$f \,\&\, g \equiv f + g = 2 \equiv \neg f + \neg g = 0 \equiv f \cdot g.$$

Теперь условия нашей задачи получат следующее представление:

1. $x_a + x_b + x_c \geq 1$,
2. $(1-x_a) + x_b + x_c \geq 1$,
3. $(1-x_a) + (1- x_c) \geq 1$.

Относительно этой модели знаний поставлен вопрос: верно ли, что $x_a = 1$? Ответ на этот вопрос должна дать **машина вывода**.

Пример 1.2

Следует назначить по одной работе *w*1, *w*2, *w*3, *w*4 четырем исполнителям *a*, *b*, *c*, *d*. Известно, что *a* может выполнить либо работу *w2*, либо работу *w3; b* может выполнить работу *w1* или *w2*; *c* может выполнить *w1*, или *w3*, или *w4*; *d* может выполнить работу *w2* или *w4*.
Базу знаний для этой задачи можно записать с помощью формул логики предикатов таким образом:

//группа условий, показывающих, какие работы могут быть назначены //исполнителям

$$P(a,X) \rightarrow (X = w2) \text{ v } (X = w3),$$
$$P(b,Y) \rightarrow (Y = w1) \text{ v } (Y = w2),$$
$$P(c,Z) \rightarrow (Z = w1) \text{ v } (Z = w3) \text{ v } (Z = w4),$$
$$P(d,R) \rightarrow (R = w2) \text{ v } (R = w4).$$

//группа условий, показывающих, какие исполнители могут быть назначены //на работы

$$P(X1,w1) \rightarrow (X1 = b) \text{ v } (X1 = c),$$
$$P(Y1,w2) \rightarrow (Y1 = a) \text{ v } (Y1 = b) \text{ v } (Y1 = d),$$
$$P(Z1,w3) \rightarrow (Z1 = a) \text{ v } (Z1 = c),$$
$$P(R1,w4) \rightarrow (R1 = c) \text{ v } (R1 = d).$$

//группа условий, показывающих, что исполнителям не могут назначаться //одинаковые работы

$$P(a,X) \,\&\, P(b,Y) \rightarrow (X <> Y),$$
$$P(a,X) \,\&\, P(c,Y) \rightarrow (X <> Y),$$
$$P(a,X) \,\&\, P(d,Y) \rightarrow (X <> Y),$$
$$P(b,X) \,\&\, P(c,Y) \rightarrow (X <> Y),$$
$$P(b,X) \,\&\, P(d,Y) \rightarrow (X <> Y),$$
$$P(c,X) \,\&\, P(d,Y) \rightarrow (X <> Y).$$

Для этой системы знаний нужно доказать следующую формулу:

$$\exists S\, \exists T\, \exists Q\, \exists V\, (P(a,S) \& P(b,T) \& P(c,Q) \& P(d,V)).$$

1.3.2. Продукционная модель знаний

Продукционная модель представляет собой вариант логической модели, записанной с помощью продукционных формул (правил) вида $F \to G$, где F,G – произвольные логические формулы. Отмечается, что продукции «более близки» по форме «естественным» умозаключениям. На практике это чрезмерно общее условие можно ограничить, рассматривая формулы так называемого секвенционального вида:

$$f_1 \& f_2 \& \ldots \& f_z \to g_1 \vee g_2 \vee \ldots \vee g_t , \qquad (1.13)$$

где все используемые формулы суть атомарные. Формулу (**1.13**) нетрудно привести к форме дизъюнкта:

$$\neg f_1 \vee \neg f_2 \& \ldots \vee \neg f_z \vee g_1 \vee g_2 \vee \ldots \vee g_t . \qquad (1.14)$$

Система логического программирования Пролог использует еще более упрощенные секвенциальные формулы (так называемую нотацию Хорна):

$$f_1 \& f_2 \& \ldots \& f_z \to g_1. \qquad (1.15)$$

В (**1.15**) часть $f_1 \& f_2 \& \ldots \& f_z$ называется **телом** продукции, а g_1 – **головой**. Продукция без головы называется **целью** (*goal*). Каждая предикатная формула f_i называется в этом случае подцелью. Перейти от (**1.13**) к представлению (**1.14**) просто, если принять во внимание следующую эквивалентность:

$\neg f_1 \vee \neg f_2 \& \ldots \vee \neg f_z \vee g_1 \vee g_2 \vee \ldots \vee g_t \equiv \varnothing g_2 \& \varnothing g_3 \ldots \& \varnothing g_t \& f_1 \& \ldots \& f_z \to g_1.$

Рассмотрим представление знаний в системе программирования Пролог.

Пример 1.3. Владимиру нравится все, что нравится Ире.

нравится(*владимир*, *X*) :–
нравится(*ира*, *X*).

Эта продукция соответствует формуле

нравится(*ира*, *X*) → *нравится*(*владимир*, *X*).

В Прологе тело продукции записывается после символов «:–».

Пример 1.4. Алексей мечтает о деньгах и карьере.
Это предложение определяет два факта:

мечтает(*алексей*, *карьера*).
мечтает(*алексей*, *деньги*).

Факты – это особый вид правил, которые не нужно доказывать. Такие правила, как можно видеть, представляют продукции без «головы».

При программировании на языке Пролог особую трудность вызывает использование рекурсивных продукций, в которых тело продукции содержит предикат, одновременно стоящий в голове продукции. Рассмотрим пример вычисления суммы чисел от 1 до заданного числа *n*.

```
sum(0, X, X).
sum(Z, S, X):–
Z1=Z-1,
S1=S+Z,
sum(Z1,S1,X).
```

Во-первых, заметим, что переменные в языке Пролог пишут с больших букв. Во-вторых, в данном примере назначение аргументов предиката *sum* таково: первый аргумент есть текущее число, второй аргумент есть промежуточная накапливаемая сумма, третий аргумент есть результирующая сумма. Продукция *sum*(0, *X*, *X*) является фактом, утверждающим, что если текущее число равно 0, то промежуточная накапливаемая сумма и результирующая сумма совпадают. Вторая продукция как раз и дает пример рекурсии в Прологе. Эту продукцию следует читать таким образом:

если

(*Z*1 = *Z* - 1) и (*S*1 = *S* + *Z*) и (1 + 2 + 3 +…+ *Z*1 = *S*1),

то

(1 + 2 + 3+**…**+ (*Z*- 1) + *Z* = *S*).

Запись рекурсивных определений подчиняется некоторым общим правилам. Рекурсия должна сводиться к простому факту. Тело рекурсии должно связывать значения аргументов доказываемой формулы с изменяемыми значениями аргументов той же формулы.

Продукционные модели используются в определении формальных языков (грамматик). Например, определение целого числа без знака, допускающего ведущие нули, может иметь такой вид:

```
Цифра :- 0.
Цифра :- 1.
Цифра :- 2.
. . .
Цифра :- 9.
```

Число :- Цифра.
Число :- Цифра Число.

1.3.3. Модель знаний на основе фреймов

Фреймы предложены Марвином Минским. Минский также ввел терминологию и язык фреймов, включающий понятия «фрейма», «слота», «терминала», «значения по умолчанию». Фрейм имеет следующую структуру:

{<имя-фрейма><имя слота$_1$ ><значение слота$_1$>
…
<имя слота$_n$ ><значение слота$_n$>}

В качестве примера рассмотрим фрейм <выбор скорости>:

{<выбор скорости>
<состояние дороги>:**0.6**
<состояние машины>:**0.8**
<состояние водителя>:**0.5**
}

Наша гипотетическая экспертная система должна определять оптимальную скорость автомобиля (в пределах дозволенной) с учетом состояния дороги, машины и водителя. В данном примере уже указаны конкретные значения, так что необходима процедура оценки скорости по конкретным значениям слотов. Такая процедура называется **демоном**. Процедура-демон запускается автоматически, когда фрейм удовлетворяет некоторому образцу. Наряду с процедурами-демонами имеются процедуры-слуги, которые используются для установления значений слотам. Так, для определения состояния дороги можно было использовать следующий фрейм:

{<состояние дороги>
<состояние покрытия>:**0.5**
<видимость>:**1.0**
}

Такие же фреймы могли быть установлены для состояния машины и состояния водителя. Далее, например, для состояния покрытия можно использовать фрейм:

{<состояние покрытия>
<материал покрытия>:**0.4**
<наличие влаги>:**0.0**

<изношенность дороги>:**0.1**
}

Ясно, что организация экспертной оценки скорости требует создать все необходимые фреймы и разработать процедуру комплексной оценки. Хорошей основой для такой процедуры является метод Т. Саати [1].

Модель знаний на основе фреймов является сравнительно простой и легко реализуемой. В ее основу можно положить обычную базу данных с визуальным интерфейсом.

1.3.4. Спецификации и семантические сети

В реальном мире любую ситуацию можно охарактеризовать следующим образом. Во-первых, указать, какие объекты участвуют в ситуации. Во-вторых, определить, в какие отношения вступают объекты. Наконец, указать свойства объектов и свойства отношений. Таким образом, можно передать знания о широком классе ситуаций. Рассмотрим пример. Дано предложение: «Студент Максимов сдал экзамен по химии». В этой ситуации выделяем объекты: **Максимов** и **экзамен**. Отношения между объектами передаются с помощью глаголов. В нашем случае отношением является глагол **сдать**. Свойством Максимова является <**студент**>. Свойством экзамена является <**по химии**>. Свойством отношения <сдать> является время и характер действия (в нашем случае – это прошедшее время). Таким образом, описываемую ситуацию можно было бы представить с помощью следующей спецификации:

Situation : Сессия
Objects: Максимов (студент), экзамен (по химии)
Relation: Сдать[Максимов, Экзамен] (past time)

Спецификацию можно охарактеризовать как описание предметной ситуации на формализованном языке. Спецификацию можно усложнить, если включить в нее *возможные* и *необходимые* действия, тем самым задав возможность развертывания ситуации. Для подобных спецификаций становится актуальной задача: определить последовательность действий, которая могла бы привести к некоторой ситуации. Подобные спецификации строятся в языках моделирования сигналов и схем. Построенную выше спецификацию можно отобразить в форме сети (рис. 1.1).

Рис. 1.1

Объекты, отношения и свойства отображаются на семантической сети различными типами вершин. Ситуации могут образовывать целые сценарии. Например, рассмотрим второй фрагмент: «Экзамен по химии принимал профессор ZZZ». Объединенная семантическая сеть представлена на рис. 1.2.

В современном программировании получили достаточное распространение текстовые документы XML. Так, «содержимое» рис. 1.2 «укладывается» в структуру документа XML следующим образом (русские слова заменены на английские) – рис. 1.3.

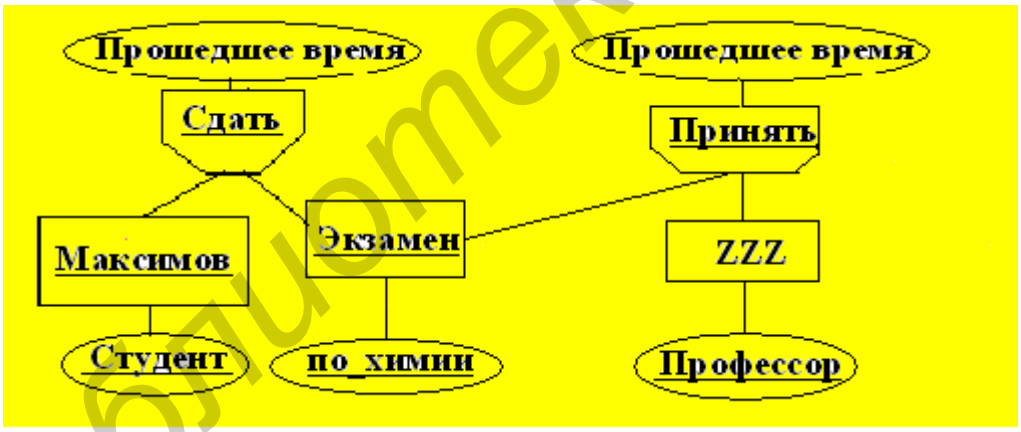

Рис. 1.2

Строки XML-документа представляют собой теги и их значения. Имена тегов связываем с объектами (object), ситуациями (situation) и отношениями (relation).

Идея использования XML-документа может состоять в обработке запросов типа «Кто принимал экзамен по химии у Максимова?»

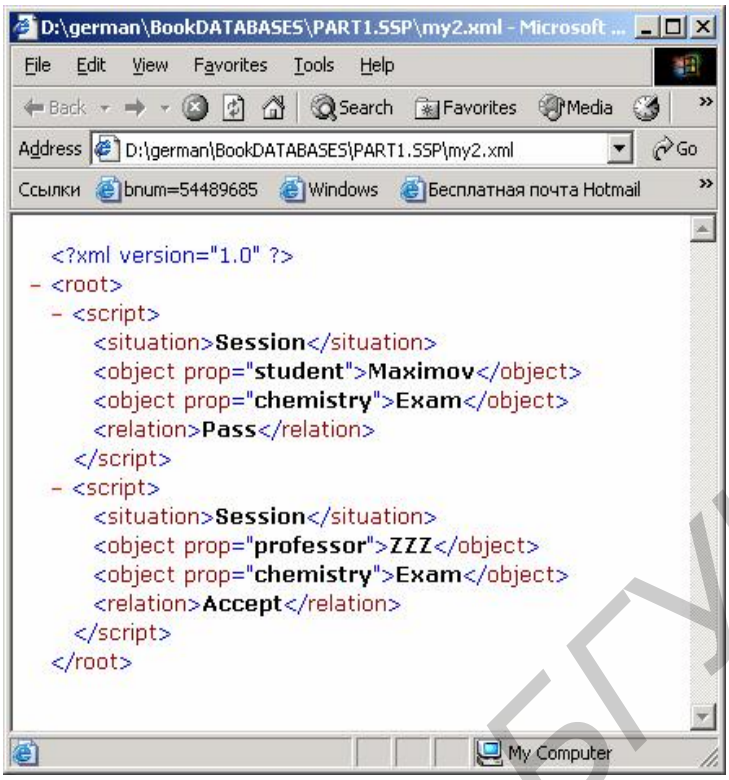

Рис. 1.3

Для реализации этой идеи сам запрос можно перевести в XML-структуру:

<object> Maximov </object>
<object prop=”chemistry”> Exam</object>
<object> ??? </object>
<relation> Accept </relation>

Вопрос представляет фрагмент знаний. Значение (???) подлежит определению. Из контекста вопроса ясно, что это значение участвует в отношении Accept (принять). Вторым объектом данного отношения является ZZZ. Именно это значение и должно быть выдано, поскольку других объектов у отношения Accept нет. Ясно, что ссылка на объект Maximov является наводящей. При обработке запроса система может выйти на связанную ситуацию, определяемую глаголом «Pass» (Сдать), в которой Maximov – действующее лицо. Таким образом, задача системы – выяснить, в какой мере обе ситуации связаны, чтобы считать Maximov существенной характеристикой запроса. Кроме того, при выполнении семантических запросов важно то обстоятельство, что отношения могут включать более двух объектов и не ясно, о каком объекте в запросе идет речь. Эту ситуацию пытаются развязать, используя падежные отношения объектов, передаваемые словами **кто**, **что**, **как**, **какой**, **чем** и пр. Но, опять же, более естественно искать нужный объект не с помощью падежных отношений, а через наводящие характеристики запроса.

1.4. МАШИНА ВЫВОДА

Задача **машины вывода** состоит в «извлечении» ответа на вопрос задачи. Очевидно, для различных моделей знаний требуются различные машины вывода. Так, логическая машина вывода должна строить доказательства (теорем существования решений например). Следует иметь в виду, что для неклассических логик, к которым относятся, например, нечеткая логика или многозначные логики, требуется строить собственную машину вывода, учитывающую специфику исчисления.

Машина вывода для продукционной базы знаний является интерпретатором правил, который использует факты, содержащиеся в базе знаний. На входе машина вывода получает цель консультации.

Машина вывода для табличной базы знаний часто основывается на использовании теоремы Байеса, но может использовать механизм классифицирующего дерева, дискриминантных функций, таблиц решений и поиска по образцу.

Машина вывода может использовать **механизм эвристического поиска** на дереве состояний задачи (которое называют также деревом решения). Рассмотрим пятерку:

$$<\boldsymbol{S_0}, \boldsymbol{S_F}, \boldsymbol{A}, \boldsymbol{R_A}, \boldsymbol{R_S}>, \qquad \textbf{(1.16)}$$

где $\boldsymbol{S_0}$ – исходное состояние системы (объекта, модели); $\boldsymbol{S_F}$ – конечное (искомое) состояние системы (объекта, модели); $\boldsymbol{A}$ – алгоритм отображения $\boldsymbol{A}: \boldsymbol{S_0} \rightarrow \boldsymbol{S_F}$.

Определение 1.9. Состояние $\boldsymbol{S_i}$, непосредственно достижимое из состояния $\boldsymbol{S_k}$, называется преемником $\boldsymbol{S_k}$, а $\boldsymbol{S_k}$ – предшественником $\boldsymbol{S_i}$.

Определение 1.10. Граф $\boldsymbol{\Gamma}(\boldsymbol{S}, p)$ с множеством вершин-состояний $\boldsymbol{S_j} \in \boldsymbol{S}$ и дуг p, связывающих предшественников и преемников, образует дерево решения с корнем $\boldsymbol{S_0}$ и листом $\boldsymbol{S_F}$, если:

- m-й ярус в $\boldsymbol{\Gamma}$ образуют те преемники (m–1)-го яруса (m =1,2,…), которые не входят в ярусы (m–2), (m–3) и т.д.
- листьями в $\boldsymbol{\Gamma}$ являются состояния, все преемники которых содержатся в верхних ярусах (за исключением $\boldsymbol{S_F}$).

Задача машины вывода состоит в отыскании пути (как правило, кратчайшего) из $\boldsymbol{S_0}$ в $\boldsymbol{S_F}$. Любая часть этого пути называется трассой вывода. Для построения трассы вывода на дереве задачи используют механизм оценочных эвристических функций. Примером подобного эвристического алгоритма является алгоритм A^* Нильссона [2]. Качество эвристических оценочных

функций определяется тем, насколько близко создаваемая алгоритмом трасса вывода «держится» к оптимальной; можно использовать, например, такую оценку:

$$K=L/T. \qquad (1.17)$$

Здесь T – общее число вершин, пройденных в дереве решения; L – длина оптимального пути из S_0 в S_F.

2. ЛОГИЧЕСКИЕ МЕТОДЫ РЕШЕНИЯ ЗАДАЧ В ЭКСПЕРТНЫХ СИСТЕМАХ

2.1. ОБЩИЕ ЗАМЕЧАНИЯ

Выше мы показали, что логико-комбинаторные задачи эффективно сводятся к задаче ВЫПОЛНИМОСТЬ (ВЫП). Таким образом, проверка выполнимости системы дизъюнктов является центральной задачей, которую должна уметь решать логическая машина вывода в составе ЭС. Задача ВЫП, несмотря на простую постановку, относится к классу сложных с вычислительной точки зрения задач, для которых до настоящего времени не найден эффективный алгоритм. Рассмотрим некоторые известные методы решения ВЫП.

2.2. ПРИНЦИП РЕЗОЛЮЦИЙ ДЖ. РОБИНСОНА

Имея множество S дизъюнктов логики высказываний, с помощью принципа резолюций всегда можно установить, противоречиво S или нет. Для логики предикатов первого порядка можно гарантированно установить противоречивость S, если S в самом деле противоречиво.

Определение 2.1. Пусть даны два дизъюнкта $D_1 = x_i \vee F$ и $D_2 = \neg x_i \vee R$ (F и R – дизъюнкции литер). Резольвентой D_1 и D_2 является дизъюнкт $F \vee R$.

Литеры $x_i, \neg x_i$ образуют контрарную пару литер, которая из резольвенты исключается.

Пусть S^+ обозначает все множество следствий (резольвент), которые можно получить из S. Пусть $S \subseteq S^+$. Имеет место следующая теорема.

Теорема 2.1 (о полноте метода резолюций). Если S – противоречивое множество формул, то пустая формула $\square \in S^+$ и наоборот.

Доказательство. Приведем доказательство, которое впоследствии будем использовать для реализации одной из важных модификаций метода резолюций. Пусть z_1 , z_2 ,…, z_k , d – дизъюнкты системы S. Возьмем дизъюнкт z_1. Пусть $z_1 = a \vee F$ (где F – дизъюнкция литер, не содержащая a). Построим все возможные резольвенты: z_1 и z_2 , z_1 и z_3 ,…, z_1 и d с отсекаемой парой литер (a, $\neg a$). Если для a нет контрарной литеры ($\neg a$), то резольвенту не строим. Рассмотрим систему: $S' = \{R_{12}, R_{13}, \ldots, R_{1k}, z_2, z_k, \ldots, d\}$, где z_1 не входит в S', а R_{1j} – резольвента (если есть) z_1 и z_j . Докажем, что S и S' эквивалентны в смысле выполнимости: (A) если S выполнима, то выполнима S', (B) если S противоречива, то S' противоречива.

(A) имеет место в силу свойств резольвент.

(B) Пусть S противоречива, а S' нет. Покажем, что это невозможно. Например, пусть S' выполнима в интерпретации I, а S нет. Рассмотрим какую-нибудь резольвенту R_{1j} , для которой в I имеем:

$$z_1 = a \vee F,$$
$$z_j = \neg a \vee D,$$
$$R_{1j} = F \vee D.$$

При этом z_j , R_{1j} истинны в I, а z_1 ложен в I. Тогда a=0, F=0 в I, а D=1. Если принять a=1, то z_1 станет истинным, z_j останется истинным и R_{1j} также останется истинным. Тем самым мы получим интерпретацию I', выполняющую и S, и S' вопреки предположению.

Таким образом, обоснован метод отсечения литер как модификация принципа резолюций: последовательно избавляемся от каждой литеры, пока либо не получим противоречия (пустую формулу), либо не придем к тривиально выполнимой системе. Для иллюстрации рассмотрим пример. Пусть дана система S:

$$\begin{cases} x_1 \vee x_2, \\ x_3 \vee x_4, \\ x_5 \vee x_6, \\ x_7 \vee x_8, \\ x_9, \\ \neg x_4 \vee \neg x_5, \\ \neg x_1 \vee \neg x_3, \\ \neg x_1 \vee \neg x_7, \\ \neg x_2 \vee \neg x_5, \\ \neg x_2 \vee \neg x_9, \\ \neg x_4 \vee \neg x_9, \\ \neg x_6 \vee \neg x_8. \end{cases} \qquad (2.1)$$

Избавимся от литеры x_9. Для этого присоединим к системе все резольвенты пар дизъюнктов с отсекаемой контрарной парой (x_9, $\neg x_9$), а все дизъюнкты, содержащие x_9,$\neg x_9$, исключим из рассмотрения. Новая система S' примет такой вид:

$$
\left\{
\begin{array}{l}
x_1 \vee x_2, \\
x_3 \vee x_4, \\
x_5 \vee x_6, \\
x_7 \vee x_8, \\
//x_9 \text{ исключен}, \\
\neg x_4 \vee \neg x_5, \\
\neg x_1 \vee \neg x_3, \\
\neg x_1 \vee \neg x_7, \\
\neg x_2 \vee \neg x_5, \\
//\neg x_2 \vee \neg x_9 \text{ исключен}, \\
//\neg x_4 \vee \neg x_9 \text{ исключен}, \\
\neg x_2 \quad //\text{добавлен как резольвента } x_9 \text{ и } \neg x_2 \vee \neg x_9, \\
\neg x_4 \quad //\text{добавлен как резольвента } x_9 \text{ и } \neg x_4 \vee \neg x_9, \\
\neg x_6 \vee \neg x_8.
\end{array}
\right. \qquad (2.2)
$$

Избавимся теперь от литеры x_1 . Для этого присоединим к системе все резольвенты пар дизъюнктов с отсекаемой контрарной парой (x_1 , $\neg x_1$), а все дизъюнкты, содержащие x_1, $\neg x_1$, исключим из рассмотрения. Новая система S'' примет такой вид:

$$
\left\{
\begin{array}{l}
//x_1 \vee x_2 \text{ исключен}, \\
x_3 \vee x_4, \\
x_5 \vee x_6, \\
x_7 \vee x_8, \\
//x_9 \text{ исключен}, \\
\neg x_4 \vee \neg x_5, \\
//\neg x_1 \vee \neg x_3 \text{ исключен}, \\
//\neg x_1 \vee \neg x_7 \text{ исключен}, \\
\neg x_2 \vee \neg x_5, \\
//\neg x_2 \vee \neg x_9 \text{ исключен}, \\
//\neg x_4 \vee \neg x_9 \text{ исключен}, \\
\neg x_2 \quad //\text{добавлен как резольвента } x_9 \text{ и } \neg x_2 \vee \neg x_9, \\
\neg x_4 \quad //\text{добавлен как резольвента } x_9 \text{ и } \neg x_4 \vee \neg x_9, \\
\neg x_6 \vee \neg x_8, \\
x_2 \vee \neg x_3 \text{ //резольвента } x_1 \vee x_2 \text{ и } \neg x_1 \vee \neg x_3, \\
x_2 \vee \neg x_7 \text{ //резольвента } x_1 \vee x_2 \text{ и } \neg x_1 \vee \neg x_7.
\end{array}
\right. \qquad (2.3)
$$

Продолжая процесс в том же духе, избавимся от литеры x_3. Получим следующую систему:

$$\begin{cases} x_5 \vee x_6, \\ x_7 \vee x_8, \\ \neg x_4 \vee \neg x_5, \\ \neg x_2 \vee \neg x_5, \\ \neg x_2, \\ \neg x_4, \\ \neg x_6 \vee \neg x_8, \\ x_2 \vee x_4 \text{ //добавлен}, \\ x_2 \vee \neg x_7. \end{cases} \quad \textbf{(2.4)}$$

Избавимся теперь от x_2:

$$\begin{cases} x_5 \vee x_6, \\ x_7 \vee x_8, \\ \neg x_4 \vee \neg x_5, \\ \neg x_5, \\ \neg x_7, \\ \neg x_4, \\ \neg x_6 \vee \neg x_8, \\ x_4. \end{cases} \quad \textbf{(2.5)}$$

Система (**2.5**) является противоречивой (содержит контрарную пару однолитерных дизъюнктов: $\neg x_4$ и x_4). Процесс доказательства завершается.

Имеются другие модификации принципа резолюций: линейная входная резолюция, лок-резолюция, метод семантической резолюции Слэйгла, упорядоченная резолюция (OL-резолюция) и пр. У них всех одна общая проблема: экспоненциальные от размера задачи в общем случае временные потери. Эта проблема напрямую касается эффективности логического программирования как такового.

2.3. МЕТОД Р. КОВАЛЬСКОГО

Метод Р. Ковальского [3] использует представление системы дизъюнктов в форме графа. Каждому дизъюнкту соответствует некоторая вершина. Ребро соединяет любые две контрарные литеры. Рассмотрим следующий пример системы дизъюнктов *S* (рис. 2.1):

$$\begin{cases} D1 = x_1 \vee \neg x_2, \\ D2 = x_2 \vee x_3, \\ D3 = \neg x_1 \vee x_2 \vee x_3, \\ D4 = \neg x_2 \vee \neg x_3. \end{cases}$$

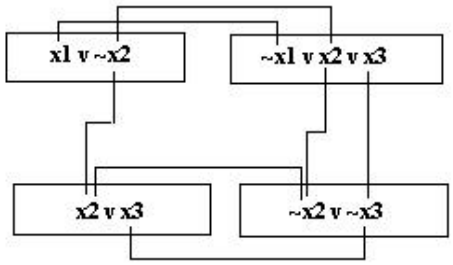

Рис. 2.1

Резольвенты строим таким образом. Выбираем любое ребро, например связывающее литеры $x_1 \vee \neg x_1$. Строим резольвенту дизъюнктов $D1$ и $D3$ по контрарной паре литер x_1, $\neg x_1$. Добавляем резольвенту в форме новой вершины графа, а ребро $(x_1, \neg x_1)$ удаляем (рис. 2.2). При этом действуют следующие правила (**П1**–**П4**).

П1. Если в вершине имеется литера, которая не связана ребром, то такая вершина удаляется. (В нашем примере удаляем перечеркнутые вершины, поскольку литера x_1, $\neg x_1$ стала несвязанной.)

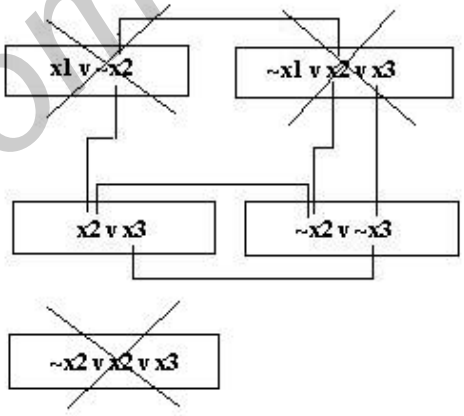

Рис. 2.2

П2. Тавтологические дизъюнкты удаляем.

(На основании этих правил получим рис. 2.3.)

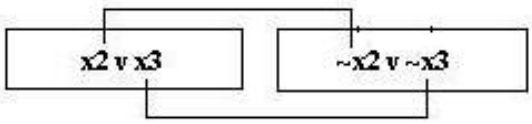

Рис. 2.3

П3. Вершины-резольвенты связываются связями с другими вершинами по контрарным литерам.

П4. Если в результате применения предыдущих правил в графе не останется вершин, то система **не противоречива**. Если система противоречива, то на каком-то этапе вывода получим две контрарные однолитерные вершины.

(В нашем случае система оказывается непротиворечивой.)

2.4. МЕТОД ГРУППОВЫХ РЕЗОЛЮЦИЙ

Метод групповых резолюций [4] позволяет найти решение задачи о минимальном покрытии 0,1-матрицы B множеством строк. Пусть дана система дизъюнктов из подразд. 2.3. Для нее строим 0,1-матрицу B следующим образом. Строки матрицы соответствуют литерам x_i и их отрицаниям $\neg x_i$. Таким образом, число строк составит $2n$, где n – число различных булевских переменных. Столбцы матрицы соответствуют дизъюнктам системы Dj , причем столбец содержит единицу в строке x_k ($\neg x_k$), если переменная x_k входит в Dj без отрицания (с отрицанием). Кроме того, матрица дополнительно содержит n столбцов, соответствующих тавтологическим дизъюнктам $x_k \vee \neg x_k$. С учетом сказанного, матрица B для рассматриваемого примера имеет вид, изображенный на рис. 2.4.

	$D1$	$D2$	$D3$	$D4$	$D5$	$D6$	$D7$
x_1	1				1		
x_2		1	1			1	
x_3		1	1				1
$\neg x_1$			1		1		
$\neg x_2$	1			1		1	
$\neg x_3$				1			1

Рис. 2.4

Определение 2.2. Под *минимальным покрытием* матрицы B понимается минимальное по числу строк множество p_{min}, такое, что для каждого столбца

матрицы B найдется как минимум одна строка в $p_{\min}$, которая содержит в данном столбце «1» (иными словами, покрывает данный столбец).

Утверждение. Пусть дана матрица B, построенная для системы дизъюнктов с $n>1$ переменными. Если минимальное покрытие $p_{\min}$ матрицы B содержит более n строк, то данная система дизъюнктов невыполнима; в противном случае – выполнима.

Принцип групповых резолюций (ПГР) позволяет порождать новые – групповые резольвенты, используя любой эвристический метод для отыскания минимального или близкого к нему покрытия. Гарантируется, что рано или поздно будет порожден полностью нулевой столбец. В этом случае алгоритм прекращает работу. Наилучшее из найденных к этому моменту покрытий и является минимальным.

В качестве эвристического алгоритма можно использовать следующий. На каждой итерации p отыскиваем столбец (из числа невычеркнутых) с минимальным числом единиц. Обозначим этот столбец r_p и назовем его **синдромным**. Найдем невычеркнутую строку i_p, покрывающую r_p и такую, что из всех других строк, покрывающих r_p, она содержит наибольшее число единиц. Включим i_p в отыскиваемое на данной итерации p покрытие. Вычеркнем все строки, содержащие в столбце r_p «1», а также все столбцы, покрываемые строкой i_p. Итерация ведется до тех пор, пока имеется хотя бы один невычеркнутый столбец и одна невычеркнутая строка.

Так, выберем столбец $D1$ и строку $\neg x_2$, которую включим в отыскиваемое покрытие на итерации 1. Вычеркнем строки и столбцы согласно описанному правилу (рис. 2.5).

	$D2$	$D3$	$D5$	$D7$
x_1			1	
x_2	1	1		
x_3	1	1		1
$\neg x_1$		1	1	
$\neg x_3$				1

Рис. 2.5

Выполним теперь вторую итерацию. Выберем столбец $D2$ и строку x_3. Вычеркнем строки и столбцы согласно описанному правилу (рис. 2.6). Остается единственный невычеркнутый столбец, так что включим, например, строку x_1 в предполагаемое минимальное покрытие. Таким образом, итерация завершается отысканием покрытия $p_1 = \{\neg x_2, x_3, x_1\}$. Согласно представленному выше утверждению, данное покрытие минимально и дает выполняющую интерпретацию для исходной системы дизъюнктов.

	$D5$
x_1	1
x_2	
x_3	
$\neg\neg x_1$	1
$\neg\neg x_3$	

Рис. 2.6

Описанный эвристический алгоритм не всегда отыскивает минимальное покрытие, и необходимо выполнять этап построения групповой резольвенты. Это делается так. Берем текущее найденное покрытие и оставляем в нем любые n+1 строк. Формируем матрицу R из *синдромных* столбцов, найденных для этих строк. Формируем столбец-резольвенту, исходя из следующего: он содержит «1» в тех и только тех строках, которые в R имеют две или более единиц; в противном случае строка содержит «0». Этот столбец присоединяется к матрице B, и итерации возобновляются снова (все вычеркнутые строки и столбцы восстанавливаем на новой итерации). Так, найденное покрытие $p_1 = \{\neg x_2, x_3, x_1\}$ содержит ровно $n = 3$ строки. Тем не менее, построим для него матрицу R (рис. 2.7) на синдромных столбцах $D1$, $D2$, $D5$.

	$D1$	$D2$	$D5$	Резольвента
x_1	1		1	1
x_2		1		
x_3		1		
$\neg\neg x_1$			1	
$\neg\neg x_2$	1			
$\neg\neg x_3$				

Рис. 2.7

В данном случае групповая резольвента содержит единственную «1» в строке x_1. Поэтому при возобновлении итераций (если бы это было необходимо) следовало добавить данную резольвенту к матрице B.

Недостатком описанного метода является рост размеров матрицы при присоединении новых резольвент. Существенное усиление этого метода дано в подразд. 3.4.

3. ЛОГИЧЕСКОЕ ПРОГРАММИРОВАНИЕ, АЛЬТЕРНАТИВНОЕ ЯЗЫКУ ПРОЛОГ

3.1. ОПИСАНИЕ ПОДХОДА

Концепция Пролога (ПРОграммирования в ЛОГике), предложенная Робертом Ковальским, базировалась на процедурной интерпретации логических исчислений в форме Хорна. Главный недостаток логических исчислений – отсутствие эффективной в вычислительном смысле процедуры вывода – в Прологе усугублен еще более: процедура вывода последовательно перебирает варианты, сохраняя в памяти все дерево вывода, включая все его бесполезные части, что представляется не лучшим техническим решением.

С современной позиции можно искать возможность для поднятия уровня логических формализаций в программировании (в том числе ЭС), взяв за основу более прагматичную концепцию. Современное программирование может естественным образом развиваться как технология решения задач – как задачно-ориентированное программирование. В этом смысле логическое программирование дает не отдельную парадигму, а дополняет универсальный подход к программированию в части постановки задач и реализации методов их решения. Перебор можно значительно сократить механизмом угадывания решения, который, с одной стороны, легко интегрируется в методы типа отсечений и ветвей и границ, а с другой – при решении задач следует использовать не одну, а множество эвристик в разных сочетаниях и в разных контекстах задачи. Важно, чтобы на каждой итерации алгоритма эвристическая процедура «проходила по новой траектории».

В связи с этим покажем, как строить логический вывод в логике предикатов путем последовательного решения индивидуальных задач ВЫП, которые строятся для текущих систем дизъюнктов, на идеях, отличных от эрбрановских.

Обратимся снова к задаче о минимальном покрытии (ЗМП) 0,1-матрицы множеством строк. Пусть B – 0,1-матрица с $n > 0$ строками и $m > 0$ столбцами.

Определение 3.1. Множество строк p матрицы B называется неизбыточным покрытием матрицы B, если (А) для каждого столбца матрицы B в p найдется строка, которая его покрывает, и при этом (Б) ни одну строку нельзя удалить из p без потери выполнимости условия (А).

Поиск минимального покрытия можно связать с последовательным поиском покрытий с заданным числом строк: k, $k-1$, $k-2$, …, $k_{\min}$.

3.2. АНАЛИЗ ИДЕЙ КОВАЛЬСКОГО–ЭРБРАНА

Для проверки противоречивости системы дизъюнктов, построенных с помощью предикатов (литералов), процедура, предложенная Ж. Эрбраном [3], последовательно строит системы дизъюнктов, записанные с помощью булевских литер (переменных), для которых решаются соответствующие задачи ВЫП. Очевидная неэффективность процедуры Эрбрана состоит в том, что порождаемые задачи ВЫП могут содержать значительное и не ограничиваемое сверху в общем случае число избыточных дизъюнктов, для записи которых используются все константы исходной системы, затем – новые константы, получаемые из функторов, и т.д.

С другой стороны, основная проблема метода резолюций [3,4] остается той же – построение бесполезных (не необходимых) резольвент, число которых может быть также значительным.

В методе графа соединений Р. Ковальского последовательно выводятся из системы дизъюнкты, если вершины графа соединений, соответствующие этим дизъюнктам, содержат литералы, не связанные ребрами с другими вершинами графа. Таким образом, метод графа соединений Ковальского в определенной мере противодействует росту числа дизъюнктов системы.

Существенный недостаток методов резолюционного типа, к числу которых относится и метод Р. Ковальского, состоит главным образом в том, что они решают задачу ВЫП далеко не лучшим образом.

В пособии мы объединяем эти два подхода; достигаемое техническое преимущество состоит в следующем:

- на каждом итерационном шаге решается задача ВЫП (так же, как это делается при методе Эрбрана), для которой имеется широкий арсенал методов [5,6];
- в отличие от процедуры Эрбрана, системы дизъюнктов, образующие задачи ВЫП, формируются «более узким фронтом» из числа необходимых для установления противоречивости.

3.3. ПРИМЕР И ОПРЕДЕЛЕНИЯ

Рассматриваем следующий пример системы дизъюнктов в качестве рабочей иллюстрации:

$$\begin{cases} \text{(a1)} \quad P(x,a) \vee Q(y), \\ \text{(a2)} \ \neg P(a,z) \vee Q(b), \\ \text{(a3)} \ \neg Q(t) \vee C(t), \\ \text{(a4)} \ \neg Q(b) \vee \neg C(b). \end{cases} \qquad (\mathbf{3.1})$$

Замечание. Везде далее принимается, что в представлении дизъюнктов опущена префиксная часть, так что, например, (а1) следует читать так:

$$\forall x \forall y\, (P(x,a) \vee Q(y)).$$

В таком представлении переменные x и y называются **связанными**; далее опускаем слово «связанные» и говорим просто про переменные литерала.

Определение 3.2. Пусть дан дизъюнкт $D = L_1(\ldots) \vee L_2(\ldots) \vee \ldots \vee L_z(\ldots)$. Говорим, что литералы L_i и L_j из D **разделены**, если одновременно истинно то, что:

- они не содержат общих переменных;
- эти литералы не одноименны.

В (а1) $P(x,a)$, $Q(y)$ разделены. В (а3) $\neg Q(t)$, $C(t)$ не разделены.

Определение 3.3. Литерал L_i из D, разделенный с каждым другим литералом из дизъюнкта D, называется **независимым** (в D); в противном случае – **зависимым**.

Литерал $C(t)$ зависим в (а3).

Определение 3.4. Пусть даны два контрарных унифицируемых литерала $L_i \in D_a$ и $L_m \in D_b$. Пусть каждый из них независим в своем дизъюнкте. Тогда пара литералов L_i, L_m называется **несовместной**.
Литералы $P(x,a)$ из (а1) и $\neg P(a,z)$ из (а2) несовместны.

Сопоставим с каждым литералом L_i исходной системы дизъюнктов S булевскую переменную x_i. Назовем x_i представляющей переменной. Запишем для S задачу ВЫП с помощью представляющих переменных следующим образом.

1. Дизъюнкты из S записываются с помощью представляющих переменных.

2. Добавляем новые двухлитерные дизъюнкты вида $\neg x_k \vee \neg x_m$, если и только если L_k и L_m несовместны.

Для системы (**3.1**) получим следующую задачу ВЫП:

$$\begin{cases} x_1 \vee x_2, \\ x_3 \vee x_4, \\ x_5 \vee x_6, \\ x_7 \vee x_8, \\ \neg x_1 \vee \neg x_3, \\ \neg x_2 \vee \neg x_7, \\ \neg x_4 \vee \neg x_7. \end{cases} \qquad (\textbf{3.2})$$

($x_1 \leftrightarrow P(x,a)$; $x_2 \leftrightarrow Q(y)$; $x_3 \leftrightarrow \sim P(a,z)$; $x_4 \leftrightarrow Q(b)$; $x_5 \leftrightarrow \neg Q(t)$; $x_6 \leftrightarrow C(t)$; $x_7 \leftrightarrow \neg Q(b)$; $x_8 \leftrightarrow \neg C(b)$.)

Построенную задачу ВЫП для S назовем **характеризационной**.

Теорема 3.1. Если характеризационная задача ВЫП для S не имеет допустимого решения, то S противоречива.

Доказательство. Если S не противоречива, то она имеет модель M, в которой все дизъюнкты истинны [7]. Пусть $L_p \in D_a$ и L_p – независимый литерал для D_a. Тогда D_a можно переписать таким образом:

$$D_a = \forall w_1 \ldots \forall w_z \ (L_1(\ldots) \vee L_2(\ldots) \vee \forall u_1 \ldots \forall u_q \ L_p(u_1, \ldots, u_q),$$

где ни одна переменная из $u_1, \ldots, u_q$ не содержится среди переменных $w_1, \ldots, w_z$. Но тогда $\forall u_1 \ldots \forall u_q \, L_p(u_1, \ldots, u_q)$ есть высказывание (истинное или ложное), так что его можно заменить представляющей переменной x_p. Следовательно, в M ни один из характеризационных дизъюнктов вида $\neg x_p \vee \neg x_r$ не может быть ложен.

Описание метода вывода

Найдем решение системы (**3.2**). Если система (**3.2**) невыполнима, то исходная система дизъюнктов в силу теоремы 3.1 противоречива. Одно из решений (выполняющих интерпретаций) есть: $I=<x_1, x_2, \neg x_3, x_4, x_5, \neg x_6, \neg x7, x_8>$. Выпишем те предикаты (литералы) из системы (**3.1**), которым соответствуют позитивные (вошедшие в I без отрицания) представляющие литеры, причем из каждого дизъюнкта выпишем ровно один (произвольно) такой предикат, например,

$$\begin{array}{ll} P(x,a) & (a1), \\ Q(b) & (a2), \\ \neg Q(t) & (a3), \\ \neg C(b) & (a4). \end{array} \qquad (\mathbf{3.3})$$

Назовем множество выписанных литералов **кандидатом** в решение.

Теорема 3.2. Если кандидат в решение не содержит унифицируемой контрарной пары литералов, то исходная система дизъюнктов выполнима (имеет модель).

Доказательство. Утверждение теоремы просто констатирует наличие модели, каковой является выписанный кандидат в решение.

Пусть, напротив, кандидат в решение содержит унифицируемую контрарную пару литералов. Ясно, что в этом случае он не может служить моделью и должен быть исключен. Оказывается, что для этой цели можно найти обычную резольвенту и присоединить ее к системе вместе с одним дополнительным характеризационным дизъюнктом (для присоединения которого оговариваются условия ниже). Именно, найдем любую унифицируемую контрарную пару литералов в (**3.3**), например $Q(b)$ и $\neg Q(t)$, и построим резольвенту дизъюнктов (a2) и (a3) из (**3.1**), которые содержат эти литералы. Эта резольвента такова: $\neg P(a,z) \vee C(b)$. Присоединим эту резольвенту к исходной системе (**3.1**):

$$\begin{cases} \text{(a1)} & P(x,a) \vee Q(y), \\ \text{(a2)} & \neg P(a,z) \vee Q(b), \\ \text{(a3)} & \neg Q(t) \vee C(t), \\ \text{(a4)} & \neg Q(b) \vee \neg C(b), \\ \text{(a5)} & \neg P(a,z) \vee C(b). \end{cases} \qquad (\mathbf{3.4})$$

($x_1 \leftrightarrow P(x,a)$; $x_2 \leftrightarrow Q(y)$; $x_3 \leftrightarrow \neg P(a,z)$; $x_4 \leftrightarrow Q(b)$; $x_5 \leftrightarrow \neg Q(t)$; $x_6 \leftrightarrow C(t)$; $x_7 \leftrightarrow \neg Q(b)$; $x_8 \leftrightarrow \neg C(b)$; $x_9 \leftrightarrow \neg P(a,z)$; $x_{10} \leftrightarrow C(b)$.)

Строим характеризационную систему дизъюнктов и присоединяем к ней новый дополнительный дизъюнкт, отсекающий найденное ранее решение: $\neg x_1 \vee \neg x_2 \vee \neg x_3 \vee \neg x_4 \vee \neg x_5 \vee x_6 \vee x_7 \vee x_8$:

$$\begin{cases} x_1 \vee x_2, \\ x_3 \vee x_4, \\ x_5 \vee x_6, \\ x_7 \vee x_8, \\ x_9 \vee x_{10}, \\ \neg x_1 \vee \neg x_2 \vee \neg x_3 \vee \neg x_4 \vee \neg x_5 \vee x_6 \vee x_7 \vee x_8 \text{ //присоединен}, \\ \neg x_1 \vee \neg x_3, \\ \neg x_2 \vee \neg x_7, \\ \neg x_4 \vee \neg x_7, \\ \neg x_1 \vee \neg x_9, \\ \neg x_8 \vee \neg x_{10}. \end{cases} \qquad (\mathbf{3.5})$$

Дополнительный характеризационный дизъюнкт и сопутствующие преобразования системы вводятся согласно следующим правилам.

• Определяем, какие из контрарных литералов, исключенные из резольвенты, являются **зависимыми.** Заметим, что оба они не могут быть одновременно независимыми в силу правил построения характеризационной задачи ВЫП. Пусть L_p – один из таких зависимых литералов. Пусть, кроме того, $L_p \in D_a$.

• Если для L_p в системе нет другого дизъюнкта D_b, содержащего контрарный и унифицируемый с L_p литерал L_w такой, что резольвента с отсекаемой парой $<L_p, L_w>$ дизъюнктов D_a и D_b еще не строилась, то удаляем из системы дизъюнкт D_a и никакого дополнительного характеризационного дизъюнкта не строим. Данная операция применяется в алгоритме Ковальского для удаления вершин-дизъюнктов с висячими литералами. Этот пункт следует проверить и выполнить также на втором из исключенных литералов, если он зависим.

• Если ни один из дизъюнктов D_a и D_b не был исключен в предыдущем пункте, то присоединяем к системе дополнительный характеризационный дизъюнкт, исключающий найденный кандидат в решение.

В нашем примере литерал $Q(b)$, отсеченный в резольвенте, независим, а литерал $\neg Q(t)$ зависим и для него имеется другой контрарный и унифицируемый с ним литерал $Q(y)$ из (a1), причем резольвента для этой контрарной пары еще не строилась. Значит, дополнительный характеризационный дизъюнкт можно присоединить к системе, что мы и сделали.

Выполняющая интерпретация для (**3.5**):

$$I = <\neg x_1, x_2, x_3, x_4, \neg x_5, x_6, \neg x7, x_8, x_9, \neg x_{10}>.$$

Получаем новый кандидат в решение:

$$Q(y), \neg P(a,z), C(t), \neg C(b). \qquad (\mathbf{3.6})$$

В (**3.6**) имеем унифицируемую контрарную пару литералов: $C(t)$, $\neg C(b)$. Найдем резольвенту содержащих их дизъюнктов с этими отсекаемыми литералами: $\neg Q(b)$. Присоединяем резольвенту к системе:

$$\begin{cases} \text{(a1)} & P(x,a) \vee Q(y), \\ \text{(a2)} & \neg P(a,z) \vee Q(b), \\ \text{(a3)} & \neg Q(t) \vee C(t), \\ \text{(a4)} & \neg Q(b) \vee \neg C(b), \\ \text{(a5)} & \neg P(a,z) \vee C(b), \\ \text{(a6)} & \neg Q(b). \end{cases} \qquad (\mathbf{3.7})$$

Дизъюнкт (a3) теперь следует исключить согласно правилам: литерал $C(t)$ является зависимым и не имеет контрарного и унифицируемого с ним литерала из другого дизъюнкта, с которым (a3) еще не давал резольвент. В графе соединений Ковальского такому литералу не инцидентно ни одно ребро. Следовательно, система (**3.7**) перепишется таким образом:

$$\begin{cases} \text{(a1)} & P(x,a) \vee Q(y), \\ \text{(a2)} & \neg P(a,z) \vee Q(b), \\ \text{(a4)} & \neg Q(b) \vee \neg C(b), \\ \text{(a5)} & \neg P(a,z) \vee C(b), \\ \text{(a6)} & \neg Q(b). \end{cases} \qquad (\mathbf{3.8})$$

Строим новую характеризационную задачу ВЫП для системы (**3.8**):

$$\begin{cases} x_1 \vee x_2, \\ x_3 \vee x_4, \\ x_5 \vee x_6, \\ x_7 \vee x_8, \\ x_9, \\ \neg x_4 \vee \neg x_5, \\ \neg x_1 \vee \neg x_3, \\ \neg x_1 \vee \neg x_7, \\ \neg x_2 \vee \neg x_5, \\ \neg x_2 \vee \neg x_9, \\ \neg x_4 \vee \neg x_9, \\ \neg x_6 \vee \neg x_8. \end{cases} \qquad (\mathbf{3.9})$$

(Здесь соответствия таковы: ($x_1 \leftrightarrow P(x,a)$; $x_2 \leftrightarrow Q(y)$; $x_3 \leftrightarrow \neg P(a,z)$; $x_4 \leftrightarrow Q(b)$; $x_5 \leftrightarrow \neg Q(b)$; $x_6 \leftrightarrow \neg C(b)$; $x_7 \leftrightarrow \neg P(a,z)$; $x_8 \leftrightarrow C(b)$; $x_9 \leftrightarrow \neg Q(b)$.)

Система (**3.9**) противоречива. В силу теоремы 3.1 делаем вывод о противоречивости исходной системы дизъюнктов (**3.1**).

Резюмируем особенности предложенной процедуры вывода.

Во-первых, множество характеризационных дизъюнктов задачи ВЫП существенно уже, чем множество основных примеров дизъюнктов Эрбрана.

Во-вторых, в сравнении с процедурой Ковальского не строятся резольвенты с отсекаемыми парами независимых контрарных литералов – известные методы резолюционного типа не проводят различий между независимыми и зависимыми литералами. Предложенный метод нацелен на сведение задачи вывода в логике предикатов на решение задачи ВЫП, что дает шансы повысить скорость вывода, используя для решения ВЫП весь арсенал имеющихся средств.

Утверждение. Если исходное множество дизъюнктов противоречиво, то приведенная стратегия вывода финитна.

Изложенная стратегия отличается от стратегии вывода Ковальского тем, что независимые контрарные унифицируемые литералы не участвуют в образовании резольвент. Следовательно, дизъюнкты, содержащие только независимые литералы, не могут быть исключены из графа соединений Ковальского. Но если все дизъюнкты состоят из одних только независимых литералов, то решение вопроса о выполнимости осуществляется на одной-единственной характеризационной задаче ВЫП, иначе процесс вывода повторяет процедуру Ковальского.

3.4. УСИЛЕННЫЙ МЕТОД ГРУППОВЫХ РЕЗОЛЮЦИЙ ДЛЯ ЗАДАЧИ ВЫП

Усовершенствование метода групповых резолюций заключено в следующих двух существенных пунктах:

I. Количество добавляемых к исходной матрице резольвент (дополнительных столбцов) ограничивается сверху значением числа строк матрицы.

II. Достигаемое лучшее качество порождаемых резольвент сокращает время работы алгоритма.

Предлагаемый алгоритм решает задачу ВЫП сведением ее к ЗМП. Решение ЗМП основано на модификации алгоритма на базе принципа групповых резолюций. Важной частью этой модификации является то, что на заключительной фазе алгоритма выполняется поиск точного решения на значительно сокращенной части исходной 0,1-матрицы. Таким образом, получаемые на каждой итерации покрытия состоят из двух частей. Первая соответствует эвристически отобранным строкам, вторая часть – точному решению, найденному на редуцированной матрице. Для отыскания точного решения можно использовать сам же принцип групповых резолюций либо описываемую далее частично определенную процедуру детерминированной полиномиальной проверки выполнимости системы дизъюнктов (ДПВЫП), которая либо сходится к ответу за полиномиальное время, либо завершается принудительно, но без ответа. Основная идея использования ДПВЫП в том, чтобы при получении ответа строить более сильную резольвенту (т.е. дизъюнкт с меньшим числом литер), чем при отсутствии ответа.

3.4.1. Детерминированная полиномиальная процедура проверки выполнимости

Пусть A и B – пропозициональные формулы. В силу законов вероятностной логики [8] можно записать:

$$P(A \vee B) = P(A)+P(B)-P(AB) = P(A)+P(AB)+P(\text{Ø}AB)- P(AB) = P(A)+P(\text{Ø}AB). \quad (\mathbf{3.10})$$

Здесь $P(A)$ – вероятность того, что произвольная интерпретация I выполняет формулу A Далее, если $A\&B=false$, то $P(A \vee B) = P(A)+P(B)$. В этом случае A и B называют ортогональными. Обобщением (**3.10**) служит

$$P(A \vee B \vee \ldots \vee W) = P(A)+ P(\text{Ø}AB)+ P(\text{Ø}A\text{Ø}BC)+\ldots+ P(\text{Ø}A\text{Ø}B\text{Ø}C\ldots W). \quad (\mathbf{3.11})$$

Ясно, что если формула A тождественно истинна, то $P(A) = 1$. Если A и B ортогональны, то

$$P(\text{Ø}AB) = P(B). \quad (\mathbf{3.12})$$

Наконец, воспользуемся тем, что для $A= x_{k1}^{z1} \vee x_{k2}^{z2} \vee \ldots \vee x_{kr}^{zr}$ $P(A)=1-2^{-r}$, а для $A= x_{k1}^{z1} \,\&\, x_{k2}^{z2} \,\&\, \ldots \,\&\, x_{kr}^{zr}$ $P(A)=2^{-r}$.

Так, для системы (**3.2**) найдем

$P((\mathbf{3.2}))= 1- [P(\text{Ø}x_1\text{Ø}x_2) + P((x_1 \vee x_2)\, \text{Ø}x_3\text{Ø}x_4)+ P((x_1 \vee x_2)\,(x_3 \vee x_4)\, \text{Ø}x_5\text{Ø}x_6) + P((x_1 \vee \vee x_2)\,(x_3 \vee x_4)(x_5 \vee x_6)\, \text{Ø}x_7\, \text{Ø}x_8\,) + P((x_5 \vee x_6)\,(x_7 \vee x_8)\, x_1x_3) + P((x_3 \vee x_4)\,(x_5 \vee x_6)\times \times(\text{Ø}x_1 \vee \text{Ø}x_3)\, x_2x_7) + P((x_1 \vee x_2)\,(x_5 \vee x_6)\,(\text{Ø}x_1 \vee \text{Ø}x_3)(\text{Ø}x_2 \vee \text{Ø}x_7)\;\; x_4x_7)] = 1 -[1/4 + +3/16+ 9/64+27/256 +9/64+ P((x_3 \vee x_4)\,(x_5 \vee x_6)\,(\text{Ø}x_1 \vee \text{Ø}x_3)\, x_2x_7) + P(\,(x_5 \vee x_6)\times \times\text{Ø}x_3x_1\text{Ø}x_2\;\, x_4x_7)]=1 -[1/4 + 3/16+ 9/64+27/256 +9/64+ P((x_3 \vee x_4)(x_5 \vee x_6)(\text{Ø}x_1 \vee \vee\text{Ø}x_3)\, x_2x_7) +3/128].$

Далее, по аналогии, будем иметь

$P((x_3 \vee x_4)\,(x_5 \vee x_6)\,(\text{Ø}x_1 \vee \text{Ø}x_3)\, x_2x_7)=P((x_3 \vee x_4)(x_5 \vee x_6)\text{Ø}x_1\, x_2x_7) \vee(x_3 \vee x_4)\,(x_5 \vee x_6)\times \times\text{Ø}x_3\, x_2x_7)\,)= P((x_3 \vee x_4)\,(x_5 \vee x_6)\,\text{Ø}x_1\, x_2x_7)+P((\text{Ø}x_3\text{Ø}x_4 \vee \text{Ø}x_5\text{Ø}x_6 \vee x_1 \vee \text{Ø}x_2 \vee\text{Ø}x_7)(x_5\vee \vee x_6)\,\text{Ø}x_3x_4x_2x_7)\,)=9/128+ P(x_1\,(x_5 \vee x_6)\,\text{Ø}x_3x_4x_2x_7)= 9/128+3/128=12/128.$

Отсюда окончательно:

$P((\mathbf{3.2}))= 1-[1/4 + 3/16+ 9/64+27/256 +9/64+12/128+3/128]=15/256.$

Из этого следует, что система (**3.2**) не противоречива (вероятность ее выполнения для произвольной интерпретации не нулевая). Важно заметить, что

полиномиальность процедуры вычисления вероятности выполнимости системы дизъюнктов будет гарантированной, если отказаться от тотальной определенности ДПВЫП и ограничить число выполняемых ею шагов полиномом. Простой способ осуществления этого состоит в следующем. Из представления (**3.11**) следует, что вычисление каждого слагаемого в правой части вида $P((\varnothing A \varnothing B \varnothing C \ldots \varnothing R)$ можно заменить вычислением $1\text{-}P(A' \vee B' \vee C' \vee \ldots \vee R')$, где $A', B', C', \ldots, R'$ получены соответственно из $A, B, C, \ldots, R$ с помощью следующих правил:

1. $x_i Y(x_i \vee Z) = x_i Y$;
2. $x_i Y(\sim x_i \vee Z) = x_i YZ$.

Следовательно, вычисление $P(\varnothing A \varnothing B \varnothing C \ldots W)$ реализуется рекурсивно, причем на каждом шаге рекурсии пропадает как минимум одна переменная, что гарантирует конечность процедуры. Остается лишь ограничить глубину рекурсии, например h=3, так что если за 3 шага рекурсии вероятность не будет вычислена, то процедура ДПВЫП завершается без результата.

3.4.2. Решение ЗМП

В проведенных экспериментах с помощью улучшенного алгоритма удалось найти решение выполнимых задач ВЫП, например, с 40–60 переменными и 200–300 дизъюнктами за 0,1–1,5 мин на ПЭВМ с частотой работы 1,7 МГц. Достаточно редкие экземпляры, впрочем, в эти временные рамки не попадали. При этом получаем результаты для самых «тяжелых» задач; это задачи с крайне низким процентом выполняющих интерпретаций от общего числа наборов. Задачи, не относящиеся к категории «тяжелых», решались почти мгновенно.

Решение ЗМП выполняется на основании ранее изложенного принципа групповых резолюций, однако новые столбцы-резольвенты в описываемом здесь методе накладываются на некоторые из ранее присоединенных столбцов-резольвент, поэтому число дополнительно присоединенных столбцов-резольвент не превосходит числа строк матрицы B. Остается подключить к этой общей схеме процедуру ДПВЫП, чтобы еще более усилить качество порождаемых столбцов-резольвент (качество в этом контексте равносильно возможному минимуму числа единиц в столбце-резольвенте).

Рассмотрим суть итерации алгоритма. Каждая итерация алгоритма состоит в многократном выполнении одних и тех же шагов до тех пор, пока в текущей матрице B не останется невычеркнутых столбцов. Именно:

- Находим столбец r из числа невычеркнутых столбцов с минимальным числом «1». Этот столбец является **синдромным** для строки a, определяемой на следующем шаге.
- Находим строку α из числа невычеркнутых строк, покрывающих столбец r, с максимальным числом «1».
- Редуцируем матрицу B следующим образом:
  - Вычеркиваем столбцы, покрываемые строкой a.
  - Вычеркиваем строки, содержащие в столбце r «1».
  - Вычеркиваем строку a и столбец r.
- Включаем строку a в формируемое неизбыточное покрытие на этой итерации. Запоминаем для нее номер синдромного столбца r.

Замечания.

1. Ясно, что в результате многократного выполнения этих шагов будет сформировано очередное неизбыточное покрытие p_i.

2. На каждой очередной итерации i все вычеркнутые строки и столбцы снова включаются в матрицу B.

Итак, пусть p_i сформировано. Пусть, кроме того, R – наименьший к данному моменту размер найденного неизбыточного покрытия – рекорд. Строим новый столбец-резольвенту w, выбрав ровно R синдромных столбцов для любых R строк из p_i . В столбце-резольвенте w пишем «1» в тех и только тех строках, в которых **как минимум два** столбца из выбранных R синдромных столбцов содержат «1». В остальных строках в w пишем «0».

Присоединяем w к матрице B и переходим на следующую итерацию.

Эта схема алгоритма соответствует ПГР. Доказано [9], что столбец w исключает покрытие p_i, т.е. p_i не может более повториться на последующих итерациях. Пусть p_i было получено последовательным включением строк $a_1, a_2, \ldots, a_k$.

Теперь допустим, что новая итерация i+1 полностью повторяет предыдущую итерацию i. Это значит, что выбираются те же и в том же порядке синдромные столбцы и строки $a_1, a_2, \ldots, a_k$ и, возможно, еще какие-то строки. В момент включения в p_{i+1} строки a_k, последней в p_i, матрица B не может быть полностью разрушенной – тогда было бы $p_i = p_{i+1}$. Это значит, далее, что должен остаться хотя бы столбец w, который строки $a_1, a_2, \ldots, a_k$ не покрывают (свойство синдромных столбцов [9]). Но столбец w будет к данному моменту полностью нулевым, т.к. все строки, содержащие в столбце w «1», будут вычеркнуты при выборе тех же синдромных столбцов для строк $a_1, a_2, \ldots, a_k$. А этого быть не может, поскольку мы включаем в число синдромных столбцов столбец с наименьшим числом единиц на каждом шаге итерации. Поэтому:

- либо p_{i+1}. содержит меньшее число строк, чем k;

- либо на одном из шагов 1,2,…, k при формировании p_{i+1} в выбираемом синдромном столбце окажется меньше единиц, чем мы имели на предыдущей итерации по формированию p_i на том же шаге.

Это важное наблюдение дает возможность ограничить число присоединяемых столбцов-резольвент только теми, которые были использованы на текущей итерации в качестве синдромных. Таковых может быть не более R. Остальные столбцы резольвенты можно исключить совсем. Поверх исключаемых столбцов-резольвент записываем порождаемые новые столбцы-резольвенты.

Таким образом, приведенное усовершенствование алгоритма позволяет не выходить за пределы памяти, отводимой под m строк и $N+m$ столбцов матрицы B. (Изначально в B содержится m строк и N столбцов.)

Остается рассмотреть, как выполнить подключение ДПВЫП (или любой другой точной процедуры, хорошо работающей на матрицах небольших размеров. В частности, сам метод на основе принципа групповых резолюций дает очень быстрые ответы на матрицах с числом строк до 30 и числом столбцов до 45–50).

Процедура ДПВЫП подключается при выполнении каждой итерации для отыскания очередного неизбыточного покрытия, когда в текущей матрице B остается не более 25 строк и 40 столбцов. Эмпирически установлено, что в этом случае ДПВЫП, как правило, быстро отыскивает точное решение. Имея точное решение, найденное ДПВЫП, можно строить более качественные резольвенты.

Пусть значение рекорда есть $\boldsymbol{R}$. Размер текущего неизбыточного покрытия можно представить как

$$\boldsymbol{k} + \boldsymbol{r} \geq \boldsymbol{R},$$

где $\boldsymbol{k}$ – первые k строк покрытия, найденные до того, как ДПВЫП позволила найти точное решение, состоящее из $\boldsymbol{r}$ строк.

Будем строить резольвенту следующим образом. Выпишем k первых синдромных столбцов, соответствующих k первым строкам покрытия, найденным эвристически. Из остальных строк матрицы выпишем любые r столбцов, желательно с наименьшим возможным числом единиц. Полученную так матрицу назовем BW. Тогда в порождаемом столбце-резольвенте w ставим «1» в каждой строке α при условии, что истинно любое из следующих условий:

Условие А1. α содержит в $\boldsymbol{k}$ первых столбцах из BW две или более единицы.

Условие А2. α содержит в $\boldsymbol{k}$ первых столбцах из BW ровно одну «1», а в $\boldsymbol{r}$ остальных столбцах из BW не менее одной единицы.

Теорема 3.3. Резольвента w, построенная в силу **А1, А2**, такова, что при условии, что наилучшее из найденных покрытий с рекордом $\boldsymbol{R}$ не является минимальным, каждое минимальное покрытие содержит как минимум одну строку, покрывающую w.

Доказательство. Допустим обратное. Пусть ни одна из строк, покрывающих w, не входит в какое-нибудь минимальное покрытие. Тогда эти строки можно удалить без потери как минимум одного точного решения. Удалим их и рассмотрим полученную так подматрицу BW. В этой подматрице выделим строки, содержащие две или более единицы. Пусть ни одна из этих строк не входит ни в одно точное решение. Тогда улучшить покрытие для BW нельзя, ибо в каждой из оставшихся строк из BW не более одной единицы. Поэтому пусть одна или более строк, содержащая в r последних столбцах подматрицы BW две или более единиц, входит в минимальное покрытие. Заметим, что ни одна из таких строк не содержит ни одной единицы в k первых столбцах в BW. Удалим из матрицы BW каждую такую строку и покрываемые ею столбцы. В оставшейся подматрице BW в каждой строке будет не более одной единицы. Но в ней останутся все строки, для которых выписаны первые k синдромных столбцов. Теперь уже получаем противоречие, т.к. включив эти строки в точное решение, мы бы для оставшейся матрицы нашли меньшее покрытие, но этого нет по условию.

Пример. Для системы (**3.2**) исходная матрица покрытия представлена на рис. 3.1 (приведен результат вычеркивания сток и столбцов при выборе в качестве синдромного элемента в строке $\neg x_7$ и столбце j_6) . Пусть найдено неизбыточное покрытие со строками: $\neg x_7$, $\neg x_3$ || $x_1, x_2, x_5, x_4, x_8, x_6$ (до двойной разделительной черты записаны строки, выбранные эвристически, а после двойной черты записано точное решение оставшейся после этого редуцированной матрицы). И хотя найденное решение является точным, построим для него синдромную матрицу (рис. 3.2) и столбец-резольвенту для покрытия: $\neg x_7$, $\neg x_3$ || $x_1, x_2, x_5, x_4, x_8, x_6$. При этом первые $k = 2$ столбца выделены фоном. Согласно условиям А1, А2, результирующий синдромный столбец w получается нулевым, хотя в алгоритме [9] он содержал бы единицы в строках x_5, x_4, содержащих более одной единицы.

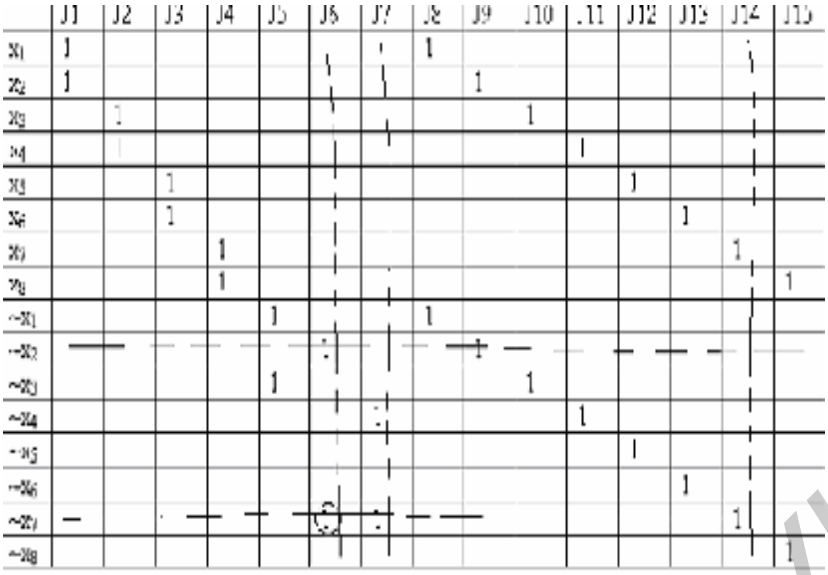

Рис. 3.1

	***J*5**	***J*6**	*J*1	*J*2	*J*3	*J*4	*J*11	*J*12
x_1			1					
x_2			1					
x_3				1				
x_4				1			1	
x_5					1			1
x_6					1			
x_7						1		
x_8						1		
$\neg x_1$		**1**						
$\neg x_2$	**1**							
$\neg x_3$		**1**						
$\neg x_4$							1	
$\neg x_5$								1
$\neg x_6$								
$\neg x_7$	**1**							
$\neg x_8$								

Рис. 3.2

4. МНОГОЗНАЧНЫЕ ЛОГИКИ И СИСТЕМЫ С НЕОПРЕДЕЛЕННОСТЯМИ

Неклассические логики играют существенную роль в теории и практике экспертных систем. Остановимся на многозначных логиках Я. Лукасевича с неопределенностями. Ясно, что n-значная логика при достаточно большом n может вполне удовлетворительно аппроксимировать нечеткую логику Л. Заде. Таким образом, эффективная машина вывода для многозначной логики важна и в плане использования ее для нечеткой логики Заде. Однако имеются два аспекта, касающихся нечеткой логики:

- использование принципа резолюций ограничено в силу следующего ограничения [10]:

$$a ? b ? \text{val}(a) ? \text{val}(a),$$

где a, b – формулы и $val(a)$, $val(b)$ представляют их нечеткие меры;

- $a \& \neg a$ не представляет противоречивой формулы в общем случае. Поэтому концепция противоречивости для таких систем должна быть ясно сформулирована и разрешена.

Введем 2-значное логическое исчисление, эквивалентное 3-значному исчислению Лукасевича с тремя значениями истинности: 0, ½, 1 и логическими операциями: & (конъюнкция), $\vee$ (дизъюнкция), $\neg$ (отрицание), $\rightarrow$ (следование) и $\leftrightarrow$ (эквивалентность). Это позволяет построить машину вывода стандартным способом. Описываемый подход может быть обобщен для произвольных n-значных логик Лукасевича и, следовательно, применим для нечеткой логики Заде с континуумом значений истинности для формул.

Для достижения поставленных целей нам потребуется некоторое промежуточное исчисление, названное здесь векторным логическим исчислением со специфически определенной операцией отрицания ($\neg$). Базовая идея, следовательно, такова: любая неклассическая логика может быть заменена эквивалентной ей 2-значной логикой.

4.1. ОПРЕДЕЛЕНИЯ

Пусть x,y,...,z представляют термы (переменные) 3-значной логики Лукасевича, которые принимают значения из множества {0, ½, 1}, где $0 < ½ < 1$. Логические операции ($?, \&, \neg$) вводятся стандартным способом:

$$val(x ? y) = \max(val(x); val(y)),$$
$$val(x \,\&\, y) = \min(val(x); val(y)),$$
$$val(\neg x) = 1 - val(x) = \begin{cases} 1, \text{if } val(x) = 0, \\ \frac{1}{2}, \text{if } val(x) = \frac{1}{2}, \\ 0, \text{if } val(x) = 1, \end{cases} \qquad (4.1)$$
$$val(x ? y) = \min(1, 1 ? val(x) + val(y)).$$

Здесь *val(x)* представляет значение истинности переменной *x*.

Пусть a, b – формулы. Тогда $a \to b$ означает, что $val(a) \le val(b)$ в любой интерпретации *I* (наборе значений истинности) для переменных в формулах a и b. Обозначим $a[m_\alpha]$ формулу, которая допускает только такие интерпретации *I*, в которых $val_I(a) \ge m_\alpha$. Можно рассматривать m_α как значение неопределенности формулы a.

Определение 4.1. Формула $e[м_e]$ выводима из формул $б_1[м_1],..., б_k[м_k]$ тогда (и только тогда), когда любая интерпретация *I*, допускаемая каждой формулой $б_i[м_i]$ $(i = \overline{1,k})$, допускается также формулой $e[м_e]$.

Ясно, что

$$a_1 \,\&\, a_2 \,\&...\&\, a_k ? e \qquad (4.2)$$

эквивалентно

$$б_1[0], б_2[0], ..., б_k[0] \vdash e[\min\{val(б_1),...,val(б_k)\}],$$

где $\vdash$ означает выводимость. Следовательно, (4.2) является частным случаем выводимости в *k*-значной логике $(k ? 3)$.

Импликация $a \to b$ понимается таким способом, что в любой интерпретации *I*, в которой $val(a \to b)=1$, имеет место $val(a) \le val(b)$.

Для 2-значной стандартной логики выводимость $(a \vdash b)$ и импликация $(a \to b)$ суть эквивалентные понятия. Для *n*-значной логики $(n > 2)$ можно получить аналогичный результат в форме следующей теоремы.

Теорема 4.1. $a\,[m_a] ? b\,[m_b] ? a\,[m_a] \quad в [\vdash м_в]$.

Доказательство.

1. Пусть $a\ [m_a]?b\ [m_b]$. Это означает, что в любой интерпретации ***I***, в которой $val_I(a) \geq m_\alpha$, имеет место $val_I(b) \geq m_\beta$ также. Тогда $б\ [м_б]\ \vdash\ в\ [м_в]$ в силу конечности числа интерпретаций *I*.

2. Пусть $б\ [м_б] \vdash\ в\ [м_в]$. Тогда $a\ [m_a]?b\ [m_b]$ в силу определения операции $(\rightarrow)$.

4.2. МАШИНА ВЫВОДА

Для того чтобы интерпретировать формулы 3-значного исчисления Лукасевича, введем векторную логику с формулами, аргументы которых представляют векторы, причем каждая переменная вектора представляет булевскую переменную, булевскую формулу или константу (0,1). Обозначим векторные формулы v и w как $v = (v_1, v_2)$, $w = (w_1, w_2)$ и определим следующие отношения:

$$
\begin{aligned}
v = 1 &\leftrightarrow (v_1 = 1, v_2 = 1),\\
v = 0 &\leftrightarrow (v_1 = 0, v_2 = 0),\\
v = \frac{1}{2} &\leftrightarrow (v_1 = 1, v_2 = 0),\\
\neg v &\leftrightarrow (\neg v_2, \neg v_1),\\
v \,\&\, w &\leftrightarrow (v_1 \,\&\, w_1, v_2 \,\&\, w_2),\\
v?w &\leftrightarrow (v_1?w_1,\ v_2?w_2),\\
v?w &\leftrightarrow v_i?w_i,\ \forall i = 1,2,\\
v[б] &\leftrightarrow I_1?I_2?...?I_t,\ val(v_{I_m})?a\ \ (m = 1,...,t).
\end{aligned}
\qquad (\mathbf{4.3})
$$

Замечание. Считается, что для каждой векторной формулы a выполняется следующее условие $a_1?\neg a_2$, которое не допускает набор $б = <0,1>$, исключаемый определением (**4.3**).

Рассмотрим следующие примеры в качестве пояснения.

Пример 4.1. Доказать или опровергнуть утверждение, что из формул $a \vee b$ $[m=1]$, $\neg a \vee b$ $[m \geq 0{,}5]$ можно вывести b $[m \geq 0{,}5]$.

Перепишем $a \vee b$ в векторной форме, как показано ниже:

$$(a_1,\ a_2)?(b_1,\ b_2) = (a_1?b_1,\ a_2?b_2) = (1\ ,\ 1).$$

Поскольку мера этой формулы равна 1, то можно записать

$$a_1 ? b_1,$$
$$a_2 ? b_2. \qquad \textbf{(4.4)}$$

Вторая формула $\neg a ? b \ [m ? 0,5]$ переписывается следующим образом:

$$(\neg a_2,\ \neg a_1) ? (b_1,\ b_2) = (\neg a_2 ? b_1,\ \neg a_1 ? b_2) =$$
$$= (1,\ 0) ? (1,\ 1),$$

откуда получаем $\neg a_2 \vee b_1$,
что дает исходную систему посылок в виде

$$\neg a_2 \vee b_1,$$
$$a_1 \vee b_1,$$
$$a_2 \vee b_2, \qquad \textbf{(4.5)}$$
$$a_1 \vee \neg a_2,$$
$$b_1 ? \neg b_2.$$

Необходимо показать, что из (**4.5**) следует формула $в = (в_1, в_2) \ [m ? 0,5]$ (т.е. $(1,\ 0) ? (1,\ 1)$ или просто $в_1$). Это можно сделать с помощью стандартной резолюционной стратегии: $в_1$ действительно следует из (**4.5**).

Пример 4.2. Доказать или опровергнуть утверждение, что из формул

$$a \vee b \ [m ?\ 0],$$
$$\neg б \ [m ?\ 0]$$

можно вывести

$$в \ [m ? \min\{ val\ (a ? b);\ val\ (\neg a)\}].$$

Имеем

$$(a ? b) = (a_1 ? b_1, a_2 ? b_2) = (0,\ 0) ? (1,\ 0) ? (1,\ 1),$$

что дает

$$a_1 ? b_1 ? \neg (a_2 ? b_2),$$
$$\neg a = (\neg a_2,\ \neg a_1) = (0,\ 0) ? (1,\ 0) ? (1,\ 1),$$

откуда

$$\neg a_2 ? a_1 .$$

Покажем, что

$$(a ? b) \& \neg a ? b ,$$

что эквивалентно

$$\neg a_2 \& (a_1 ? b_1) ? b_1 ,$$
$$\neg a_1 \& (a_2 ? b_2) ? b_2 .$$

Итак, для системы посылок

$$b_1 ? \neg b_2 ,$$
$$a_1 ? b_1 ? \neg (a_2 ? b_2) ,$$
$$\neg a_2 ? a_1$$

необходимо вывести

$$\neg a_2 \& (a_1 ? b_1) ? b_1 ,$$
$$\neg a_1 \& (a_2 ? b_2) ? b_2 .$$

Нетрудно заметить, что выводимость в этом случае места не имеет.

Общая схема проверки выводимости состоит из следующих шагов:

1. Рассматриваем выводимость в общем виде:

$$a_1[m_1] \& \ldots \& a_z[m_z] ? b[m_b] . \qquad (\mathbf{4.6})$$

2. В соответствии с векторным представлением формул получаем систему посылок и заключений для (**4.6**).

3. Выполняем стандартную резолюционную стратегию в 2-значной логике.

4. Доказательство формулы в форме $a_1 \& a_2 \& \ldots \& a_n ? b_1 ? b_2 ? \ldots ? b_m$ требует записать каждую формулу $б_i$, $в_j$ в векторной форме $б_i = (б_{i_1}, б_{i_2})$, $в_j = (в_{j_1}, в_{j_2})$ и затем показать выводимость

$$\begin{cases} a_{11} \& a_{21} \& \ldots \& a_{z1} ? b_{11} ? b_{21} ? \ldots ? b_{m1}, \\ a_{12} \& a_{22} \& \ldots \& a_{z2} ? b_{12} ? b_{22} ? \ldots ? b_{m2}. \end{cases}$$

4.3. ОБОБЩЕНИЕ ПОДХОДА

Представленный выше подход может быть сравнительно легко обобщен для случая *k*-значной логики с $k > 3$. Прежде чем мы сформулируем общий случай, рассмотрим варианты с $k = 4$, $k = 5$.

$\boldsymbol{k} = 4$. В этом случае используем векторное представление

$$v = (v_1, v_2, v_3)$$

с операцией отрицания в форме

$$\neg v = (\neg v_1, \neg v_3, \neg v_2)$$

и значениями истинности

$$v = 0?(0,\ 0,\ 0),$$
$$v = 1?(0, 1,\ 0),$$
$$v = 2?(1,\ 1,\ 0),$$
$$v = 3?(1,\ 1,\ 1).$$

$\boldsymbol{k} = 5$. В этом случае имеем

$$v = (v_1, v_2, v_3, v_4) \text{ и } \neg v = (\neg v_2, \neg v_1, \neg v_4, \neg v_3),$$
$$v = 0?(0,\ 0,\ 0, 0),$$
$$v = 1?(0,\ 0, 1,\ 0),$$
$$v = 2?(0, 1,\ 1,\ 0),$$
$$v = 3?(1,\ 1,\ 1, 0),$$
$$v = 4?(1,\ 1,\ 1, 1).$$

Общий случай может быть представлен в таком виде:

$$v = (v_1, v_2, v_3, \ldots, v_n),$$
$$v = 0?(0,\ 0,\ 0,\ 0, \ldots,\ 0,\ 0),$$
$$v = 1?(0,\ 0,\ 0, 0, \ldots,\ 1,\ 0),$$
$$v = 2?(0, 0, 0, \ldots 1, 1, 0),$$
$$\ldots$$
$$v = n - 1?(1, 1, 1, 1, \ldots, 1,\ 0),$$
$$v = n?(1,\ 1,\ 1, 1, \ldots, 1,\ 1)$$

с операцией отрицания $\neg v$, удовлетворяющей $val(\neg v) = n - val(v) + 1$ и

$$\neg \mathrm{v} = (\neg \mathrm{v}_{n-2}, \ldots, \neg \mathrm{v}_3, \neg \mathrm{v}_2, \neg \mathrm{v}_1, \neg \mathrm{v}_n, \neg \mathrm{v}_{n-1}). \qquad (\mathbf{4.7})$$

Корректность операций & и $\vee$ в общем случае устанавливается непосредственно. Рассмотрим операцию отрицания (**4.7**). Для наборов $(0,0,...,0)$ и $(1,1,...,1)$ она корректна. Возьмем произвольный набор $I_1 = (\underbrace{0,0,...,0}_{k},\underbrace{1,1,...,1}_{m},0)$. Ясно, что I_1 соответствует значению истинности m. По определению, $I_2 = \neg I_1$ соответствует набору $(0,0,...,0,\underbrace{1,1,...,1}_{k-1},1,1,0)$.

Поэтому $val(I_2) = k+1$ и $val(I_2) = n - val(I_1) + 1$, $val(I_1) = m$, где $n = k + m$.

Также легко определяются формулы, устанавливающие только допустимые наборы. Эти формулы таковы: $v_i ? v_{i+1}$ $(i = 1,...,n-2)$ и $v_n = 1 ? v_i = 1$. Они указывают, что «1» в векторном представлении могут располагаться только последовательно одна за другой.

Рассмотренный здесь общий подход дает ключ к построению машины логического вывода в нечеткой логике, посредством аппроксимации нечетких значений значениями многозначной логики, достаточными для практических приложений. Следует заметить, что проблема построения машины вывода для нечеткой логики не решена удовлетворительно до настоящего времени. Привлекательными чертами изложенного подхода являются следующие. Рассмотрим, к примеру, формулу $б = (б_1 б_2 б_3 б_4)$ в 5-значной логике с мерой неопределенности $м_б = 2$. Тогда легко записать все интерпретации, допускаемые $б$ в форме единственного терма $б_2$, поскольку $б_2 = 1$ в <0110>, <1110> и <1111>. Если, например, $б = (б_1 б_2 б_3 б_4)$, то интерпретации, допускаемые $б$, представляются термом $б_1 = 1$ и т.д. Это наблюдение показывает, что использование меры неопределенности для многомерных формул не ведет к росту сложности представления формул.

5. ПРОГРАММИРОВАНИЕ В СРЕДЕ VISUAL PROLOG

5.1. ОСНОВЫ СОЗДАНИЯ И ВЫПОЛНЕНИЯ ПРОГРАММ В VISUAL PROLOG

Программы, разрабатываемые в среде Visual Prolog (VIP) , оформляются как проекты. Проекты могут строиться на различных платформах. От этого зависит «уровень функциональных возможностей» приложения. Сначала рассмотрим создание проекта на платформе MS DOS.

Этот вариант сохраняет полную преемственность с языком Borland Prolog.

Запустите Пролог и выберите пункт меню **Project** главного окна системы, затем – опцию **New Project**. На экране появится диалоговая форма (рис. 5.1).

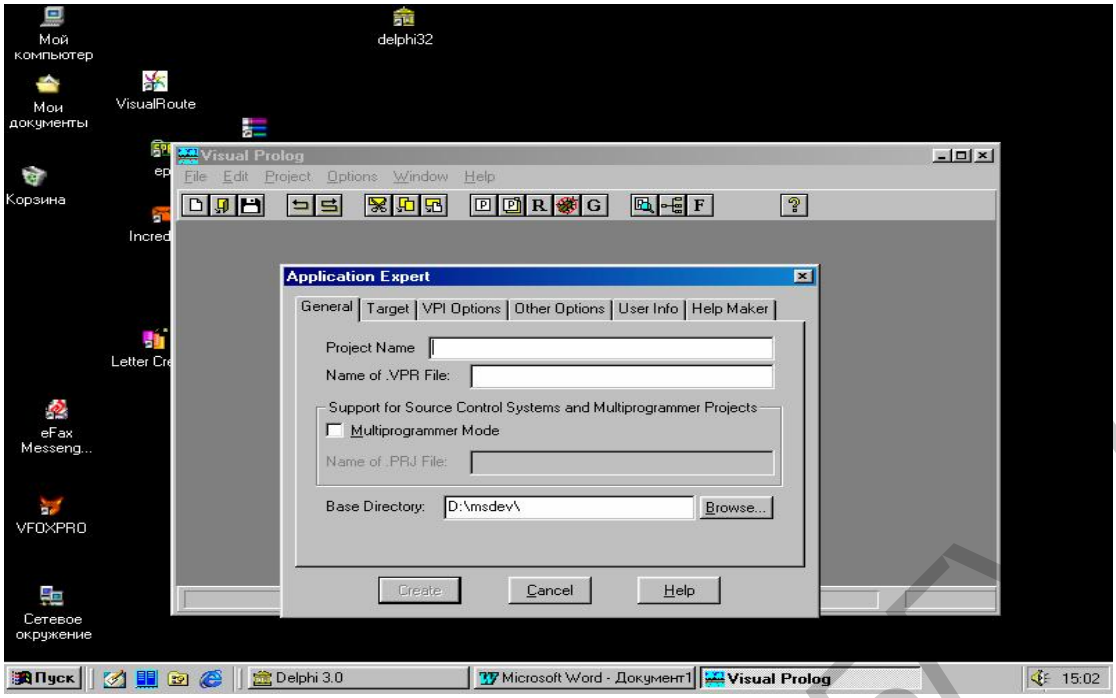

Рис. 5.1

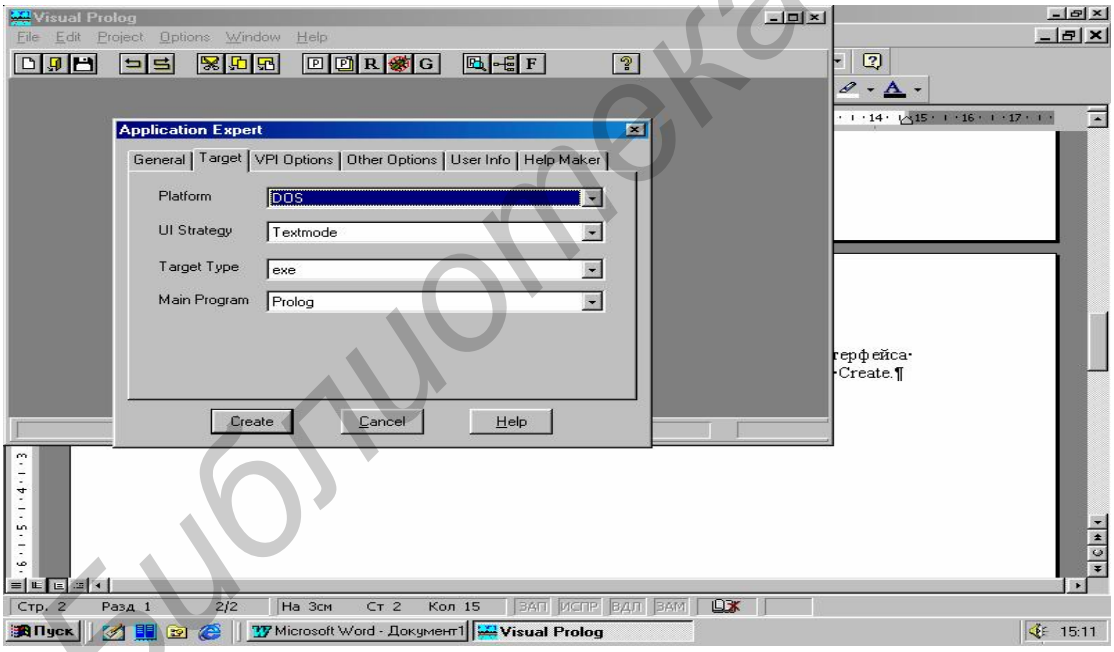

Рис. 5.2

Введите имя проекта в поле **Project Name**. В поле **Name of .VPR file** достаточно просто щелкнуть мышью и задать нужное имя файла проекта. В поле **Base Directory** нужно установить имя используемого для сохранения проекта директория. Заполнив эти поля, щелкните по закладке **Target**. Новое содержимое экрана будет таким, как показано на рис. 5.2. Вам надлежит

заполнить в этом окне поля: **Platform** (введите DOS), **UI Strategy** (введите TextMode).

После этих действий нажмите кнопку **Create** (рис. 5.2). Система создаст пустой проект. В окне проекта будет отображена следующая информация (рис. 5.3):

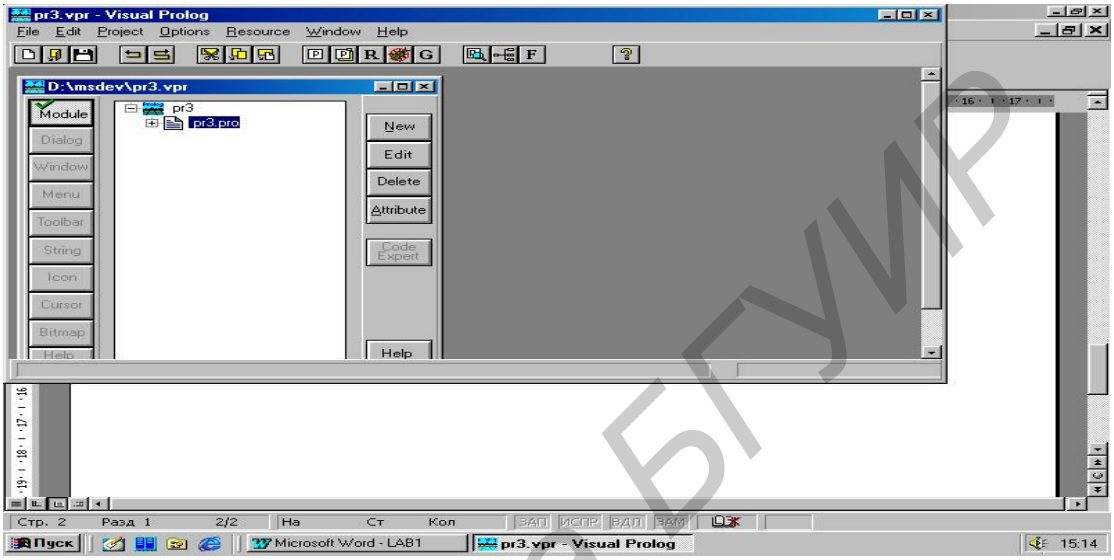

Рис. 5.3

Теперь все, что остается сделать, это набрать собственно код программы и выполнить ее из пункта **Project–Run**. Для создания программы выделите щелчком мыши имя модуля (показано темным цветом) и нажмите кнопку **Edit**. В окне редактора запишите следующий учебный пример:

```
GOAL
Makewindow(1,10,4,”myWin”,0,0,25,80),
Write(“Привет, мои друзья из VISUAL PROLOG”),
Readchar(_),
Removewindow.
```

Эта программа при запуске выведет сообщение, как показано на рис. 5.4:

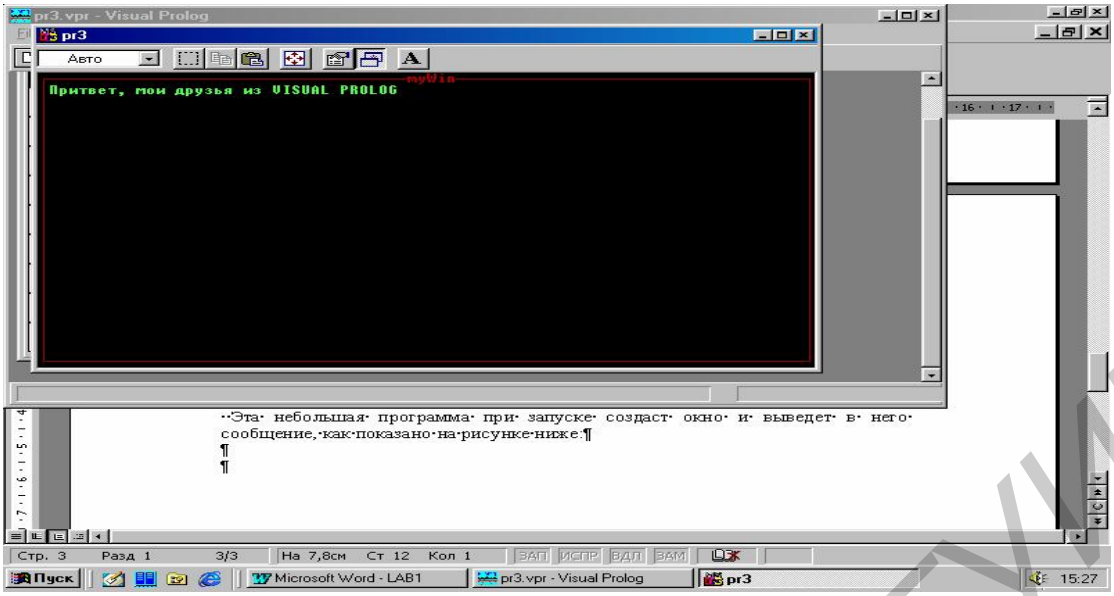

Рис. 5.4

Некоторое неудобство создает платформа DOS, которая приводит к выводу промежуточных указаний при выполнении программы. При выводе подсказок DOS игнорируйте их нажатием кнопки **Enter**.

Теперь приступим к изучению способа написания программ для WINDOWS. Эта первая программа должна продемонстрировать общий стиль создания приложений в объектной среде VIP. При запуске программы появится окно с кнопкой; после нажатия на кнопку система выдаст еще одно диалоговое окно с некоторым сообщением. Прежде всего создадим пустой проект, поступая аналогично ранее рассмотренному. Для этого закроем текущий проект с помощью опций **Project–Close Project**. Введем имя проекта и имя файла проекта. Выберем вкладку Target и зададим платформу WINDOWS32 и интерфейс VIP. Нажмем кнопку **Create Project**. В результате этих действий появится полноценное окно проекта. В левой части окна проекта отображены составные его части: модули, окна, диалоги, меню, курсоры, растровые изображения и др. В правой части окна проекта размещены кнопки для управления процессом проектирования и редактирования.

Первым делом добавим в проект свое окно: для этого в окне проекта сначала щелкнем мышью по элементу **Window**, а затем – по кнопке **New**. В окне, которое появится в ответ на эти действия, нужно задать имя окна, его размеры, флаги, определяющие такие характеристики, как наличие меню, наличие кнопок управления окном, сворачиваемость окна, возможность изменять размеры и др. Создадим окно, нажав кнопку **OK**. Теперь в проекте будет два окна: **Task Window** (окно задачи) и ваше. Запустим приложение на выполнение. При этом на экране появится только окно **Task Window**.

Для отображения созданного вами окна потребуются дополнительные действия. Во-первых, нажмите кнопку **Code Expert** и установите для вашего

окна место расположения программного кода внутри создаваемого программного модуля (рис. 5.5, поле **Module**).

Во-вторых, поместите команду создания вашего окна в обработчике события **e-create** для окна приложения **Task Window**. С этой целью сделайте следующее. Зайдите в **Code Expert** , выберите в нем объект **Task Window**, найдите в окне событий (**Event Handler**) событие **e-Create** и затем, выделив это событие, нажмите кнопку **Edit Clause**. В результате мастер кода отыщет как раз то место, где расположен обработчик этого события для окна **Task Window**. Вам нужно вставить фрагмент собственного кода в этот обработчик. Обратите внимание, что комментарии отмечаются в тексте программы символом % перед строкой либо ограничиваются символами /* ... */.

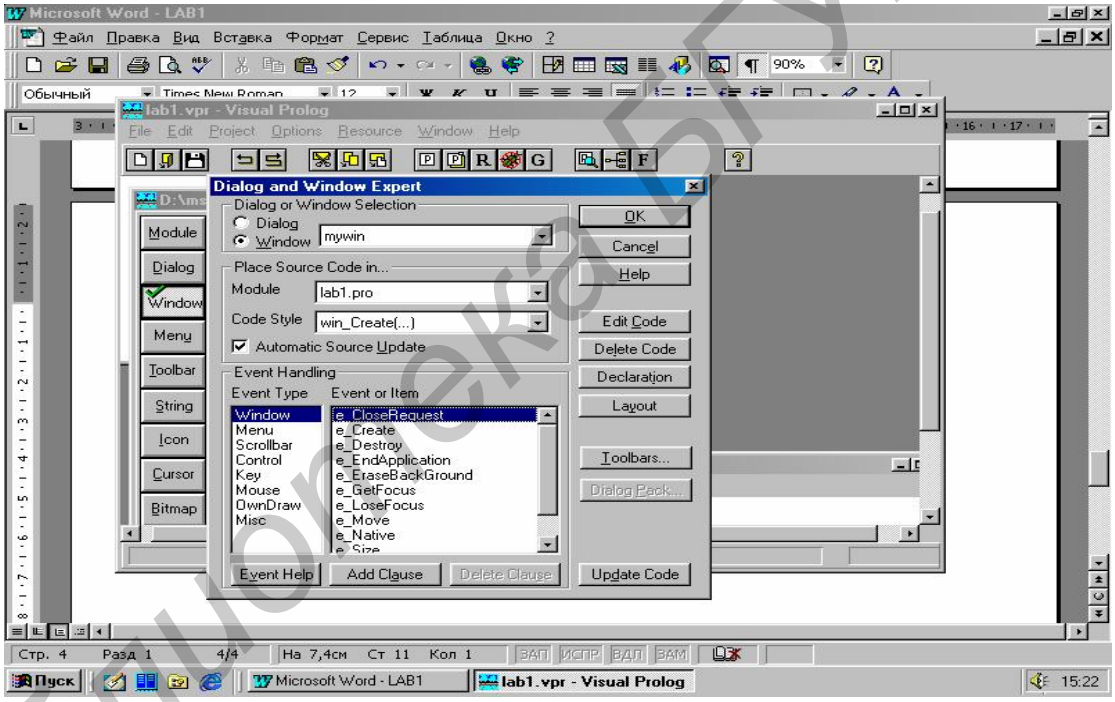

Рис. 5.5

Теперь отметим важнейшее для понимания работы VIP обстоятельство. Система работает на основе принципа обработки событий от объектов приложения. Такими событиями для окон, например, являются их активация, создание, изменение размеров, щелчки мышью и др. При программировании необходимо самим запрограммировать те из событий, которые требуются для логики приложения. В частности, чтобы отобразить наше собственное окно, мы поместили команду его активизации в обработчик события **e-Create** для другого окна, именно – для окна **Task Window**.

%BEGIN Task Window, InitControls, 15:16:19-18.7.2002, Code automatically updated!

```
%END Task Window, InitControls
%BEGIN Task Window, ToolbarCreate, 15:16:19-18.7.2002, Code automatically updated!
        tb_project_toolbar_Create(_Win),
        tb_help_line_Create(_Win),
         win_mywin_Create(_Win),   %  !!! то, что добавлено нами
%END Task Window, ToolbarCreate
ifdef use_message
        %msg_Create(100),      !!! то, что добавлено нами
enddef
```

Итак, мы ввели команду win_mywin_Create(_Win) и взяли в комментарий команду msg_Create(100) , которая отображает окно сообщений.

Для построения собственных обработчиков сообщений следует использовать программу **Code Expert**. Окно этой программы активизируется при нажатии одноименной кнопки из окна проекта (рис. 5.6).

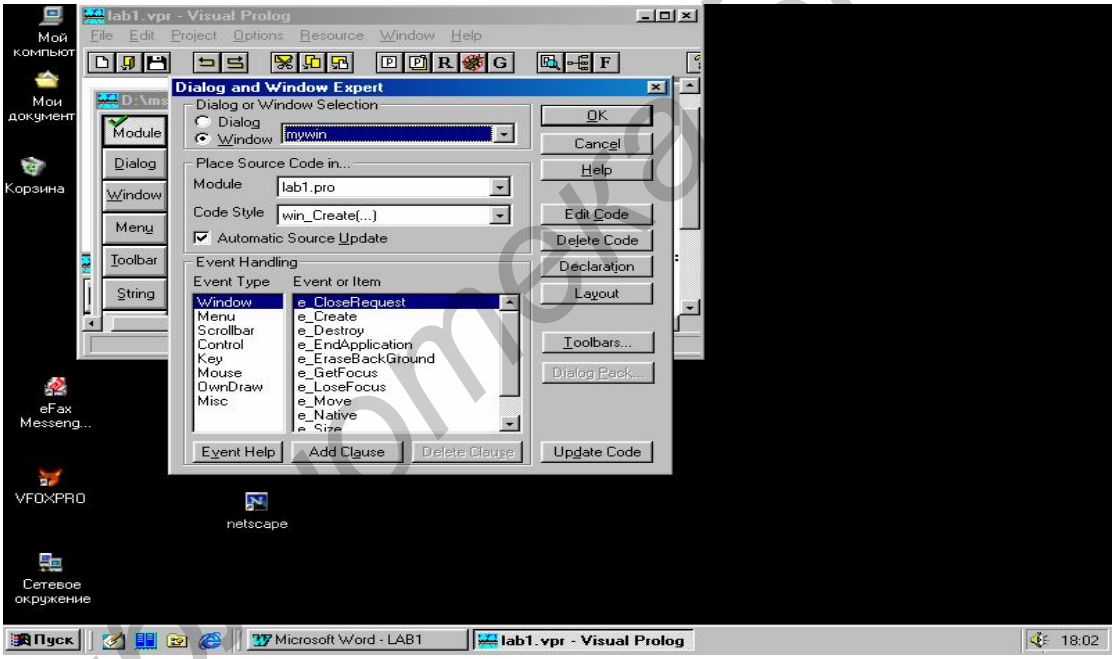

Рис. 5.6

Список событий помещен в окне **Event or Item**. Для того чтобы запрограммировать обработку события, следует выделить имя события щелчком мыши и нажать кнопку **Add Clause**, которая создаст пустой фрагмент обработчика. Эта кнопка будет переименована в **Edit Clause** сразу же. Теперь нажмите кнопку **Edit Clause** и откроется именно требуемый участок кода. Вам надлежит запрограммировать его по собственному усмотрению. Продемонстрируем сказанное следующим. Создадим на нашем окне кнопку и запрограммируем событие ее нажатия так, чтобы появлялось некое диалоговое окно с текстом приветствия. Для этого сначала вернемся в окно проекта.

Выберите пункт **Window** главного меню и затем щелкните по имени файла проекта. Еще раз выберите п. **Window** и щелкните по имени вашего окна. После этого ваше окно отобразится на экране вместе с окном визуальных элементов управления (рис. 5.7).

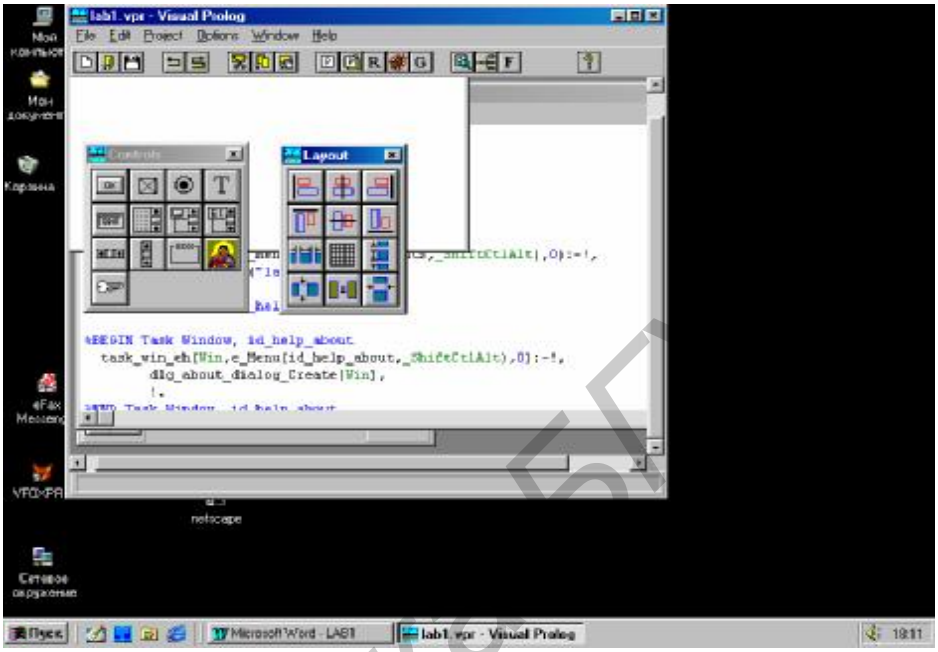

Рис. 5.7

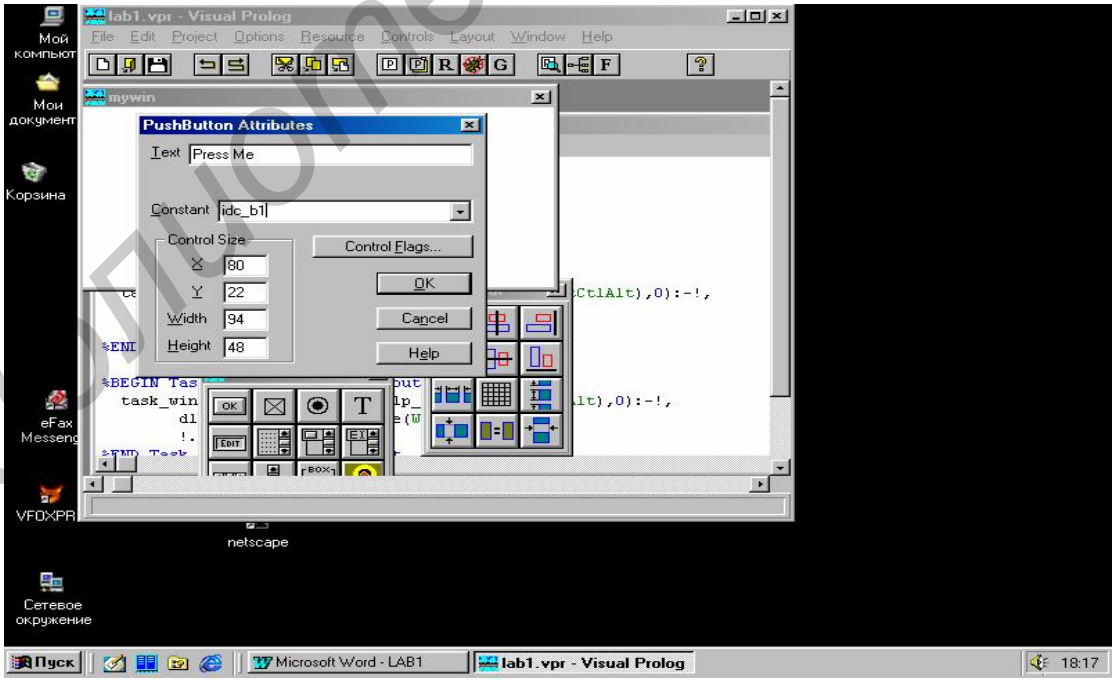

Рис. 5.8

Элемент с надписью **OK** представляет обычную командную кнопку. Щелкните по нему мышью и, не отпуская кнопку мыши, прорисуйте командную кнопку в вашем окне. Появится новое окно (рис. 5.8). Введите надпись на кнопке (в примере – **Press Me**) и укажите программный идентификатор кнопки в поле **Constant**. Этот идентификатор следует использовать при ссылке на кнопку в обработчике событий. Нажмите кнопку **OK** и вызовите мастер кода **Expert Code**.

Выберите элемент **Control** в левом окошке: появится список событий для созданной вами кнопки (рис. 5.9).

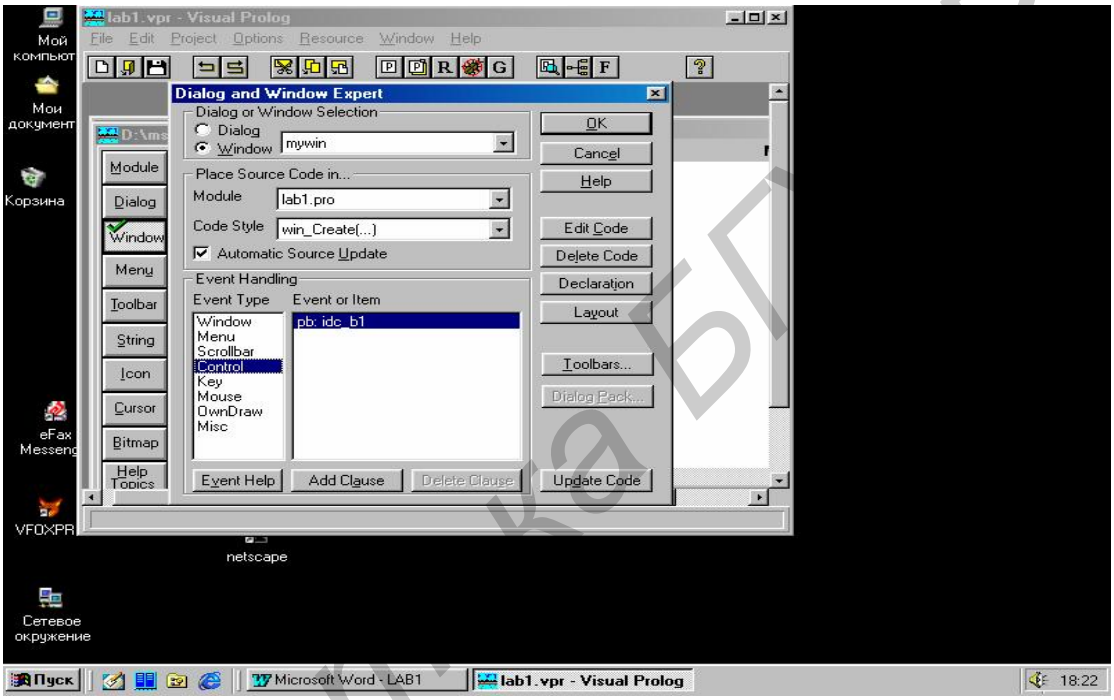

Рис. 5.9

Нажмите кнопку **Add Clause**, а затем **Edit Clause**. Теперь откроется именно то место редактора кода, где вам следует дописать собственный код. Это место выглядит следующим образом:

```
%END mywin, e_Size
%BEGIN mywin, idc_b1 _CtlInfo
  win_mywin_eh(_Win,e_Control(idc_b1,_CtrlType,_CtrlWin,_CtlInfo),0):-!,
        !.
%END mywin, idc_b1 _CtlInfo
```

Теперь мы вставляем собственный код, представленный одной командой:

```
%END mywin, e_Size
%BEGIN mywin, idc_b1 _CtlInfo
  win_mywin_eh(_Win,e_Control(idc_b1,_CtrlType,_CtrlWin,_CtlInfo),0):-!,
```

```
        dlg_ask("Привет Всем",["OK"]).
%END mywin, idc_b1 _CtlInfo
```

Добавленная нами строка выделена жирным шрифтом. Эта строка запускает диалоговое окно с единственной кнопкой **OK**.

Промежуточный вариант приложения готов. Запустите приложение на выполнение командой **Run** из пункта **Project**. Появится созданное нами окно с кнопкой, нажатие на которую выводит на экран новое диалоговое окно с приветствием.

Рассмотрим другие элементы. Элемент типа поля редактирования **Edit Control** позволяет вводить и изменять текст. Этот элемент порождает только три события: e_GetFocus, e_loseFocus, e_modified. Первые два связаны с получением и потерей фокуса. Последнее событие – с изменением содержимого. Для получения текста в поле редактирования следует применить команду

String= win_GetText(WinHandle),

где аргумент команды должен быть предварительно определен командой

WinHandle= win_GetCtlHandle(ParentWin,CtlId),

причем CtlId – это программный идентификатор поля редактирования, который мы вводили в поле **Constant** при его создании. **ParentWin** – программное имя родительского окна. Соответствующий фрагмент кода для обработки события e_getFocus помещен нами ниже:

```
%BEGIN mywin, id_edit1 getfocus
  win_mywin_eh(_Win,e_Control(id_edit1,_CtrlType,_CtrlWin,getfocus),0):-!,
    H_win =win_GetCtlHandle(_WIN,id_edit1),
     win_SetText(H_win,"Cheers"),
        !.
%END mywin, id_edit1 getfocus
```

Выделенное жирным добавлено нами в обработчик события. Интересный класс событий составляет **e_user**(_ID,_PTR). Это событие предоставлено для пользователя. Оно не связано с какими-либо конкретными объектами. Поэтому такое событие можно использовать для запуска различных предикатов. При этом VIP предоставляет средство для активизации событий. Это команда

win_SendEvent(_Win, e_user(10, 20)) //пример.

В команде **win_SendEvent(родительскоеокно, событие)** следует указать заголовок (handle) родительского окна и событие вместе с его аргументами,

которое должно быть активизировано. В примере выше мы указали одну из многочисленных возможностей. Теперь рассмотрим, как создавать меню в программах.

Для создания меню в окне проекта выберите элемент **Menu** и нажмите кнопку **New**. Введите во всплывающем окне название меню и создайте его подпункты. Не забывайте при создании каждого подпункта нажимать кнопку **New**. Созданное меню можно просмотреть с помощью кнопки **Test** в окне проектировщика меню. Закройте окно меню с помощью кнопки **Close**. Теперь нужно связать созданное меню с конкретным окном. Воспользуемся уже созданным нами окном для этих целей. В окне **Code Expert** выберите имя нашего нового окна и измените содержимое поля **non-menu** на имя созданного меню.

Нам потребуется еще использовать элемент **ListBox** (список элементов). Разместите его в окне подобно тому, как мы размещали кнопку и поле **Edit**. Ниже приведены основные события и методы объекта **ListBox** .

Когда пользователь выполняет выделение элемента списка

```
ehandler(Win, e_Control(CtrlID, CtrlType, CtrlWin, selchanged), 0):-
% при двойном щелчке по элементу списка
ehandler(Win, e_Control(CtrlID, CtrlType, CtrlWin, activated), 0):-
% при получении списком фокуса
ehandler(Win, e_Control(CtrlID, CtrlType, CtrlWin, getfocus), 0):-
% при потере списком фокуса
ehandler(Win, e_Control(CtrlID, CtrlType, CtrlWin, losefocus), 0):-.
```

Эти события отображаются в окне события мастера Code Expert.

Основные предикаты для работы со списком таковы:

```
lbox_Add(WINDOW, INTEGER Index, STRING Str)
```

Данный предикат добавляет строку в список. WINDOW – заголовок окна. Index – порядковый номер добавляемого элемента (если Index = –1, то элемент добавляется в конец списка).

```
lbox_Add(WINDOW, INTEGER Index, SLIST StringList)
```

аналогичен предыдущему предикату, но добавляет не одну строку, а список строк StringList .

```
lbox_Clear(WINDOW)
```

удаляет все элементы списка.

```
lbox_Delete(WINDOW, INTEGER Index)
```

удаляет элемент с заданным индексом из списка.

INTEGER = lbox_CountAll(WINDOW)

возвращает число элементов в списке.

lbox_GetItem(WINDOW, INTEGER Index)

возвращает элемент с заданным номером.

lbox_GetSel(WINDOW, SLIST, ILIST)

возвращает список выделенных элементов списка и их номера во втором списке.

INTEGER = lbox_GetSelIndex(WINDOW)

возвращает номер выделенного элемента списка.

lbox_IsSel(WINDOW, INTEGER Index)

проверяет, выделен элемент списка с данным номером или нет.

lbox_SetColumnWidth(WINDOW, INTEGER width)

устанавливает ширину списка.

Подведем основные итоги. Мы выяснили, что основной платформой для создания VIP-приложений является объектно-ориентированная WINDOWS-платформа. Мы научились создавать собственные окна, добавлять в них меню и размещать в окнах визуальные элементы управления (кнопки, поля редактирования, списки и др.). Было показано, что с визуальными элементами управления обеспечивается событийный интерфейс. С каждым элементом связывается множество событий, и программирование событий возлагается на пользователя. Имеется программа-мастер Code Expert, которая позволяет связать выбранное событие с программным фрагментом, ответственным за обработку события.

5.2. РАЗРАБОТКА ЛОГИЧЕСКИХ ПРОГРАММ

Основной структурной единицей при программировании на языке Пролог является **клоз** (от англ. clause – предложение). Клоз имеет общий формат следующего вида:

Если (условие) То (результат).

В языке Пролог этот формат имеет обратный порядок:

То (результат) Если (условие).

Последнее выражение упрощается до такого:

Результат :- Условие.

Записанное выше и есть клоз. Отметим следующее. Клоз всегда заканчивается точкой. Условие, как правило, представляет несколько записанных подряд выражений, отделенных друг от друга запятой. Каждое условие обрабатывается последовательно, в порядке очередности. Пример:

предпочитает(иван,X):- интересен(X), дешевый(X).

Здесь два условия: **интересен(X)** и **дешевый(X).** Если оба они истинны, то система выводит заключение, что **предпочитает(иван,X)** также истинно**.** Большими буквами (запомните) представляются переменные, малыми – символьные константы. Так что «иван» это константа; X – переменная. Пролог придуман так, что можно записать отношения между переменными, а система из этих отношений определит и сами переменные. Это аналогично системе уравнений в математике, если под уравнениями понимать отношения. Однако Пролог оперирует только *логическими* отношениями, так что нужно еще придумать, как систему алгебраических уравнений представить в логической форме. Запомним терминологию Пролога. Условие (отношение) называется иначе предикатом. Предикат – это формула с произвольными аргументами, принимающая только два значащих значения: **истина** или **ложь**. Незначащим значением является неопределенность, но в Прологе оно не допустимо. Аргументы предикатов называются **термами**. Термы могут быть **константами**, **переменными** или **функторами** (функциями). Запятая, отделяющая два предиката в клозе, имеет смысл «**И**», а точка с запятой, отделяющая два предиката в клозе, имеет смысл «**ИЛИ**».

Рассмотрим подходящий алгебраический пример. Пусть некто сообщает, что он задумал код (шифр) из пяти двухзначных (0 или 1) цифр. Обозначим цифры X1, X2, X3, X4, X5. При этом некто приводит следующие отношения между цифрами:

X1+X2+X3 >= 2,
X2+X3+X4 <= 2,
X2+X4+X5 <= 1,
X3+X4+X5 >= 1,
X1+X5 <= 1.

Попробуем заставить Пролог найти эти цифры.
Введем предикат

znach(X),

который устанавливает значение переменной X. Определение этого предиката таково:

znach(B):-B=0; B=1.

Предикат можно прочитать так: если B = 0 или B = 1, то значением переменной B является соответственно «0» или «1» и другого быть не может. Интересно, как осмыслить предикат без аргументов? Например, рассмотрим клоз:

```
EQUATION:-
                znach(X1),
                znach(X2),
                X1+X2=2.
```

Системе предлагается подобрать такие значения для переменных X1 и X2, чтобы их сумма равнялась 2. Решение, разумеется, единственное. Но для чего нужен предикат **EQUATION**? Этот предикат «возглавляет целый клоз», а потому его надо и рассматривать как структурную единицу программы. Все клозы должны быть явным образом связаны. Для этого используется раздел программы, называемый GOAL (по-английски – цель). Чтобы понять сказанное, рассмотрим уже завершенный вариант нашей программы:

```
  predicates
nondeterm      equation
nondeterm      znach(integer)

               GOAL
                     equation.
      CLAUSES

        EQUATION:-
```

```
                znach(X1),
                znach(X2),
                znach(X3),
                znach(X4),
                znach(X5),
               X2+X3+X4<=2,
               X2+X4+X5<=1,
               X3+X4+X5>=1,
               X1+X5<=1,
Write("X1=",X1," X2=",X2," X3=",X3," X4=",X4," X5=",X5).
znach(B):-B=0; B=1.
```

Итак, мы привели законченную программу. Программа на Прологе состоит из секций **PREDICATES, CLAUSES,GOAL** (и др.). В секции PREDICATES объявляются предикаты и их аргументы (если есть). Стандартными типами аргументов предикатов являются **integer, real, string, char, symbol** (соответственно – целый, вещественный, строковый – берется в двойные кавычки, символьный – берется в одиночные кавычки; символьный также записывается без кавычек малыми буквами). В целях сопряжения с программами в Windows дополнительно к стандартным введены типы: **byte** (байтовый без знака), **sbyte** – (байтовый со знаком), **long** – длинное целое, **short** – короткое целое, **ushort** – беззнаковое целое, **dword** – двойное слово, **ulong** – беззнаковое длинное целое. В секции GOAL удобнее всего объявить один единственный предикат, представляющий всю программу как таковую. (Впрочем, это не обязательно.) В секции CLAUSES приводятся клозы, определяющие предикаты. Клозы для одного и того же предиката должны быть записаны вместе.

Теперь рассмотрим важнейший механизм вывода, который неявно включался в процессе выполнения программы. Этот механизм можно назвать **ветвление-возврат**. Данное в программе определение предиката znach можно переписать «более естественно» так:

```
znach(B):-B=0.
znach(B):- B=1.                                    (*)
```

Здесь использованы два клоза вместо одного, хотя смысл действий при этом не изменился. Именно, система всегда обрабатывает клозы последовательно, в порядке записи. При первом обращении к предикату znach переменная B получает значение B=0. Если это значение не позволяет удовлетворить остальным предикатам, то выполняется повторное обращение к znach и переменная B получает значение из второго клоза, т.е. B = 1. Теперь рассмотрим целиком процесс выполнения логической программы нашего

примера. Первым выполняется самый первый предикат раздела GOAL – EQUATION. Система ищет определение этого предиката в разделе CLAUSES. Это определение имеется. Далее система выбирает первое условие из определения предиката EQUATION – znach(X1). Система пытается доказать это условие и ищет определение предиката znach в разделе CLAUSES. Таким определением является **(*)**. Система использует первое определение:

znach(B):-B=0.

Рассмотрим этот момент внимательнее. В определении (*****) имеется две альтернативы: для B=0 и B=1. Процесс выбора одной из альтернатив называется **ветвлением**. Всегда выбирается первая по порядку из числа неисследованных альтернатив. Итак, в момент выбора альтернативы имеется два предиката znach: первый из них – это предикат **znach(X1),** который система пытается доказать; и второй – **znach(B):-B=0,** который система берет из определения. Система сопоставляет эти два предиката и пытается согласовать их аргументы. Согласование аргументов называется **унификацией**. При унификации аргументы предикатов сопоставляются в порядке их расположения. У предиката znach только один аргумент. У доказываемого – это переменная X1, у взятого из определения - это B (тоже переменная). Таким образом, выполняется привязка аргумента X1 к аргументу B. Далее выбирается условие B = 0, доказывать которое не надо, т.к. это стандартная арифметическая операция. Переменная B получает значение B = 0. Поскольку X1 привязана к B, то и X1 получает значение X1 = 0. На этом доказательство предиката **znach(X1)** успешно завершено. Процесс доказательства переходит на предикат **znach(X2),** для которого действия выполняются по аналогии, так что X2 получает значение X2 = 0. И далее, таким же путем, X3 = 0, X4 = 0, X5 = 0. Теперь система успешно проходит проверку условий

X2+X3+X4 <= 2,
X2+X4+X5 <= 1,

но проверка условия

X3+X4+X5 >= 1

заканчивается неудачей. Теперь в действие вступает механизм **возврата**. Система возвращается в точку последнего **ветвления**. Это опять соответствует определению предиката znach, использованному при выборе значения X5 = 0. На этот раз система воспользуется определением

znach(B):- B=1. Поэтому **X5=1** на этот раз, и теперь уже все условия

X2+X3+X4<=2,
X2+X4+X5<=1,
X3+X4+X5>=1,
X1+X5<=1

будут успешно пройдены.

В заключение система выполнит простой стандартный вывод на экран значений переменных, полученных на этом этапе:

write(“X1=”,X1,” X2=”,X2,” X3=”,X3,” X4=”,X4,” X5=”,X5).

Итак, мы рассмотрели простейшую структуру программы, ее составные части и описали процесс выполнения логической программы. При этом мы выяснили, в чем состоят логические механизмы ветвления, возврата и унификации. Этих знаний нам хватит, чтобы составить собственные программы для выполнения. Но прежде чем перейти к практической части, покажем, как реализовать приведенную программу на платформе Windows. Поскольку наши практические навыки достаточно скромны, то создадим приложение с собственным окном и кнопкой. По нажатии на кнопку будет выполнена наша программа, а ее результат занесем в список.

Поместим в раздел PREDICATES объявление своих предикатов:

predicates

```
  win_mywin_eh : EHANDLER
  nondeterm equation(WINDOW)
  nondeterm znach(integer)
clauses
```

Обратим внимание, что мы ввели аргумент типа WINDOW в объявление предиката EQUATION, поскольку по логике программы мы должны получить доступ к окну-родителю по отношению к списку, в который помещаются результаты выполнения программы.

Вызов предиката EQUATION помещен в обработчик события нажатия на кнопку:

```
%BEGIN mywin, idc_task _CtlInfo
  win_mywin_eh(_Win,e_Control(idc_task,_CtrlType,_CtrlWin,_CtlInfo),0):-
!,
        equation(_Win),!.
%END mywin, idc_task _CtlInfo
```

Наконец, описание предикатов EQUATION и ZNACH помещено в свободную часть кода в разделе CLAUSES:

```
%END_WIN mywin
      EQUATION(_Win):-
                          znach(X1),
                          znach(X2),
                          znach(X3),
                          znach(X4),
                          znach(X5),
                         X2+X3+X4<=2,
                         X2+X4+X5<=1,
                         X3+X4+X5>=1,
                         X1+X5<=1,
                        WinHandle=win_GetCtlHandle(_Win,idl_L1),
                        bound(X1),
                        str_int(S1,X1),
                      lbox_Add(WinHandle,0,S1),
                         str_int(S2,X2),
                      lbox_Add(WinHandle,0,S2),
                         str_int(S3,X3),
                      lbox_Add(WinHandle,0,S3),
                         str_int(S4,X4),
                      lbox_Add(WinHandle,0,S4),
                         str_int(S5,X5),
                      lbox_Add(WinHandle,0,S5).


znach(B):-B=0; B=1.
```

Как видно, пришлось добавить несколько команд, чтобы занести значения X1,...,X5 в список.

Рассмотрим пример какой-нибудь одной из подобных задач.

Задача. Известно, что

1) Если *A* виновен и *B* невиновен, то *C* виновен;
2) *C* никогда не действует в одиночку;
3) *A* никогда не действует вместе с *C*;
4) По крайней мере один из тройки виновен.

Требуется найти хотя бы одного виновного.

Для решения этой задачи нужно составить систему логических условий (уравнений).

Для составления логических уравнений прежде всего нужно ввести переменные задачи, принимающие значения 1 или 0. Обозначим:

X1– *A* виновен;
X2 –*B* виновен;
X3 –*C* виновен;

Выражение *A* V *B* означает, что верно либо *A*, либо *B*, либо и *A*, и *B*.
Выражение *A*&*B* означает, что верно и *A*, и *B*.
Выражение Если *A*, то *B* означает, что либо *A* ложно, либо верно *B*;
Выражение *не-A* означает, что *A* ложно.

Итак, составим выражения для условий задачи. Для условия 1) имеем
Если X1 & не-X2 то X3.
Для условия 2) имеем выражение
Если X3 то X1 V X2.
Для условия 3) составим выражение:
Если X1 то не-X3.

Теперь этим выражениям следует придать алгебраический вид. У нас есть в распоряжении следующие эквивалентности:

A V *B* = *A*+*B* >= 1; и *A* V *B* = *A*+*B*-*A***B*,
A & *B* = *A*+*B* >= 2,
Если *A* То *B* = *B*-*A* >= 0,
не-A = 1- *A*.

Итак, из первого условия получаем

X3-(X1*(1-X2)) >= 0.

Из второго условия получаем

X1+X2-X1*X2- X3 >= 0.

Наконец, из третьего получаем

1-X3-X1 >= 0.

Остается ввести условие

X1+X2+X3>=1.

Таким образом, получили задачу, аналогичную рассмотренной в теоретической части. Ее решением является: *B* виновен.

5.3. РАБОТА С ВНЕШНИМИ БАЗАМИ ДАННЫХ

Данные хранятся в базе данных в форме записей. Запись состоит из полей, представляющих различные типы данных. В Прологе необходимо объявить структурированный тип для записей базы данных – функтор. Пусть в нашей учебной базе содержатся сведения о футбольных командах; полями таких записей будут название команды, страна и международный рейтинг. Объявим тип записей так:

```
DOMAINS
name=symbol
country=symbol
rate=integer
teams=teams(name, country, rate)
```

Раздел DOMAINS в Прологе используется для объявления нестандартных или пользовательских типов. Нами объявлены следующие типы: name, country, rate, teams. Наиболее интересен последний тип, который называется функтором. Примером индивидуальной записи такого типа может служить следующая:

```
teams(милан, италия, 75).
```

Теперь начнем рассматривать команды для работы с базами данных. В Прологе есть два типа баз данных: внешние и внутренние. Последние целиком грузятся в оперативную память, а потому размеры таких баз данных невелики. Поэтому рассмотрим внешние базы данных. Все команды для работы с внешними базами данных делятся на четыре группы.

Первая группа: общие команды для работы с базами данных. Сюда относятся команды создания, открытия, закрытия, удаления, сжатия, копирования и др.

Вторая группа: команды для работы с цепочками записей. Под цепочкой (chain) понимается последовательность записей, имеющих один и тот же тип. Например, выше мы ввели тип teams для записей базы данных, но могли ввести и другие типы, так что получили бы несколько цепочек соответственно.

Третья группа: команды для работы с индивидуальными записями.

Четвертая группа: команды для работы с B-деревьями. B-деревья предназначены для быстрого поиска данных по ключам. В качестве ключей могут использоваться, например, названия команд.

Начнем с первой группы.

1) Создание базы данных выполняется командой
   db_create(селектор, имя_файла, место)

Селектор является объектом типа **db_selector** и объявляется в разделе **GLOBAL DOMAINS**. Этот раздел должен быть единственным в приложении. Однако система сама его создает в файле с расширением **PRE**, который она подключает к программе с помощью команды **#include.** Вам придется открыть этот PRE-файл и сделать добавочную запись, как будет объяснено ниже. *Селектор* используется в качестве программного имени базы данных. *Имя файла* – это имя файла на диске. *Место* – это один из следующих стандартных спецификаторов:

in_file – база данных создается в файле и сохраняется в файле;
in_memory – база данных создается в памяти ЭВМ;
in_ems – для создания базы данных используется расширенная память. Мы будем использовать вариант **in_file**.

2) Открытие базы данных выполняется командой
   db_open(селектор, имя_файла, место).
3) Закрытие базы данных выполняется командой
   db_close(селектор).
4) Удаление базы данных выполняется командой
   db_delete(имя_файла, место).

Заметим, что удалять можно только закрытую базу. Команда создания базы данных одновременно делает ее открытой.

Пример.

```
DOMAINS

        db_selector = mydba

 GOAL
        db_create(mydba, “db.bin”,in_file),
      % ..... какие-то действия
        db_close(mydba),
        db_delete(“dd.bin”, in_file).
```

5) Для открытия поврежденной базы данных используется команда
   db_openinvalid(селектор, имя_файла, место).

Повреждение базы может возникнуть при сбое питания и открытой базе, некорректном завершении работы.

6) Для проверки факта обновления базы данных применяется команда
   db_update(селектор).

7) Следующая команда выполняет безопасное сохранение базы данных на диске; при этом база данных регистрируется как valid:

   db_flush(селектор).

8) Последние команды этой группы:
   db_statistics(селектор,число_записей, размер_оперативной_памяти, размер_БД_в_байтах, свободная_память_на_диске).

Эта команда выдает важную информацию. Здесь

число_записей имеет тип ULONG и получает значение, равное числу записей в базе;
размер_оперативной_памяти имеет тип ULONG и определяет число записей, загруженных в текущий момент в оперативную память;
размер_БД_в_байтах имеет тип ULONG;
свободная_память_на_диске имеет тип ULONG и определяет доступную (незанятую) память на внешнем диске.

Наконец, весьма полезна команда

9) **db_garbagecollect**(селектор),

которая собирает в кучу все пустые фрагменты в базе данных, т.е. предупреждает фрагментацию пространства памяти на диске на множество мелких незанятых участков.

Переходим теперь ко второй группе команд. Отметим, что в одной и той же базе может быть несколько различных цепочек.

10) Команда выбора следующей по порядку цепочки:

   db_chains(селектор, Chain).

Эта команда присваивает переменной Chain имя очередной цепочки. Если цепочек больше нет, то возникает возврат в ближайшую точку ветвления.

11) Следующая команда возвращает ссылку на первую запись в цепочке:

chain_first(селектор, имя_цепочки, ссылка).

Отметим, что ссылка – это переменная адресного типа. По данной ссылке можно получить и саму запись.

12) Получение ссылки на следующую запись выполняет команда

chain_next(селектор, ссылка_текущая, ссылка_следующая).

Заметим, что текущая ссылка должна быть задана.

13) Следующий предикат отыскивает и возвращает ссылку на указанную запись:

chain_terms(селектор, имя_цепочки, тип_цепочки, запись, ссылка).

Пример.

```
DOMAINS
 db_selector=mydba
 dbdom = city(symbol,symbol)

 GOAL

db_open(mydba,"d:\\work\\city.dat",in_file),
db_chains(mydba,Chain),
chain_terms(mydba,Chain,dbdom,city("London","England"),Ref),
write("REF=",Ref).
```

Обратим внимание, что при указании маршрута к файлу БД наклонные черты пишутся парами. Приведенный фрагмент программы отыскивает ссылку на запись **city("London","England")** и при удачном исходе поиска печатает ее на экране (в режиме DOS).

14) Следующая команда добавляет запись в начало цепочки:

chain_inserta(селектор, имя_цепочки,тип_записи,запись,ссылка).

Последний аргумент не должен быть задан и получает значение. Пример:

chain_inserta(mydba, chain1,dbdom,city("Minsk","Belarus"),Ref).

Команда

chain_insertz(селектор, имя_цепочки,тип_записи,запись,ссылка)

вставляет запись в конец цепочки, а команда

chain_insertafter(селектор,тип_записи,ссылка,запись,ссылка_новая)

вставляет **запись** после записи с указанной **ссылкой** и возвращает новую ссылку на вставленную запись. Последний аргумент не должен иметь значения.

15) Последняя команда данной группы удаляет всю цепочку:
chain_delete(селектор, имя_цепочки).

Рассматриваем теперь команды третьей группы.

16) Следующая команда удаляет запись из цепочки по ее ссылке:
term_delete(селектор, имя_цепочки, ссылка).

17) Следующая команда заменяет текущую запись, определяемую ссылкой, на новую и имеет такой формат:

term_replace(селектор, тип_записи, ссылка, новая_запись).

18) Следующая команда (обратите на нее особое внимание) возвращает саму запись по ее ссылочному номеру:

ref_term(селектор, тип_записи, ссылка, запись)

Остается рассмотреть последнюю четвертую группу команд для работы с B-деревьями, ускоряющими поиск данных.

19) Команда создания дерева имеет такой формат:

bt_create(селектор, имя_дерева, селектор_дерева, длина_ключа, степень_ветвления_вершин).

Здесь **селектор_дерева** не должен быть задан и получает значение в результате выполнения команды. Остальные параметры должны быть заданы. **Длина_ключа** имеет тип UNSIGNED и определяет максимальное число символов в ключе; параметр **степень_ветвления_вершин** имеет тип UNSIGNED, и рекомендуется устанавливать его равным 3 или 4.

20) Команда **bt_close**(селектор,селектор_дерева) закрывает дерево.

Пример.

```
DOMAINS
Db_selector= mydba

GOAL

db_open(mydba, “dd.dat”,in_file),
bt_create(mydba,”bt_cities”, Btree_Sel,10,4),
%  какие-то действия
 bt_close(mydba,Btree_Sel),
 db_close(mydba).
```

21) Следующая команда удаляет B-дерево:

bt_delete(селектор, имя_дерева).

22) Для открытия B-дерева используется команда:
bt_open(селектор, имя_дерева,селектор_дерева).

Эта команда использует в качестве входных аргументов первые два. Селектор дерева возвращается в качестве результата.

23) Команда закрытия дерева имеет такой вид:
bt_close(селектор,селектор_дерева).

Основными операциями на дереве являются операции с ключами.

24) Для вставки ключа в дерево используется команда
key_insert (селектор,селектор_дерева, ключ, ссылка).

Все аргументы должны быть заданы. Это значит, что запись уже должна содержаться в базе данных и на нее существует ссылка. Ссылку, в частности, можно получить при вставке записи в базу данных командой chain_inserta.

25) Удаление ключа из дерева выполняет команда

key_delete(селектор,селектор_дерева, ключ, ссылка) .

Опять же все аргументы должны быть заданы.

26) Для получения ссылки по значению ключа используется команда
key_search(селектор,селектор_дерева, ключ, ссылка).

Первые три аргумента должны быть заданы, последний возвращается системой. Если в дереве нет указанного ключа, то возникает состояние неудачи и выполняется возврат к предыдущей точке ветвления.

27) Следующая команда возвращает значение ключа и ссылки для текущей записи:

key_current(селектор,селектор_дерева, ключ, ссылка).

Первые два параметра должны быть заданы.

28) Наконец, последней командой рассматриваемой группы является команда

key_first(селектор,селектор_дерева, ссылка) .

Эта команда возвращает ссылку на первый ключ в дереве, хранящийся в его вершине.

Итак, мы рассмотрели большую часть команд для работы с внешними базами данных. Перейдем к описанию практической части работы.

Пусть необходимо построить окно, отображающее меню для работы с внешней базой данных. Меню должно содержать следующие пункты:

1. Создать/Открыть/Удалить БД.
2. Ввод данных.
3. Удаление данных.
4. Поиск данных.

Приведем пример реализации этих пунктов. Для создания БД будем использовать фиксированное имя «lab3.bin». Профиль БД – футбольные команды. Определяем тип записей:

```
DOMAINS
    name=symbol
    country=symbol
    rate=integer
   teams=teams(name, country, rate)

  %  db_selector=mydb      объявлять не надо! т.к. это объявление нужно
  %                        вставить в подсоединяемый файл  .pre

  CLAUSES
  create:-
     existfile(“lab3.bin”),
     db_create(mydb,”lab3.bin”,in_file),
     bt_create(mydb,”treeteem”,Bsel,10,3),
     dlg_MessageBox(“Info”,”Created”,mesbox_iconInformation,
```

```
                    mesbox_buttonsOK,
                    mesbox_defaultFirst, mesbox_suspendApplication),
    bt_close(mydb,Bsel),
    db_close(mydb).
```

create:-
dlg_MessageBox(“Info”,”SuchFilealreadyexists”,mesbox_iconInformation, mesbox_buttonsOK, mesbox_defaultFirst, mesbox_suspendApplication).

Прежде всего, заметим, что объявление **db_selector=mydb** не надо делать внутри программы, так как оно подключается в программу через инструкцию **#include lab1.inc**. Поэтому откройте файл lab1.inc и установите инструкцию нужным образом:

```
global domains
DB_SELECTOR = browselist_db; mydb      % For treebrowser tool

FILE = fileselector1; fileselector2 % to be edited
```

Файл LAB1.INC создает сама система, используя имя приложения; в данном случае имя приложения – LAB1.

В этом примере задействована команда dlg_MessageBox. Первый параметр – название выводимого окна. Второй – содержание сообщения в окне. Третий – тип значка. Четвертый – отображаемые кнопки. Пятый – значение кнопки, возвращаемое по умолчанию. Шестой – тип действий, выполняемых до нажатия на кнопку. Приведенный фрагмент создает пустую базу данных.

Рассмотрим теперь фрагмент добавления записи в открытую базу данных. Данные для записи следует брать из полей редактирования, размещенных в окне. Вот фрагмент программы, который это делает:

```
%BEGIN mywin, id_add_rec
 win_mywin_eh(_Win,e_Menu(id_add_rec,_ShiftCtlAlt),0):-!,
   WinHandle1=win_GetCtlHandle(_Win,id_edit1),
   Name=win_GetText (WinHandle1),
   WinHandle2=win_GetCtlHandle(_Win,id_edit2),
   Country=win_GetText (WinHandle2),
   WinHandle3=win_GetCtlHandle(_Win,id_edit3),
   SRate=win_GetText (WinHandle3),
   str_int(Srate,Rate),
   chain_inserta(mydb,"chteems",teams,teams(Name,Country,Rate),Ref),
   bt_open(mydb,"Treeteams",BSel),
   key_insert(mydb,Bsel,Name,Ref),
   bt_close(mydb,BSel), !.
```

Снова заметим, что добавление записей может производиться только в открытую базу. Вообще говоря, необходимо предусмотреть проверку, что база открыта в момент добавления записей. Предполагаем, что пользователь обеспечит правильную последовательность операций над базой данных.

ВНИМАНИЕ! Прежде всего обеспечьте пункт меню для закрытия БД, поскольку при выходе из приложения с открытой базой последняя помечается как INVALID. Такую базу можно открыть только с помощью команды **openInvalid**.

Последнее, что будет здесь рассмотрено – это операция поиска. Для программирования этой операции нужно ввести ключ поиска в поле редактирования. Ключом в нашем примере является название команды в поле Edit1. Фрагмент программного кода для обработки запроса на поиск приведен ниже.

```
%BEGIN mywin, idr_find
  win_mywin_eh(_Win,e_Menu(idr_find,_ShiftCtlAlt),0):-!,
   WinHandle1=win_GetCtlHandle(_Win,id_edit1),
    Name=win_GetText (WinHandle1),
     bt_open(mydblab,"Treeteams",BSel),
     key_search(mydb,Bsel,Name,Ref),
     ref_term(mydb,teams,Ref,Term),
     Term=teams(_,Country,Rate),
     WinHandle2=win_GetCtlHandle(_Win,id_edit2),
     win_SetText (WinHandle2,Country),
     WinHandle3=win_GetCtlHandle(_Win,id_edit3),
      str_int(Srate,Rate),
      win_SetText (WinHandle3,Srate),
      bt_close(mydb,BSel),
         !.
%END mywin, idr_find
```

Итак, мы рассмотрели основные пункты по реализации управления БД.

5.4. КЛАССЫ. СОЗДАНИЕ КЛАССА ДИАЛОГА

Прежде всего, познакомимся со способами объявления и реализации классов. Для этого воспользуемся простейшим примером.

```
CLASS myCLASS
   PREDICATES
   Show
ENDCLASS myCLASS

 IMPLEMENT myCLASS

    Show:- dlg_ask("Hello from class",["CANCEL"]).

 ENDCLASS myCLASS
```

В этом примере дано описание и реализация класса myCLASS. В описании класса, которое должно всегда предшествовать его реализации, объявлен единственный предикат show (без аргументов). В реализации класса в разделе CLAUSES представлен клоз, выполняющий данный предикат. Составленные описание и реализация класса сами по себе ничего не выполняют. Для того чтобы использовать класс, нужно сгенерировать объекты – носители свойств и методов класса. Например, подобным фрагментом программы для создания объекта мог быть следующий:

```
Create_object:-
  O= myCLASS::new,
  O:Show,
  O:delete.
```

Строка **O=myCLASS::new** осуществляет создание объекта с именем O, причем в этом случае вызывается конструктор класса **new**. Строка **O:Show** вызывает метод Show данного класса, который, как следует из его реализации, выводит на экран диалоговое сообщение. Наконец, строка **O:delete** используется для уничтожения созданного объекта класса MyDialog. Разумеется, создание и использование объектов можно связать с любым событием, возникающим при работе приложения.

Теперь создадим собственный диалоговый класс. Прежде всего, в окне проекта выберем вкладку DIALOG и нажмем кнопку **new**. Появится редактор диалога, показанный на рисунке ниже. Введем идентификационное имя : myDlg для диалога и его программный спецификатор :idd_mydlg.

После нажатия **OK** появится само окно диалога, на котором разместим кнопку из панели визуальных компонентов.

Теперь покажем, как запустить диалоговое окно на выполнение. Для этого с помощью CODE EXPERT найдем фрагмент, где определен предикат создания диалогового окна myDlg. Этот фрагмент имеет следующий вид:

constants

```
%BEGIN mydlg, CreateParms, 13:20:01-29.7.2002, Code automatically updated!
  dlg_mydlg_ResID = idd_mydlg
  dlg_mydlg_DlgType = wd_Modal
  dlg_mydlg_Help = idh_contents
%END mydlg, CreateParms

predicates

  dlg_mydlg_eh : EHANDLER
  dlg_mydlg_handle_answer(INTEGER EndButton,DIALOG_VAL_LIST)
  dlg_mydlg_update(DIALOG_VAL_LIST)

clauses

  dlg_mydlg_Create(Parent):-

%MARK mydlg, new variables

        dialog_CreateModal(Parent,dlg_mydlg_ResID,"MyDlg",
                [
%BEGIN mydlg, ControlList, 13:20:01-29.7.2002, Code automatically updated!
%END mydlg, ControlList
                ],
                dlg_mydlg_eh,0,VALLIST,ANSWER),
        dlg_mydlg_handle_answer(ANSWER,VALLIST).

  dlg_mydlg_handle_answer(idc_ok,VALLIST):-!,
        dlg_mydlg_update(VALLIST).
  dlg_mydlg_handle_answer(idc_cancel,_):-!.  % Handle Esc and Cancel here
  dlg_mydlg_handle_answer(_,_):-
        errorexit().

  dlg_mydlg_update(_VALLIST):-
%BEGIN mydlg, Update controls, 13:20:01-29.7.2002, Code automatically updated!
%END mydlg, Update controls
        true.
```

Для того чтобы воспользоваться этим описанием, следует в каком-нибудь подходящем месте программы вызвать предикат **dlg_mydlg_Create(Parent)**, помня о том, что Parent есть указатель родительского окна и имеет тип WINDOW. Теперь создадим собственный очень простой диалоговый класс:

```
CLASS MyDialog
   PREDICATES
   procedure new(Window)
   show
ENDCLASS MyDialog

  IMPLEMENT MyDialog
   CLAUSES
     new(Wnd):-dlg_mydlg_Create(Wnd).
     show:-dlg_ask("hello",["OK"]).
ENDCLASS MyDialog
```

В приведенном классе имеется одна специфическая строка:

procedure new(Window),

которая указывает на объявляемый собственный конструктор класса. Спецификатор **procedure** указывает, что данный предикат никогда не завершается неудачей и не имеет альтернативных определений (запомните это!). Мы видим, что новый конструктор класса принимает в качестве аргумента программный указатель родительского окна. Покажем, как используется это определение диалогового класса в программе:

```
%BEGIN Task Window, e_Create
  task_win_eh(_Win,e_Create(_),0):-!,
%BEGIN Task Window, InitControls, 12:59:10-29.7.2002, Code automatically updated!
%END Task Window, InitControls
%BEGIN Task Window, ToolbarCreate, 12:59:10-29.7.2002, Code automatically updated!
        tb_project_toolbar_Create(_Win),
        tb_help_line_Create(_Win),
        O=MyDialog::new(_Win),
        O:show,
        O:delete,
        %dlg_mydlg_Create(_Win),
%END Task Window, ToolbarCreate
```

```
ifdef use_message
        %msg_Create(100),
enddef
```

Это описание напоминает предыдущее, так что какие-либо комментарии излишни. Таким образом, создание диалогового окна связано с созданием основного окна приложения и приводит к результату, изображенному на рис. 5.10.

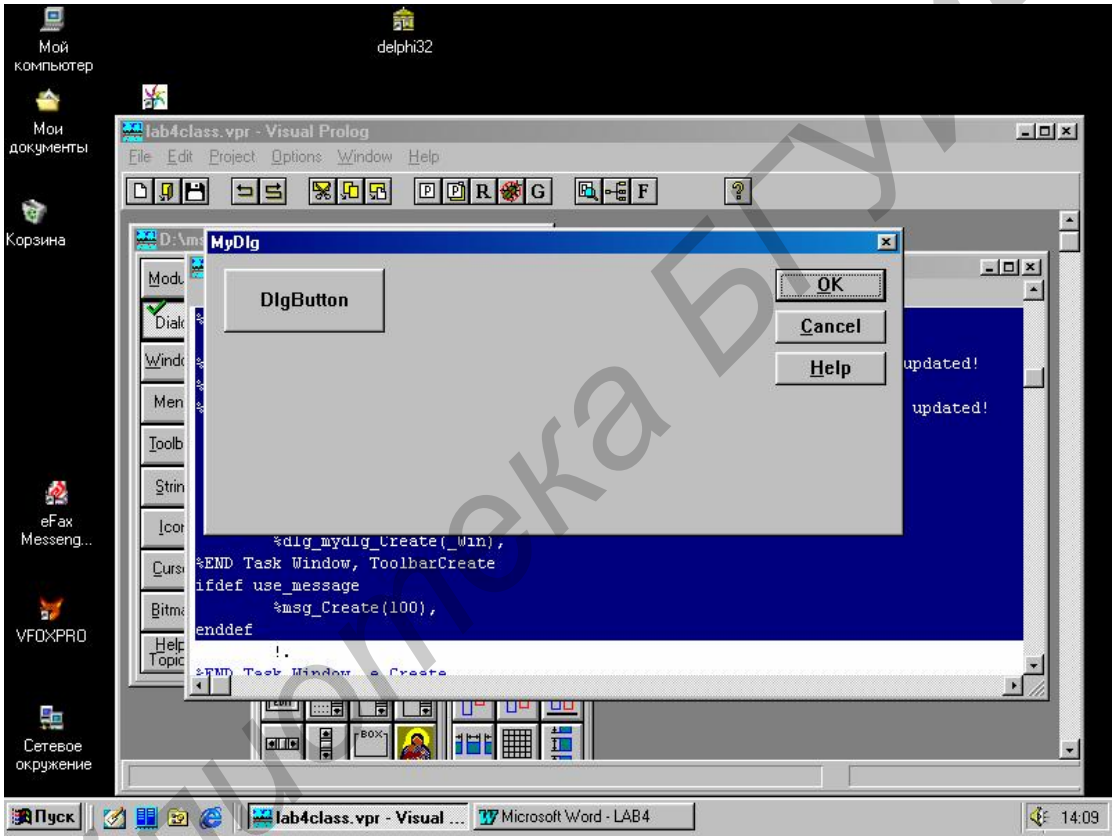

Рис. 5.10

Познакомимся теперь с весьма полезным свойством объектов: возможностью содержать в них внутренние базы данных. Таким образом, один объект – это одна база данных. Прежде всего, научимся объявлять предикаты внутренних баз данных. Дадим пример подобного объявления:

```
CLASS MyDialog
   PREDICATES
   procedure new(Window)
  nondeterm  show(integer)
   add(integer,string)
ENDCLASS MyDialog
```

```
IMPLEMENT MyDialog

  FACTS

  rec(integer,String)

   CLAUSES
     new(Wnd):-dlg_mydlg_Create(Wnd).
     Show(N):-
     rec(N,S),
dlg_ask(S,["OK"]).
Show(_):-
 Dlg_ask("No such a record encountered",["OK"]).

  Add(X,Y):-assertz(rec(X,Y)).

ENDCLASS MyDialog
```

Теперь мы расширили объявление диалогового класса, включив в него два новых предиката **show(integer), add(integer,string**:). Но сначала отметим более важное обстоятельство: мы добавили раздел **FACTS** в часть реализации класса. Раздел **FACTS** содержит имена предикатов (записей) внутренней базы данных.

В этом разделе объявлен один-единственный предикат **rec**(integer,string), первый аргумент которого содержит номер записи, а второй – саму строковую часть записи.

Для добавления предикатов (записей) базы данных в память используется предикат

assertz

(см. использование этого предиката в контексте вышеприведенного программного фрагмента).

Для удаления записи из базы данных используется предикат

retract(rec(1,"hello")).

Заметим, что в скобках указывается имя и аргументы удаляемого предиката из раздела **FACTS**. Если нет ни одного подходящего для удаления предиката, то возникает состояние неудачи и откат. Если нужно удалить несколько записей, то применяется предикат

retractall(rec(1,_)),

который в данном контексте удаляет все предикаты rec, первым аргументом которых является число 1. Чтобы удалить вообще все записи rec, нужно выдать команду

retractall(rec(_,_)).

Рассмотрим теперь, как выполняется предикат **show**. Во-первых, имеется два альтернативных определения этого предиката (обратите внимание, что ввиду этого факта данный предикат имеет тип nondeterm). Входом в первое определение является номер N искомой записи. Если такая запись имеется (успешно срабатывает предикат базы данных **rec**(N,S)), то текст S записи будет выведен в диалоговом сообщении. В противном случае будет выполняться альтернативный вариант предиката **show**(_), вместо аргумента которого указан знак подчеркивания, означающий, что аргумент может быть любым. В этом альтернативном определении осуществляется вывод диалогового сообщения об отсутствии требуемой записи.

Основной фрагмент программы теперь имеет такой вид:

```
%BEGIN Task Window, ToolbarCreate, 15:39:32-29.7.2002, Code automatically updated!
        tb_project_toolbar_Create(_Win),
        tb_help_line_Create(_Win),
        O=my1::new(_Win),
        O:add(1,"One"),
        O:add(2,"Two"),
        O:Show(1),
        O:show(4),
        O:delete,
%END Task Window, ToolbarCreate
```

Обратим внимание на особенность диалоговых окон. Если окно создается с типом Modal (для чего нужно установить соответствующий флажок), то команда dlg_ask объекта не в состоянии отобразить сообщение на таком окне. Это нужно иметь в виду при работе с объектами.

Покажем еще одну важную возможность – передачу объектов в качестве аргументов предикатов. Такая возможность реализована в демонстрируемой программе с учетом небольших изменений:

```
IMPLEMENT my1
 FACTS
  rec(integer,string)
   CLAUSES
```

```
    new(Wnd):- win_mywin_Create(Wnd).
    add(X,Y):-assertz(rec(X,Y)).

    show(N):-
      rec(N,S),
      dlg_ask(S,["OK"]).
      show(_):-dlg_ask("Not FOUND",["OK"]).
  ENDCLASS my1

predicates

  task_win_eh : EHANDLER
 nondeterm  rep(my1)
constants
```

Добавленный предикат выделен жирным шрифтом. Единственным аргументом этого предиката является объект типа **my1**. Его определение таково:

rep(O):-O:show(1).

Обратим внимание, что в качестве аргумента данный предикат получает ссылку на объект типа **my1**. Вот пример его использования:

```
tb_project_toolbar_Create(_Win),
        tb_help_line_Create(_Win),
        O=my1::new(_Win),
        O:add(1,"One"),
        O:add(2,"Two"),
        rep(O),
        O:Show(1),
        O:show(4),
        O:delete,
    .... ... ...
```

Итак, мы рассмотрели объявление и использование классов и объектов. Особое внимание надо обратить на то, что объект может содержать внутреннюю базу данных, которая удаляется вместе с объектом.

Рассмотрим задачу: создать простую экспертную систему по выбору собаки. Признаками являются: **шерсть, рост**, **уши**.

Шерсть может быть: длинная, короткая. Рост может быть: большой, средний, малый. Уши могут быть: стоячие, висячие.

Различные комбинации этих признаков соответствуют разным породам собак. Пользователь системы вводит значения признаков в полях редактирования, нажимает кнопку и получает название породы с этим признаком либо указание, что система не располагает сведениями о подходящей породе собаки.

Для простоты следует принять, что одинаковые наборы признаков соответствуют только одной породе. Вот они:

Шерсть	**Уши**	**Рост**	
Короткая	Стоят	Большой	Дог
Короткая	Висят	Средний	Боксер
Длинная	Стоят	Средний	Овчарка
Короткая	Висят	Малый	Пудель
Длинная	Висят	Большой	Сторожевая
Короткая	Стоят	Средний	Доберман

Для решения поставленной задачи нужно каждую характеристику перевести в число и найти сумму чисел для всех характеристик. Затем по этой сумме нужно найти породу собаки. При этом нужно добиться уникальности сумм. С этой целью следует закодировать характеристики числами следующим образом:

Короткая = 1, длинная = 2;
Стоят=10, висят = 20;
Средний = 100, малый = 200, большой = 300.

Далее нужно создать три предиката внутренней базы данных:

```
FACTS
 hair(symbol,integer)
 height(symbol,integer)
 ear(symbol,integer)
```

В базе данных нужно объявить также результирующий предикат:
answer(integer,symbol).

Приложение должно быть построено на оконном классе, объект которого создается и запускает окно с полями редактирования для ввода признаков и кнопки для запуска операции поиска породы собаки по признакам. Занесение фактов в базу данных должно быть привязано к запуску конструктора объекта.

5.5. РЕАЛИЗАЦИЯ НЕЧЕТКОГО ЛОГИЧЕСКОГО ВЫВОДА

Рассмотрим следующее предложение языка Пролог:

```
Suit (peter, X): -
   Big_salary (X,_)),
   Good_conditions (X,_).

   Big_salary(mailer, 0.4).
   Big_salary(officer, 0.7).
   Big_salary(bob, 0.8).
   Big_salary(artist, 1.0).

   Good_conditions(mailer, 0.7).
   Good_conditions(officer, 0.3).
   Good _conditions(bob, 0.6).
   Good_conditions(artist, 0.9).
```

В нечетком Прологе вывод осуществляется таким образом, чтобы либо найти ответ с максимальной правдоподобностью, либо «обычным образом», но, учитывая, что значение 0.5 и ниже в качестве меры истинности рассматривается как ложное. Рассмотрим, как реализовать два этих подхода.

Подход по принципу 0.5 и ниже – ложь.

В этом случае наша программа должна быть переписана следующим образом:

```
Suit (peter, X): -
   Big_salary (X,Y)),
   Y>=0.5,
   Good_conditions (X,Z),
    Z>=0.5.

   Big_salary(mailer, 0.4).
   Big_salary(officer, 0.7).
   Big_salary(bob, 0.8).
   Big_salary(artist, 1.0).

   Good_conditions(mailer, 0.7).
   Good_conditions(officer, 0.3).
   Good _conditions(bob, 0.6).
   Good_conditions(artist, 0.9).
```

Как видим, изменения в этом случае в тексте программы минимальные.

Теперь рассмотрим второй подход: поиск ветви с наибольшим значением степени истинности. Для реализации второго подхода нам потребуется воспользоваться рекурсией следующим образом:

```
Database
  Job(string)

Suit (peter, R): -
   Big_salary (X,Y)),
   Good_conditions (X,Z),
    T=Y*Z.
     T>R,
   Retractall(_), !,
   Assert(job(X)),
   Suit(peter,T).

  Suit (peter, _): -
    Job(X),
    Write (“VYBRANA RABOTA:”,X).

   Big_salary(mailer, 0.4).
   Big_salary(officer, 0.7).
   Big_salary(bob, 0.8).
   Big_salary(artist, 1.0).

   Good_conditions(mailer, 0.7).
   Good_conditions(officer, 0.3).
   Good _conditions(bob, 0.6).
```

Здесь мы используем предикат базы данных job, в котором хранится выбранная работа. Цель данной программы должна быть записана в следующем виде:

GOAL
Suit(peter,0).

Объясним наиболее сложный участок данной программы:

```
Suit (peter, R): -
   Big_salary (X,Y)),
   Good_conditions (X,Z),
    T=Y*Z.
     T>R,
```

```
Retractall(_), !,
Assert(job(X)),
Suit(peter,T).
```

Сначала последовательно выполняются предикаты:

```
Big_salary (X,Y)),
Good_conditions (X,Z),
 T=Y*Z.
```

Здесь по порядку выбирается работа, а затем выбирается соответствующая ей зарплата и условия. Вычисляется величина T=Y*Z. После этого выполняется проверка

(*****) T>R,

которая в случае удачи вызовет смену содержимого предиката базы данных job, записав в него выбранную работу и снова вызвав предикат suit (peter, T), где T – новая оценка степени истинности правила. Заметим, что хотя бы одна работа всегда будет выбрана. Если нет уже ни одной работы, для которой выполняется условие (*****), то осуществляется выход в правило

```
Suit (peter, _): -
  Job(X),
  Write (“VYBRANA RABOTA:”, X)
```

для вывода найденной работы на экран.

ЗАКЛЮЧЕНИЕ

В пособии представлены теоретические и алгоритмические аспекты построения машин вывода в экспертных системах, основанных на логических моделях знаний. Такие модели могут широко использоваться для решения задач проектирования, диагностики, управления, объяснения и пр. Многие сложные задачи комбинаторной и дискретной математики могут быть представлены с помощью логических моделей. К этой категории задач также относится проверка логической корректности юридических документов, автоматизация работы следователя, обработка текстовых запросов к базе данных и др. Задачи теории расписаний образуют еще одну важную практическую область применения логических моделей. Мы показали, как выполнить требуемую формализацию. Важным моментом является переход от алгебраических уравнений к эквивалентной системе логических формул.

Возможность использовать нечеткую логику для решения экстремальных комбинаторных задач составляет значимое теоретико-прикладное направление.

Таким образом, представленное пособие развивает аппарат математической логики применительно к реализации логической машины вывода.

Основная проблема создания машины вывода в логическом исчислении связана с решением задачи ВЫПОЛНИМОСТЬ. Эта задача является центральной не только в логике высказываний, но и в логике предикатов, поскольку все методы резолюционного типа так или иначе (пусть и в скрытой форме) решают задачу ВЫПОЛНИМОСТЬ. Основная проблема всех существующих методов решения ВЫП состоит в порождении лишних резольвент. Однако NP-полная природа этой задачи может приводить к таким ситуациям, когда все резольвенты являются необходимыми, но число их растет экспоненциально от размеров задачи. Следовательно, практическая эффективность прикладных систем математической логики, включая систему Пролог, напрямую зависит от того, насколько эффективны методы решения ВЫП. В первую очередь следует связать надежды с более эффективным использованием эвристических алгоритмов при интеграции их в систему решения задач. В разделе, посвященном логическому программированию, представлена стратегия вывода, явным образом использующая идеи Эрбрана, но более эффективная, чем стратегия метода Р. Ковальского, которая обеспечивает указанную интеграцию. Результаты практической апробации этой стратегии позволяют судить об имеющихся в ней перспективах.

Учитывая сложность задачи ВЫПОЛНИМОСТЬ в вычислительном плане, важно получить на практике приемлемые для приложений методы ее решения. Многолетний труд автора в этом направлении позволил создать метод для решения задачи о минимальном покрытии 0,1-матрицы, к которой линейно сводится ВЫПОЛНИМОСТЬ. Достоинства метода:

- верхняя граница вычислительной сложности метода в среднем оценивается как $O(m*n^2)$, где n – число строк, m – число столбцов матрицы [9];
- алгоритм ограничивает число присоединяемых столбцов-резольвент величиной n;
- алгоритм использует более эффективные резольвенты (содержащие меньшее число «1», чем в методе [9]);
- алгоритм использует эвристики для получения хороших «приближений» к точному решению.

Ограниченный объем пособия оставил за пределами проведенного рассмотрения многие интересные вопросы разработки и использования экспертных систем. Эти вопросы подробно изложены в других источниках, например [2–6]. Материал настоящего пособия может служить дополнением к отмеченным источникам.

О замеченных недостатках и опечатках автор просит сообщить по электронному адресу: **ovgerman@mail.ru**.

ЛИТЕРАТУРА

1. Саати, Т. Аналитическое планирование. Организация систем / Т. Саати, К. Керн. – М. : Мир, 1991. – 278 с.

2. Нильссон, Н. Принципы искусственного интеллекта / Н. Нильссон. – М. : Радио и связь, 1985. – 374 с.

3. Ковальски, Р. Логика в решении проблем / Р. Ковальски. – М. : Наука, 1984. – 280 с.

4. Герман, О. В. Введение в теорию экспертных систем и обработку знаний / О. В. Герман. – Минск : Дизайн-Про, 1995. – 256 с.

5. Чень, Ч. Математическая логика и автоматическое доказательство теорем / Ч. Чень, Р. Ли. – М. : Наука, 1983. – 300 с.

6. Непейвода, Н. Н. Прикладная логика / Н. Н. Непейвода. – Новосибирск, изд-во Новосиб. ун-та, 2000. – 521 с.

7. Герман, О. В. Статистически-оптимальный алгоритм для задачи о минимальном покрытии / О. В. Герман, В. Г. Найденко // Экономика и математические методы. – 1993. – №4 (29). – С. 662–667.

8. Гиндикин, С. Г. Алгебра логики в задачах / С. Г. Гиндикин. – М. : Наука, 1972. – 286 с.

9. Герман, О. В. Разрешающий принцип для задачи о минимальном покрытии 0,1-матрицы / О. В. Герман // Кибернетика и системный анализ.– 1996. – №1. – С. 135–146.

10. Кейслер, Г. Теория моделей / Г. Кейслер, Ч. Чень. – М. : Мир, 1977. – 596 с.

Св. план 2008, поз. 43

Учебное издание

Герман Олег Витольдович

ЭКСПЕРТНЫЕ СИСТЕМЫ

УЧЕБНО-МЕТОДИЧЕСКОЕ ПОСОБИЕ

Редактор *Т. Н. Крюкова*
Корректор *М. В. Тезина*
Компьютерная верстка *Е. Г. Бабичева*

Подписано в печать 15.04.2008. Формат 60х84 1/16. Бумага офсетная. Гарнитура «Таймс».
Печать ризографическая. Усл. печ. л. 5,46. Уч.-изд. л. 5,0. Тираж 100 экз. Заказ 13.

Издатель и полиграфическое исполнение: Учреждение образования
«Белорусский государственный университет информатики и радиоэлектроники»
ЛИ №02330/0056964 от 01.04.2004. ЛП №02330/0131666 от 30.04.2004.
220013, Минск, П. Бровки, 6