Министерство сельского хозяйства Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования «Дагестанский государственный аграрный университет имени М.М. Джамбулатова»

Аграрно-экономический техникум

Методические указания к практическим занятиям

ПМ.06 СОПРОВОЖДЕНИЕ ИНФОРМАЦИОННЫХ СИСТЕМ

МДК.06.04 ИНТЕЛЛЕКТУАЛЬНЫЕ СИСТЕМЫ И ТЕХНОЛОГИИ

СПО 09.02.07 Информационные системы и программирование

Махачкала 2023

Интеллектуальные системы и технологии методические указания для студентов специальности 09.02.07 Информационные системы и программирование.

Приведен теоретический и практический материал, необходимый для успешного изучения МДК.06.04 «Интеллектуальные системы и технологии».

СОГЛАСОВАНО:

Директор АЭТ подпись Магомедов Д.А.

Одобрено на заседании ПЦК общепрофессиональных, специальных дисциплин "10" марта 2023 г., протокол № 7.

Председатель ПЦК Касимовская О.О..
(подпись) (инициалы,фамилия)

СОГЛАСОВАНО:

Директор Компании Color- IT, Интернет решения Салихов А.Б.
Ф.И.О.

М.П.

СОДЕРЖАНИЕ

Предисловие

Целью освоения дисциплины «Интеллектуальные системы и технологии» является приобретение обучающимися знаний в области интеллектуальных систем, их применения при разработке интеллектуальных систем.

Основными задачами изучения дисциплины «Интеллектуальные системы и технологии», являются:

1. Типы и уровни интеллектуальных систем, их классификация и их развитие.
2. Архитектура интеллектуальных информационных систем.
3. Базы знаний информационных интеллектуальных систем.
4. Основные модели интеллектуальных систем.
5. Язык логического программирования Пролог. Логические интеллектуальные системы.
6. Разработка интеллектуальных систем.

Содержание дисциплины в соответствии с учебным планом

В соответствии с учебным планом изучение дисциплины «Интеллектуальные системы и технологии» предусматривает проведение лекционных, практических занятий и самостоятельной работы.

Общая трудоемкость дисциплины составляет __ часа.

Промежуточный контроль – экзамен (___семестр).

Содержание практических занятий

При подготовке к практическим занятиям обучающиеся самостоятельно изучают основную и дополнительную литературу, готовятконспекты по темам, предложенным преподавателем.

На практических занятиях преподаватель осуществляет контроль подготовки качества знаний обучающегося, используя: опрос, обсуждение вопросов по темам изучаемой дисциплины и отчеты по практическим занятиям.

Практическое занятие № 1. Математические основы логического программирования

Цель работы – изучить основы алгебры логики и логического вывода.

Результатом практической работы является отчет. Отчет должен содержать формулы и их решения с помощью таблиц истинности и эквивалентных преобразований

Для выполнения практической работы № 1 студент должен изучить приведенный ниже теоретический материал

Логические операции над логическими переменными

При изучении информатики (языков программирования) возникает понятие *логической* или *булевской переменной,* т. е. переменной, которая принимает одно из двух возможных значений true или false (истина или ложь, 1 или 0), называемых логическими константами.

Основные логические операции над логическими переменными:

– отрицание;
– конъюнкция (логическое «и»);
– дизъюнкция (логическое «или»);
– импликация («если...то»).

Логические операции называют также логическими (булевскими) функциями.

Знаки логических операций называют *логическими связками,* а выражения, которые получаются при использовании логических переменных и логических связок – *логическими* или *пропозициональными* (от proposition – высказывание) *формулами.*

Отрицание логической переменной *A,* обозначается $\neg A$, или подчеркиванием над буквой $\overline{A}$, или $\sim A$ (этот знак, правда, используется иногда для другой логической операции – эквивалентности): читается *«отрицание A»* или *«не A»* и означает противоположное *A* значение. В случае, когда *A* принимает значение истина (И), $\overline{A}$ принимает значение ложь (Л), и если *A* = false, то $\overline{A}$ = true.

То же самое выражают так называемой *таблицей истинности* (табл. 1).

Таблица 1. Таблица истинности для отрицания

A	***¬A***
И	Л
Л	И

Двойное отрицание переменной равно самой переменной, что может быть выражено следующей формулой: $\neg\neg A = A$. Знак равенства означает то, что таблицы истинности для выражения слева и выражения справа от знака равенства совпадают.

Конъюнкция обозначается *A* & *B*, или $A \wedge B$, или *A B,* по аналогии с умножением чисел, читается как «*конъюнкция A и B*» или «*A и B*». Конъюнкция двух логических переменных (табл. 2) истинна тогда и только тогда, когда истинны обе переменные.

Таблица 2. Таблица истинности для конъюнкции

A	***B***	***A & B***
Л	Л	Л
Л	И	Л
И	Л	Л
И	И	И

Дизъюнкция двух логических переменных обозначается $A \vee B$ *или A + B* (читается «*дизъюнкция A и B*» или «*A или B*») ложна тогда и только тогда, когда ложны обе переменные (табл. 3).

Таблица 3. Таблица истинности для дизъюнкции

A	***B***	$\boldsymbol{A \vee B}$
Л	Л	Л
Л	И	И
И	Л	И
И	И	И

Импликация обозначается $A \rightarrow B$, читается *«из A следует B»*, или *«B следует из A»*, или *«если A, то B»*, или *«A достаточное условие (достаточно) для B»*, или *«B необходимое условие (необходимо) для A»*.

A называют *посылкой* (или *антецедентом*); *B* называют *заключением* (*консеквентом*).

$A \rightarrow B$ ложна в одном только случае, когда A = true, B = false, в остальных случаях она истинна (табл. 4).

Таблица 4. Таблица истинности для импликации

A	***B***	***A→ B***
Л	Л	И
Л	И	И
И	Л	Л
И	И	И

Относительно интерпретации логических операций нужно сказать следующее. Интерпретация первых трех логических операций не вызывает сомнений, т. е. они вполне соответствуют здравому смыслу. Правда, мы иногда используем *разделяющее или:* высказывание истинно только в тех двух случаях, когда одно высказывание истинно, а другое ложно. Причем, операцию импликации можно было бы не вводить, а пользоваться вместо нее операциями отрицания и дизъюнкции.

Таблица истинности для импликации определяется исходя из следующих соображений. То, что из истинного значения посылки следует истинное значение заключения – истинно, а из истинного значения посылки следует ложное значение заключения – ложно, не вызывает интуитивного протеста. Непонятной обычно кажется истинность того, что из ложной посылки следует ложное заключение и из ложной посылки – истинное заключение (первые две строчки таблицы истинности). Это можно оправдать тем соображением, что нельзя делать неправильные выводы из правильных посылок.

Еще одна логическая операция – эквивалентность: $A \equiv B$ (иногда для этого используют знак ~): *A тождественно (или эквивалентно) B* принимает значение истина тогда и только тогда, когда значения истинности *A* и *B* совпадают. Значение ложь, когда они различные.

Алгебра логики

Под *алгеброй* понимают множество с заданными на нем операциями. Например, в качестве множества можно взять множество вещественных чисел, а в качестве операций – арифметические операции.

Пусть $B = \{0;1\}$ – двухэлементное множество, и L – множество всех возможных операций на этом множестве, алгебра $A = <B, L>$ называется *алгеброй логики*. Эти операции называют булевскими операциями или функциями алгебры логики.

Элементы двухэлементного множества можно обозначать поразному: 0, 1 или ложь, истина или true, false. Операции, которые были рассмотрены в предыдущем пункте, принадлежат множеству L.

Любая функция алгебры логики может быть выражена с помощью отрицания, конъюнкции и дизъюнкции. Проверим истинность этого утверждения. Рассмотрим в качестве примера функцию (операцию) «*исключающее или».* Обозначим ее знаком ⊕. Таблица истинности для этой функции представлена в табл. 5.

Таблица 5. Таблица истинности для «исключающего или»

A	***B***	***A*** ⊕ ***B***
Л	Л	Л
Л	И	И
И	Л	И
И	И	Л

Формула выражающая *«разделяющее или»* через дизъюнкцию конъюнкций, причем каждая конъюнкция состоит из переменных и отрицаний логических переменных A и B выглядит так:

$$A \oplus B = (\neg A \,\&\, B) \vee (A \,\&\, \neg B)$$

Причем формула $((\neg A \vee \neg B) \,\&\, (A \vee B)) \equiv ((\neg A \,\&B) \vee (A \,\&\, \neg B))$ будет истинна при любых значениях A и B, т. е. будет тавтологией. Справедливость равенства может быть проверена с помощью построения таблиц истинности.

В дальнейшем нам потребуются некоторые логические эквивалентности (функции алгебры логики, стоящие справа и слева от знака эквивалентности, имеют одинаковые таблицы истинности).

$$A \to B \equiv \neg A \vee B \quad (1)$$

$$(A \,\&\, B) \vee C \equiv (A \vee C) \,\&\, (B \vee C) \quad (2)$$

$$(A \vee B) \,\&\, C \equiv (A \,\&\, C) \vee (B \,\&\, C) \quad (3)$$

$$\neg (A \,\&\, B) \equiv \neg A \vee \neg B \quad (4)$$

$$\neg (A \vee B) \equiv \neg A \,\&\, \neg B \quad (5)$$

$$A \,\&\, \neg A \equiv \text{false} \quad (6)$$

$$A \vee \neg A \equiv \text{true} \quad (7)$$

Пропозициональные формулы

Под высказыванием (proposition) понимается утверждение, относительно которого можно сказать истинно оно или ложно, при этом оно не может быть и истинным, и ложным одновременно. Из простых высказываний с помощью логических операций, рассмотренных выше, строятся более сложные высказывания.

В исчислении высказываний рассмотренные выше символы логических (булевских) операций (функций) $\neg$, &, $\vee$, $\to$, $\equiv$ называют *пропозициональными или логическими связками.*

Логические (булевские) переменные, обозначаемые буквами латинского алфавита (иногда с индексами), называют *пропозициональными переменными (пропозициональными буквами).*

Алфавит языка исчисления высказываний состоит из пропозициональных переменных (букв), логических связок и скобок: (,).

Правильно построенные выражения языка исчисления высказываний называются *пропозициональными формулами* или просто формулами.

Все пропозициональные переменные являются формулами.

Правила построения языка исчисления высказываний задают его *синтаксис.*

Так как буквы, входящие в формулу, сами могут быть формулами, то каждую формулу можно рассматривать как *схему* бесконечного множества формул, получаемых заменой пропозициональных букв на соответствующие им формулы. Для того чтобы указать это, используют вместо термина пропозициональная формула термин *пропозициональная форма.*

Скобки (определяющие структуру формулы) могут опускаться в соответствии со следующим приоритетом логических связок: $\neg$, &, $\vee$, $\to$, $\equiv$. Любое вхождение знака $\neg$ относится к наименьшей формуле

(пропозициональной форме), следующей за ним; после расстановки всех скобок для формул, содержащих знак ¬, каждое вхождение знака & связывает наименьшие формулы, стоящие рядом с этим знаком; затем то же самое относится к знаку ∨, далее к → – импликации и потом к знаку ≡ – эквивалентности. В случае равного приоритета скобки расставляются слева направо.

Так например, пропозициональная формула $A \vee B \rightarrow C \vee D \& E$ должна пониматься как $((A \vee B) \rightarrow (C \vee (D \& E)))$, поскольку сначала выполняется (применяется) операция &, которая «старше», чем операция ∨, затем левая дизъюнкция, т. к. направление чтения формул слева направо, потом правая дизъюнкция и далее импликация.

Приписывание значений истинности (true или false) пропозициональным переменным называется *интерпретацией* этих переменных. Под *интерпретацией формулы* понимается приписывание значений истинности переменным, входящим в эту формулу.

Пропозициональная формула, истинная при любых значениях истинности входящих в нее пропозициональных переменных, т. е. истинная при любых интерпретациях, называется *тождественно истинной*, или *общезначимой пропозициональной формулой*, или *тавтологией*.

Пропозициональная формула, ложная при любых значениях истинности входящих в нее пропозициональных переменных, называется *тождественно ложной*, или *невыполнимой пропозициональной формулой*, или *противоречием*.

Пропозициональная формула, которая может принимать истинные значения при некоторых интерпретациях, называется *выполнимой*. Интерпретации, для которых пропозициональная формула принимает истинное значение, называются ее *моделями*.

Логическое следствие и логический вывод

Выявление того факта, что из множества высказываний (формул исчисления) логически следует некоторое другое высказывание (формула) и является, по существу, одной из основных задач исчисления.

U логически влечет W (*W* логически следует из *U*) в исчислении высказываний, что обозначим как $U \Rightarrow W$, если пропозициональная форма $(U \rightarrow W)$ является тавтологией.

Говорят, что множество формул $\{F_1, F_2, \ldots, F_n\}$ логически влечет F в исчислении высказываний, что будем записывать как $\{F_1, F_2, \ldots, F_n\} \Rightarrow F$, если формула $\{F_1 \& F_2 \& \ldots \& F_n\} \rightarrow F$ является тавтологией.

Для обозначения логического следования здесь используется знак $\Rightarrow$. Надо отметить, что в литературе иногда для обозначения логического следования используется знак $\models$, а знак $\Rightarrow$ используется для обозначения импликации.

U и W логически эквивалентны, что будем записывать в виде $U \Leftrightarrow W$, если пропозициональная форма $(U \equiv W)$ является тавтологией. Знак $\Leftrightarrow$ также используется для указания эквивалентных преобразований формул.

Вместо каждой буквы пропозициональной формы может быть подставлено любое высказывание естественного языка, причем одни и те же буквы должны заменяться одними и теми же высказываниями. Если сделать такую подстановку в тавтологии, то независимо от высказываний естественного языка мы получим логически истинное высказывание. Если пропозициональная формула ложна при любой интерпретации (является противоречием), то при такой подстановке будет получаться высказывание, являющееся логически ложным.

Истинность или ложность получаемых высказываний не будет зависеть от высказываний-подстановок, т. к. истинность или ложность результирующего высказывания зависит только от логической структуры пропозициональной формы, в которую они подставлялись.

В качестве примера рассмотрим высказывание: если верно, что *после снегопада крыша становится белой,* то справедливо также и следующее утверждение: *если крыша не белая, то снегопада не было.*

Обозначим буквой S утверждение «*снегопад был»,* буквой B – *«крыша белая».* Тогда утверждение о том, что «*после снегопада крыша становится белой»,* запишется в виде формулы $S \to B$. Утверждение «*если крыша не белая, то снегопада не было»* запишется так: $\neg B \to \neg S$.

То, что из первого утверждения следует второе (т. е. всегда, когда истинно первое утверждение, истинным является и второе), записывается как логическое следование: $(S \to B) \Rightarrow (\neg B \to \neg S)$, что эквивалентно тому, что формула $(S \to B) \to (\neg B \to \neg S)$ является тавтологией.

Докажем, что эта формула тождественно истина (тавтология). Это можно сделать с помощью таблицы истинности или с помощью эквивалентных преобразований (1)–(7):

$(S \to B) \to (\neg B \to \neg S) \Leftrightarrow$ по формуле (1)

$\neg (\neg S \vee B) \vee (\neg \neg B \vee \neg S) \Leftrightarrow$ по формуле (5)

$(S \,\&\, \neg B) \vee (B \vee \neg S) \Leftrightarrow$ по формуле (2)

$(S \vee B \vee \neg S) \,\&\, (\neg B \vee B \vee \neg S) \Leftrightarrow$ по формуле (7)

true & true ⇔
true

Метод резолюций

Другое рассуждение (дающее ключ к пониманию метода логического вывода, называемого методом резолюций) заключается в следующем.

Чтобы доказать, что $\{F_1, F_2, \ldots, F_n\} \Rightarrow F$, необходимо доказать, что формула $(F_1 \,\&\, F_2 \,\&\, \ldots \,\&\, F_n) \rightarrow F$ тавтология. Для того чтобы доказать, что данная формула является тавтологией, для этого достаточно доказать, что ее отрицание является противоречием, т. е. что формула $\neg(F_1 \,\&\, F_2 \,\&\, \ldots \,\&\, F_n \rightarrow F)$ – противоречие. То есть противоречием должна являться $\neg(\neg(F_1 \,\&\, F_2 \,\&\, \ldots \,\&\, F_n) \vee F)$ или формула $(F_1 \,\&\, F_2 \,\&\, \ldots \,\&\, F_n \,\&\, \neg F)$ т. е. противоречивым является множество формул Ф, состоящее из множества посылок $F_1, F_2, \ldots, F_n$ и отрицания заключения $\neg F$.

Первым шагом для применения метода резолюций служит приведение пропозициональной формулы к логически эквивалентной ей конъюнктивной нормальной форме.

Конъюнктивной нормальной формой (КНФ) называется конъюнкция (логическое «и») конечного числа дизъюнктов. Любая пропозициональная формула имеет логически эквивалентную ей КНФ.

Дизъюнктом (в англоязычной литературе называют *clause*, что означает предложение в сложносочиненном предложении) называется дизъюнкция конечного числа *литер*, т. е. пропозициональных символов или их отрицаний:

$D = (p_1 \vee p_2 \vee \ldots \vee p_n)$.

Дизъюнкты, содержащие противоположные литеры $(A \vee \neg A)$ являются тавтологиями и их можно отбросить, т. к. они не влияют на значение формы. Если в пределах одного и того же дизъюнкта встречается несколько раз одна и та же литера (например, $A \vee A$), то достаточно записать ее лишь один раз. В результате таких упрощений будет получена *приведенная* или *совершенная* КНФ.

Пустым дизъюнктом называется константа false. Если КНФ не содержит ни одного дизъюнкта (иначе *пуста)*, то она эквивалентна true.

КНФ общезначима (является тавтологией), если все ее дизъюнкты общезначимы.

Дизъюнкт невыполним в том и только том случае, если он пустой.

Понятие дизъюнкта важно в логическом программировании и, в частности, в языке Пролог, т. к. описание задачи осуществляется в терминах дизъюнктов, часто называемых (в переводах на русский) *клаузами* или *клозами.*

Исчисление высказываний как формальная теория

Рассмотренные нами методы логического вывода, доказательства истинности или ложности формул являются семантическими, т. к. используют интерпретацию формул: придание значений истинности пропозициональным переменным. Один из этих способов – построение таблицы истинности. Другой способ – доказательство того, что данная формула логически (на основе понятия логического следования) следует из некоторого множества формул.

Если пропозициональная формула логически следует из пустого множества формул, то она является тождественно истинной или тавтологией.

Еще один способ, рассмотренный нами – метод резолюций.

Другой способ доказательства, формально-логический или синтаксический, не требует для своего определения понятия интерпретации формул, он основан на построении формальной теории (формальной или дедуктивной системы).

Именно построение такой формальной системы или теории и может являться способом моделирования наших рассуждений во многих практически важных случаях.

Формальная система представляет собой набор начальных состояний или аксиом, которые могут быть описаны на некотором формальном языке, и правил вывода, позволяющих на основе аксиом и их следствий, получаемых с помощью правил вывода, строить новые и новые следствия, называемые теоремами формальной теории.

Процесс построения теорем называется формальным выводом или доказательством в формальной теории. Из школьного курса геометрии вспомним, что представляют собой аксиомы и теоремы формальной теории.

Варианты для выполнения практической работы

Решить заданные пропозициональные формы при условии что: A = False; B = True

1. A& ¬B∨ ¬¬A& ¬B→ A
2. ¬A& ¬B∨(A∨ ¬B)→ A
3. A& ¬B∨ A→ ¬B& A
4. A& ¬B∨(A→ ¬B∨ A)
5. A& ¬B→ A& ¬B∨ A
6. A& ¬B→ A∨ ¬B& A
7. A∨ ¬B& A& ¬B→ A
8. A∨ ¬B& A∨ ¬B→ A
9. A∨ ¬B& A→ ¬B& A
10. A∨ ¬B& A→ ¬B∨ A
11. A∨ ¬B→ A& ¬B∨ A
12. A∨ ¬B→ A∨ ¬B& A
13. A→ ¬B& A& ¬B∨ A
14. A→ ¬B& A∨ ¬B& A

Доказать или опровергнуть заданные равенства.

1. A& ¬B∨A = A∨B& ¬A
2. A→B∨ ¬A = (¬A& ¬B) →A
3. A∨ ¬B∨A = A&B& ¬A
4. ¬B∨A&B = ¬A∨B →A
5. ¬B∨A&B ¬A = ¬A∨ ¬B →A
6. A∨ ¬B&A = A& ¬B∨ ¬A
7. A& ¬B&A = A∨ ¬B∨ ¬A
8. ¬A&B∨A = A∨ ¬B& ¬A
9. ¬A∨B∨A = A& ¬B& ¬A
10. ¬A∨B&A = ¬A∨B&A
11. ¬A&B&A = ¬A& ¬B∨ ¬A
12. A&B∨ ¬A = ¬A∨ ¬B∨ ¬A
13. A∨B∨ ¬A = ¬A& ¬B& ¬A
14. A& ¬B∨A = A∨B& ¬A

Контрольные вопросы

1. Что такое алгебра логики?
2. Какая формула является пропозициональной?
3. Какая формула является пропозициональной?
4. В чем суть метода резолюций?

Практическое занятие № 2. Знакомство и основы работы с Visual Prolog. Структура Пролог-программы

Цель работы – ознакомиться с интерфейсом, основными правилами работы и настройками среды Visual Prolog. Получить навыки создания проектов и запуска приложений.

Результатом практической работы является отчет. Отчет должен содержать листинг программы с комментариями, результаты, выдаваемые программой на каждый из запросов.

Для выполнения практической работы № 2 студент должен изучить приведенный ниже теоретический материал.

Основы работы с Visual Prolog. Запуск Visual Prolog

Программа установки инсталлирует группу программ с пиктограммой, расположенной в меню **«Пуск»**, которая обычно используется для запуска среды визуальной разработки Visual Prolog. Впрочем, существует множество других вариантов запуска приложения в Windows, например запуск исполняемого файла VIP.EXE из папки BIN\WIN\32 (32-битная версия Windows), находящейся в основном каталоге Visual Prolog (обычно каталог называется VIP52).

Если во время закрытия среды визуальной разработки был открыт проект (PRJ или VPR файл), то при следующем запуске этот проект откроется автоматически. Если в процессе инсталляции Visual Prolog вы выбрали флажок **Associate 32-bit VDE with Project File Extensions PRJ & VPR**, то для открытия проекта достаточно дважды щелкнуть на файле с расширением prj или vpr. Среда визуальной разработки запускается и загружает выбранный проект.

Для запуска большинства примеров необходимо использовать Test Goal – утилиту среды визуальной разработки. Эта утилита может быть активизирована при помощи команды **Project | Test Goal** или комбинации клавиш <Ctrl>+<G>. Для корректного выполнения примеров с утилитой Test Goal среда визуальной разработки использует специальные настройки загружаемых проектов. *Мы рекомендуем вам создать и всегда использовать следующий специальный TestGoal-проект*.

Создание TestGoal-проекта для выполнения примеров

При использовании утилиты Test Goal для выполнения примеров и запуска программ требуется определить некоторые (не предопределенные) опции компилятора Visual Prolog. Для этого выполните следующие действия:

1. Запустите среду визуальной разработки Visual Prolog.

При первом запуске VDE (среда визуальной разработки) проект не будет загружен, и вы увидите окно, показанное на рис. 1. Также вас проинформируют, что по умолчанию создан инициализационный файл для Visual Prolog VDE.

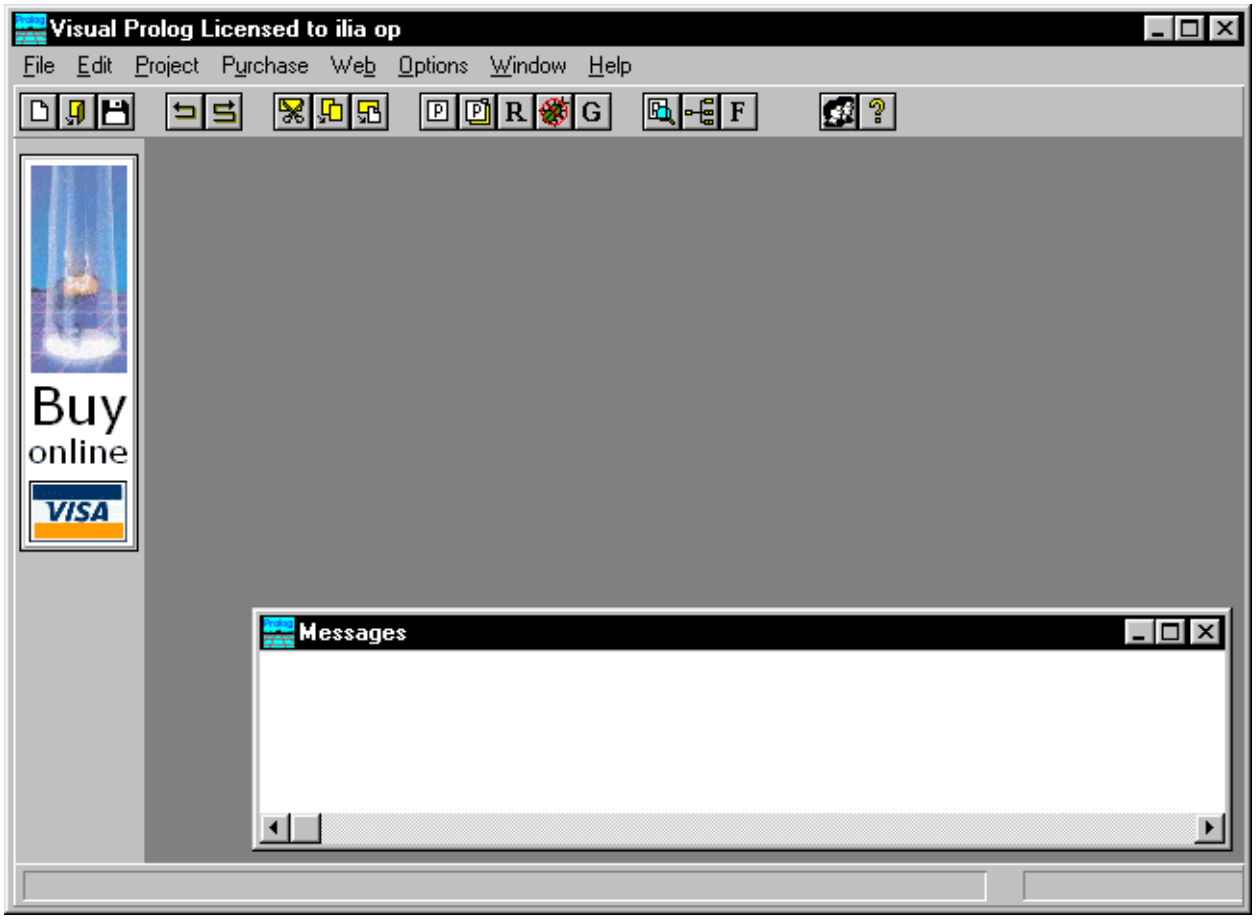

Рис. 1. Окно среды визуальной разработки после первого запуска

2. Создайте новый проект.

Выберите команду **Project | New Project**, активизируется диалоговое окно **Application Expert**.

3. Определите базовый каталог и имя проекта.

В качестве примера определим базовый каталог. Допустим, что при установке Visual Prolog в качестве корневого каталога Visual

Prolog был выбран C:\VIP. В этом случае задаем имя в поле **Base Directory** следующим образом:

C:\VIP\DOC\Examples\TestGoal

Имя в поле **Project Name** определим как «TestGoal». Также установим флажок **Multiprogrammer Mode** и щелкнем мышью внутри поля **Name of.PRJ File**. Вы увидите, что появится имя файла проекта TestGoal.prj (рис. 2).

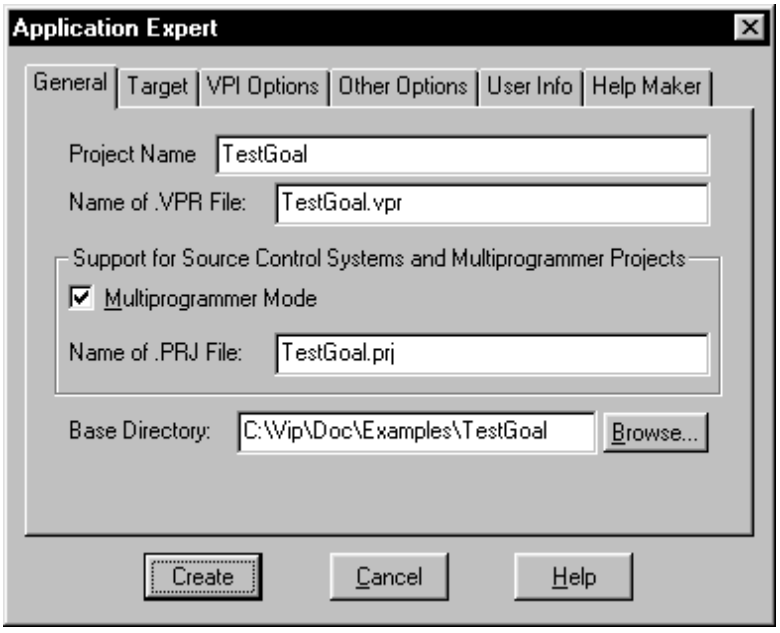

Рис. 2. Общие установки диалогового окна **Application Expert**

При создании собственных проектов рекомендуем в качестве базового каталога (Base Directory) использовать каталоги расположенные в папке **Мои документы**. Например: …\Мои документы \user \it-001 \Ivanov.

Определите цель проекта. На вкладке **Target** нужно выбрать параметры, отмеченные на рис. 3. Теперь нажмите кнопку **Create** для того, чтобы создать файлы проекта по умолчанию.

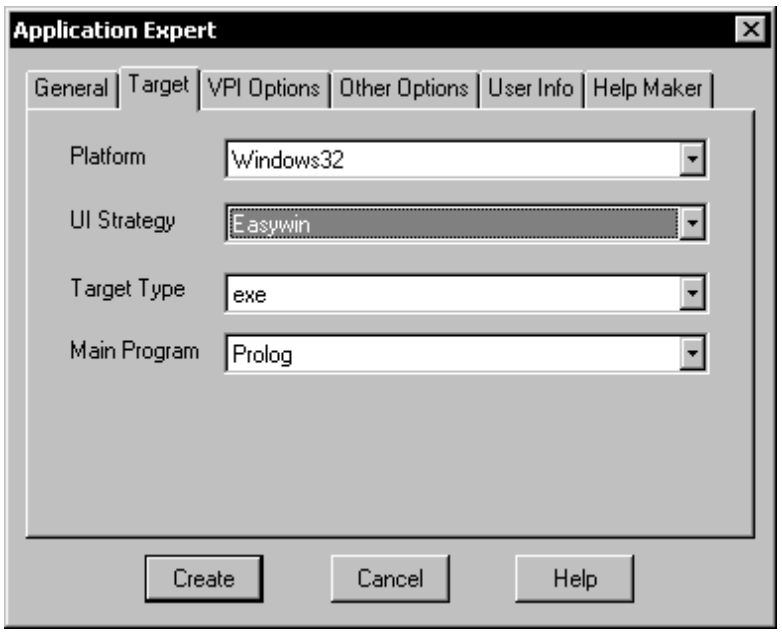

Рис. 3. Установки на вкладке **Target** диалогового окна **Application Expert**

4. Установите требуемые опции компилятора для созданного TestGoal-проекта.

Для активизации диалогового окна **Compiler Options** выберите команду **Options | Project | Compiler Options.** Откройте вкладку **Warnings.** Выполните следующие действия: снимите флажки **Non Quoted Symbols, Strong Type Conversion Check** и **Check Type of Predicates.** Это будет подавлять некоторые возможные предупреждения компилятора, которые не важны для понимания выполнения примеров. Затем нажмите кнопку **ОК**, чтобы сохранить установки опций компилятора.

В результате этих действий диалоговое окно **Compiler Options** будет выглядеть, как показано на рис. 4.

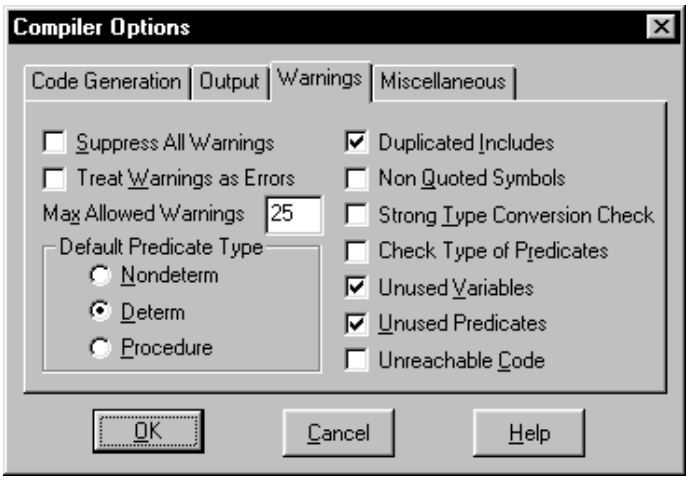

Рис. 4. Установки опций компилятора

Открытие окна редактора

Для создания нового окна редактирования вы можете использовать команду меню **File | New**, или нажать на кнопку расположенную на панели инструментов (). В результате появится новое окно редактирования с именем **noname.pro**.

Редактор среды визуальной разработки – стандартный текстовый редактор. Можно использовать клавиши управления курсором и мышь так же, как и в других редакторах. Он поддерживает команды **Cut, Copy** и **Paste, Undo** и **Redo,** которые находятся в меню **Edit.** В меню **Edit** также показаны комбинации «горячих» клавиш для этих действий. Подробное описание редактора находится в системе помощи VDE (клавиша <F1> в окне редактора).

Запуск и тестирование программы

Для проверки того, что ваша система настроена должным образом, следует напечатать следующий текст в окне:

```
GOAL
write(«Hello world»), nl.
```

В качестве приветствия можно было написать «Привет мир». Для правильного отображения букв русского языка необходимо выполнить следующие действия: нажать на кнопку  ; в появившемся окне **Выбор шрифта** необходимо в раскрывающемся списке «**Набор символов»** выбрать значение «**кириллица**»; нажать кнопку **OK**.

В терминологии языка Пролог это называется *GOAL*, и этого достаточно для программы, чтобы она могла быть выполнена. Для того чтобы выполнить **GOAL**, вам следует активировать команду **Project | Test Goal** или нажать комбинацию клавиш <Ctrl>+<G>. Если ваша система установлена и настроена правильно, то экран монитора будет выглядеть, как показано на рис. 5.

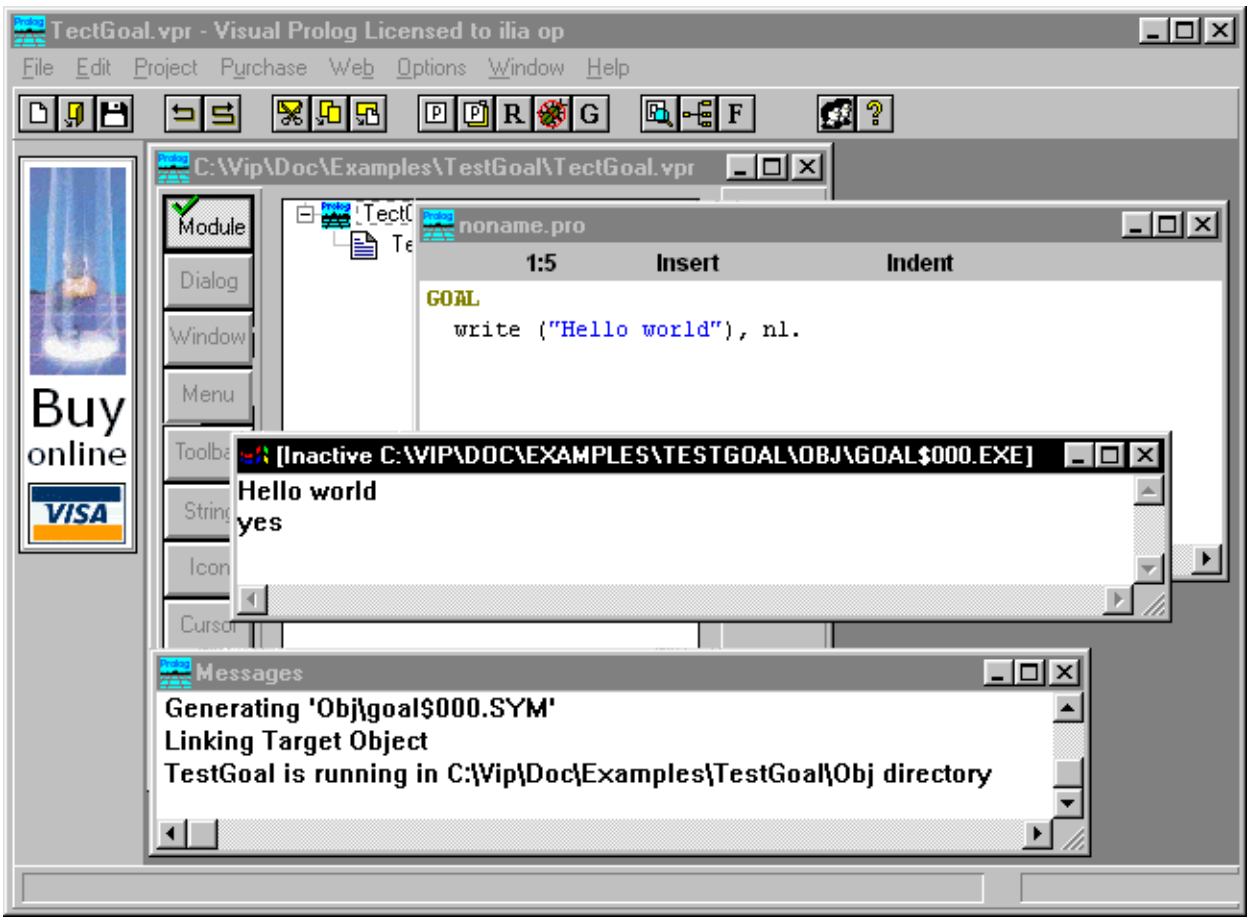

Рис. 5. Тестовая программа «Hello World»

Результат выполнения программы будет расположен вверху в отдельном окне (на рисунке оно называется **Inactive C:\Vip\Doc\Examples\TestGoal\Obj\goal$OOO.exe**), которое необходимо закрыть перед тем, как тестировать другую GOAL.

Комментарии к свойствам утилиты Test Goal

Утилита среды визуальной разработки интерпретирует GOAL как специальную программу, которая компилируется, компонуется, генерируется в исполняемый файл и Test Goal запускает его на выполнение. Эта утилита внутренне расширяет заданный код GOAL, чтобы сгенерированная программа ***находила все возможные решения и показывала значения всех используемых переменных***. Утилита Test Goal компилирует этот код с использованием опций компилятора, заданных для открытого проекта (рекомендуемые опции компилятора для TestGoal-проекта мы определили ранее).

Замечание: Утилита Test Goal компилирует только тот код, который определен в активном окне редактора (код в других открытых редакторах или модулях проектов, если они есть, игнорируется).

При компоновке исполняемого файла Test Goal не может использовать никакие глобальные предикаты, определенные в других модулях. Заметим, что утилита имеет ограничение на количество переменных, которые могут быть использованы в GOAL. На данный момент их 12 для 32-разрядной среды визуальной разработки, но это число может быть изменено без дополнительных уведомлений.

Обработка ошибок

Если вы допустили ошибки в программе и пытаетесь скомпилировать ее, то среда визуальной разработки отобразит окно **Errors (Warnings),** которое будет содержать список обнаруженных ошибок и предупреждений (рис. 6).

Дважды щелкнув на одной из этих ошибок, вы попадете на место ошибки в исходном тексте. Можно воспользоваться клавишей <F1> для вывода на экран интерактивной справочной системы Visual Prolog. Когда окно помощи откроется, щелкните по кнопке **Search,** наберите номер ошибки, и на экране появится соответствующее окно помощи с более полной информацией о ней.

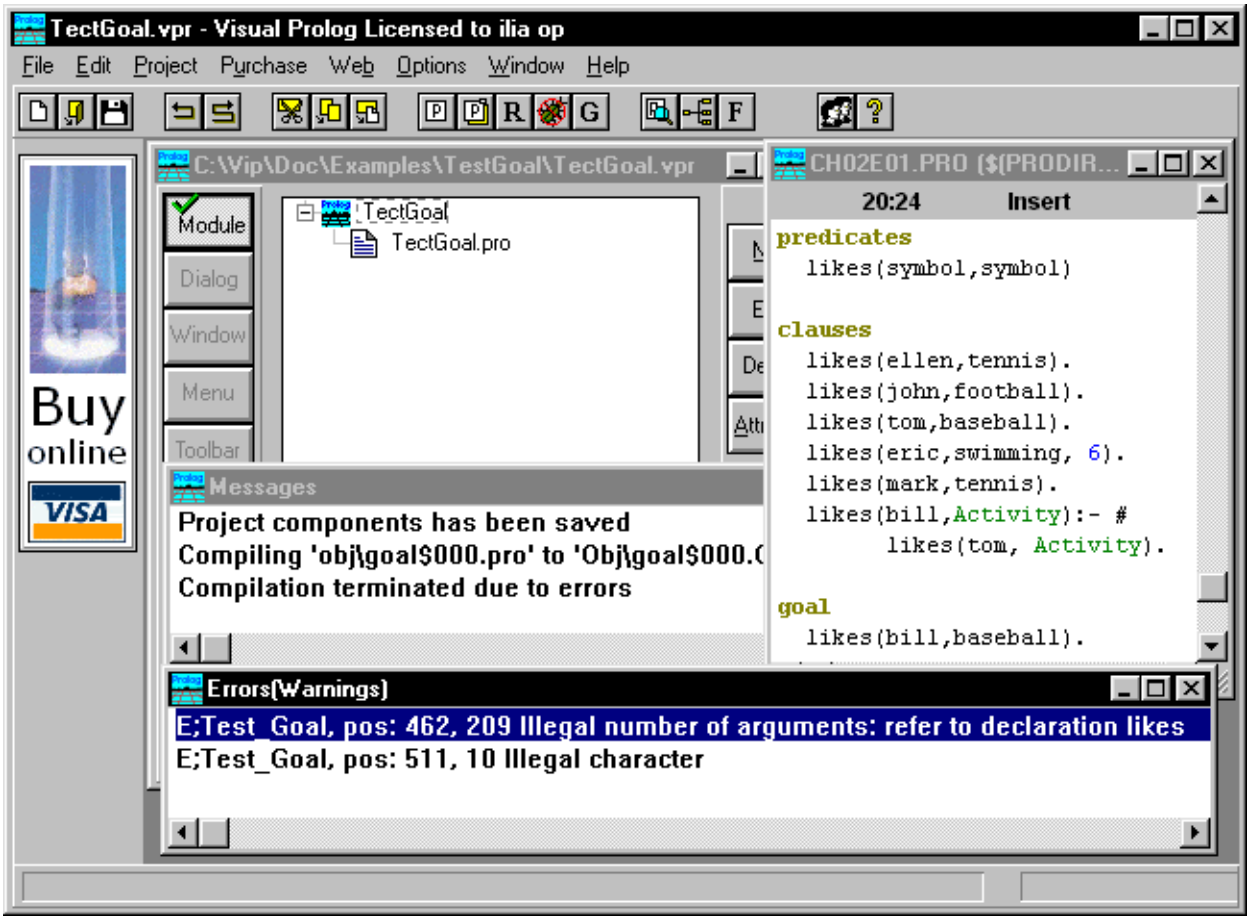

Рис. 6 Обработка ошибок

Знакомство с основами языка Visual Prolog

ПРОЛОГ – язык логического программирования (ПРОграммирование в ЛОГике), используемый для представления и манипулирования знаниями в системах искусственного интеллекта (ИИ).

Программа на языке ПРОЛОГ – набор утверждений, составляющих базу фактов и базу правил, к которым допустимо обращение с запросами, касающимися их содержимого. Запросы называются также *целевыми утверждения*.

Логическое программирование – программирование, основанное на использовании механизма доказательства теорем в логике, который позволяет выяснить, является ли противоречивым некоторое множество логических формул. При этом программа рассматривается как набор логических формул, описывающих предметную область, совместно с теоремой, которая должна быть доказана. Логическое программирование избавляет программиста от необходимости определения точной последовательности шагов выполнения вычислений (алгоритма).

Структура Пролог-программы

Программы на языке Пролог состоят из двух типов фраз: **фактов** и **правил**, также называемых предложениями.

Факты – это отношения или свойства, о которых известно, что они имеют значение «истина». Например, предложение «Билл любит цветы» устанавливает отношение между объектами Билл и цветы; этим отношением является «любит». Данный факт на языке Пролог запишется следующим образом:

любит(билл, цветы).

Правила – это связанные отношения; они позволяют Прологу логически выводить одну порцию информации из другой. Правило принимает значение «истина», если доказано, что заданный набор условий является истинным.

Пролог всегда ищет решение, начиная с первого факта и/или правила, и просматривает весь список фактов и/или правил до конца.

Задание вопросов о фактах в программе называется **запросами** к системе Пролога. Более общим термином для запроса является **цель** (goal). Пролог пытается разрешить цель (ответить на вопрос), просматривает все факты, начиная с первого до достижения последнего из них.

Запросы к системе могут быть простыми и составными (составная цель). Составная цель – это цель, включающая две или более частей; каждая часть составной цели называется подцелью.

Составная цель может быть конъюнктивной (подцель A и подцель B) или дизъюнктивной (подцель A или подцель B). Подцели отделяются друг от друга знаком «,» для конъюнкции и «;» для дизъюнкции. Используя комбинации конъюнкции и дизъюнкции можно ставить системе вопросы различной сложности.

Мы можем задавать Прологу такие же вопросы, которые мы могли бы задать вам об этих отношениях. Основываясь на известных, заданных ранее фактах и правилах, вы можете ответить на вопросы об этих отношениях, в точности так же это может сделать Пролог. На естественном языке мы спрашиваем: «Билл любит цветы?». По правилам Пролога мы спрашиваем:

любит(билл, цветы).

Получив такой запрос, Пролог мог бы ответить:

yes (да)

потому что Пролог имеет факт, подтверждающий, что это так. Немного изменив вопрос, мы могли бы спросить вас на естественном языке: «Что любит Билл?».

По правилам Пролога мы спрашиваем:

любит(билл, Что).

Заметим, что синтаксис Пролога не изменяется, когда вы задаете вопрос: этот запрос очень похож на факт. Впрочем, важно отметить, что второй объект «Что» начинается с большой буквы, тогда как первый объект «билл» – нет. Это происходит потому, что «Что» – переменная.

В Пролог переменные всегда начинаются с заглавной буквы или символа подчеркивания (_).

Получив запрос о том, что Билл любит, Пролог ответит:

Что=цветы

1 Solutions

Такой ответ получен так как Прологу известен соответствующий факт.

Так же в Прологе существует особый тип переменной – анонимная переменная, обозначаемая одним знаком подчеркивания. С ее помощью можно поставить, например такую цель: «Любой, кто любит цветы?».

любит(_, цветы).

Для написания комментариев к программе в Пролог используется знак «%» для строчного комментария и комбинация знаков «/**текст комментария**/» для блока

Программа на Пролог состоит из нескольких разделов и имеет следующую обобщенную структуру:

```
DOMAINS      /*определение типов данных*/
PREDICATES /*определение предикатов*/
CLAUSES      /*определение фактов и правил*/
GOAL         /*определение целей*/.
```

Раздел доменов – в этом разделе объявляются любые нестандартные домены, используемые для аргументов ваших предикатов. Домены в Прологе – подобно типам данных на других языках программирования. Базисные стандартные домены Пролога – char, byte, short, ushort, word, integer, unsigned, long, ulong, dword, real, string и symbol.

Раздел предикатов – объявляются предикаты и домены (типы) аргументов предикатов. Или другими словами в разделе predicates перечисляются все используемые вами предикаты (факты и правила) с описанием типов их аргументов. Имя предиката должно начинаться с символа (предпочтительно нижний регистр), сопровождаемого последовательностью символов, цифр, и символов подчеркивания, длиной до 250 символов. В именах предиката нельзя использовать пробел, знак «минус», звездочки, или наклонную вправо черту. Объявления Предиката имеют следующую форму:

```
PREDICATES
PredicateName (argument_type1, argument_type2, …)
```

Argument_type – являются или стандартными доменами или доменами, которые вы объявили в разделе доменов. Объявление домена аргумента и описание типа аргумента по сути одно и то же.

Раздел предложений (clauses) – размещены факты и правила, на основе которых Visual Prolog пытается удовлетворить цель программы.

Раздел целей – в этом разделе задается цель (запрос) вашей программы.

Приведем пример поясняющий содержание перечисленных разделов. В программе сделаем следующее: создадим домен с именем *age* целочисленного типа; опишем предикат *возраст* и предикат *профессия*, каждый из которых имеет два аргумента; в разделе предло-

жений напишем 4 факта; зададим составную цель с помощью которой можно узнать фамилии кондитеров и сколько им лет.

```
domains                          % объявили пользовательский домен
 age = integer            %  с именем age
predicates
 возраст(symbol, age)                    % объявили предикаты
профессия(symbol, symbol)      % возраст и профессия
clauses
 возраст(иванов, 25).
 возраст(петров, 40).
профессия(иванов, кондитер).
профессия(петров, строитель).
goal
% Узнаем фамилии кондитеров и сколько им лет
 профессия(Кто, кондитер),
 возраст(Кто, Сколько_лет).
```

На составленный таким образом запрос Пролог ответит буквально следующее:

Кто=иванов, Сколько_лет=25

1 Solution

Порядок выполнения работы

1. Изучить интерфейс среды визуальной разработки Visual Prolog.

2. Написать и запустить программу:

```
goal
write («Привет студентам ИТ»), nl.
```

3. Написать и запустить программу про спорт. В окне нужно напечатать следующий текст:

```
Predicates
играет (symbol,symbol)
Clauses
играет («Вася»,футбол).
играет («Петя»,футбол).
играет («Маша»,баскетбол).
играет («Ваня»,дартс).
Goal
```

играет(Кто,футбол).

В данном примере мы написали несколько фактов о том, какими видами спорта увлекаются школьники. И хотим, чтобы программа показала, кто играет в футбол.

Если на экране появится надпись **nondeterministic clause**, как на рис. 7, то это значит, что в программе произошла ошибка.

Рис. 7. Сообщение об ошибке

Это сообщение говорит о том что предикат *играет* является недетерминированным, т. е. может порождать более одного решения. В нашем случае в футбол играют Вася и Петя. Детерминированные предикаты порождают только одно решение.

Устранить такую ошибку можно двумя способам:

1) написать в разделе предикатов зарезервированное слово **nondeterm** перед описанием предиката например «**nondeterm** играет (symbol,symbol)», это указывает Visual Prolog, что предикат играет недетерминированный;

2) в настройках компилятора (рис. 4) установить переключатель **Nondeterm**. Это нужно для того, чтобы компилятор Visual Prolog принимал по умолчанию, что все определенные пользователем предикаты – недетерминированные при этом зарезервированное слово **nondeterm** можно не писать.

4. Составить программу «Телефонный справочник», в которой будут содержаться данные о владельце (фамилия или имя) и номер телефона (если он есть).

Составить простые запросы, с помощью которых выяснить:

– номер телефона по фамилии (имени);
– владельца имеющего определенный номер телефона.

Составить составные запросы, с помощью которых выясним:

– фамилии тех, кто имеет телефон и их номер;
– фамилии тех, у кого номера одинаковые номера телефонов;
– фамилии тех, у кого нет телефона;
– фамилии тех, кто имеет два номера телефона.

В программе указать разделы: predicates, clauses, goal. Цели (запросы) задавать поочередно в разделе goal.

5. Составить программу «Организация турнира по теннису», которая содержит сведения об именах и возрасте игроков. Каждая пара игроков одного возраста должна провести между собой две игры. Задача программы – составить список игр турнира, используя для этого составные запросы.

Контрольные вопросы

1. Из каких основных разделов состоит пролог-программа?
2. В чем разница между фактами и правилами?
3. Что такое составная цель и как она пишется?
4. Какое имя может иметь переменная в Пролог-программе?
5. Что такое «факт» в Пролог-программе?
6. Что такое «правило» в Пролог-программе?
7. Из каких частей состоит правило в Пролог-программе?
8. Сколько переменных можно использовать в разделе GOAL при тестировании программы утилитой Test-Goal?
9. Можно ли использовать русские буквы при написании имени переменной?

Практическое занятие № 3. Арифметические вычисления и сравнения в Прологе

Цель работы – изучить возможности встроенных предикатов и функций для выполнения вычислений, сравнений и операций ввода-вывода

Результатом практической работы является отчет. Отчет должен содержать листинг программы с комментариями, результаты, выдаваемые программой на каждый из запросов.

Для выполнения практической работы № 3 студент должен изучить приведенный ниже теоретический материал

Функции и предикаты

Возможности вычислений в Visual Prolog аналогичны соответствующим возможностям таких языков программирования, как C, Basic, Pascal.

Visual Prolog имеет полный набор встроенных математических функций и предикатов, которые используют целые и вещественные значения. Их список приведен в табл. 1.

Таблица 1. Арифметические функции и предикаты

X mod Y	Возвращает остаток от деления X на Y.
X div Y	Возвращает частное от деления X на Y.
abs(X)	Модуль X.
cos(X), sin(X), tan(X)	Косинус, синус, тангенс X соответственно (X в радианах).
arctan(X)	Арктангенс X.
exp(X)	Возводит ***e*** в степень X
ln(X), log(X)	Натуральный и десятичный логарифмы X.
sqrt(X)	Квадратный корень X.
random(X)	Присваивает X случайное число (0 <= X < 1).
random(X, Y)	Присваивает Y случайное целое число (0 <= Y < X).
randominit(SEED)	Инициализирует генератор случайных чисел. При постоянном SEED предикат random() будет выдавать одно и то же значение (используется для статистического тестирования).
round(X)	Округляет X.
trunc(X)	Усекает (отбрасывает дробную часть) X.
val(domain, x)	Явное преобразование числовых доменов (domain – имя возвращаемого домена).

Для возведения числа в произвольную степень X можно использовать комбинацию функций exp и ln. Например,

$12^{8,3}$=exp(8,3*ln(12)) или x^Y =exp(Y*ln(x))

Тригонометрические функции требуют, чтобы X был величиной, представляющей угол в радианах.

В арифметических выражениях в Visual Prolog используются: бинарные операторы (+, -, *, /), унарные операторы (-, +); скобки.

Приоритеты операций в порядке возрастания: бинарные операторы (-, +), бинарные операторы (/, *, mod, div), унарные операторы (-, +), скобки.

Сравнение

В Visual Prolog используются следующие операторы сравнения: <, <=, =, >=, >, <> или ><.

Сравнивать можно как числовые, так и символьные выражения.

При сравнении символов Пролог использует значение кодов ASCII для каждого символа. Например, сравнение ‘a’ < ‘b’ преобразуется в арифметическое выражение 97<98.

Когда сравниваются строки или идентификаторы, результат зависит от сравнения символов на соответствующих позициях. Результат равен тому, который получился бы при сравнении первых символов в том случае, если эти два символа не равны. Если же они равны, то Пролог сравнивает следующую пару символов и возвращает результат тогда, когда они не равны. В противном случае проверяется следующая пара и т. д. Если конец достигнут, и пара различных символов не найдена, более короткая строка считается меньшей.

Сравнение «antony» > «antonia» оценивается как истинное, т. к. первая пара различных символов y и i имеют значения 79 и 69 в кодах ASCII. Аналогично, сравнение «aa» > «a» -истинно.

Идентификаторы не могут сравниваться непосредственно из-за синтаксиса. Например, при сравнении P1=peter, P2=sasha идентификатор peter не может непосредственно сравниваться с идентификатором sasha. Они должны быть связаны с переменными, которые сравниваются или записаны как строки: P1>P2.

Генератор случайных чисел

Для генерации случайных чисел в Visual Prolog предусмотрены два стандартных предиката. Один возвращает случайное вещественное число в диапазоне от 0 до 1, другой возвращает случайное число в диапазоне от 0 до данного числа.

Предикат **random/1** возвращает случайное вещественное число RandomReal, удовлетворяющее условию: 0<= RandomReal<1. Формат предиката: random (RandomReal).

Предикат **random/2** имеет два аргумента, его формат**:** random(MaxValaue, RandomInt). Этот предикат ставит в соответствие RandomInt случайное число, удовлетворяющее условию: 0<=RandomInt<MaxValaue. Предикат random/2 работает быстрее, чем **random/1,** т. к. имеет целочисленную арифметику.

Предикат **randominit/1** имеет формат: randominit(Seed). По умолчанию случайное начальное значение генерируется как функция системного времени, а аргумент Seed устанавливает начальное значение. Основное назначение randominit – предоставить повторяемую последовательность псевдослучайных чисел для статистического тестирования.

Целочисленная и вещественная версии random используют одинаковое начальное значение и базовый генератор чисел.

Предикаты ввода-вывода

Вывод на экран. Для вывода на экран используется предикат **write()**. Его синтаксис:

1) **write(***список переменных для вывода [управляющие символы]***)**. Переменные разделяются запятыми;

2) **write(***«строка для вывода [управляющие символы]»***)**. Управляющие символы – \t, \r, \n и т. д.,

где \t – табуляция;

\r – возврат каретки;

\n – начало новой строки.

Для того чтобы начать вывод в новую строку используется предикат **nl** – *new line*.

Пример:

PREDICATES

```
s(string)
CLAUSES
s(«Значение»).
GOAL
s(W),
write(«\tПрограмма пример\nВторая строка\n»),
write(«Ответ =«,»\t»,W), nl,
write(«Конец программы»),nl,nl.
```

После запуска программа выдает следующий результат:

```
        Программа пример
Вторая строка
Ответ =     Значение
Конец программы

W=Значение
1 Solution
```

Последние две строки программа, а точнее утилита Test Goal, выдает автоматически.

Ввод с клавиатуры. Для ввода данных с клавиатуры используются предикаты:

readint(переменная**)** – ввод одной переменной типа INTEGER;

readreal(переменная**)** – ввод одной переменной типа REAL;

readchar(переменная**)** – ввод одной переменной типа CHAR.

Во всех случаях, кроме последнего, ввод необходимо заканчивать нажатием клавиши ENTER.

Порядок выполнения работы

1. Составить программу вычисления арифметического выражения. Для каждого задания написать 3 варианта программ:

– с заданием переменных в теле программы;

– с введением данных с клавиатуры;

– с заданием значений переменных с помощью генератора случайных чисел

Контрольные вопросы

1. Какие выражения можно сравнивать в Прологе?
2. Какие встроенные предикаты используются для ввода и вывода?
3. Каково основное назначение предиката randominit?
4. Как происходит сравнение строковых переменных?
5. Чем отличаются бинарные операторы от унарных?
6. В чем состоит отличие встроенных предикатов **random/1** и **random/2**?

Практическое занятие № 4. Управление поиском решений

Цель работы – изучить встроенный механизм поиска с возвратом и инструментальные средства управления поиском решений

Результатом практической работы является отчет. Отчет должен содержать листинг программы с комментариями, результаты, выдаваемые программой на каждый из запросов.

Для выполнения практической работы № 4 студент должен изучить приведенный ниже теоретический материал.

Использование предиката fail

Visual Prolog начинает поиск с возвратом, когда вызов завершается неудачно. В определенных ситуациях бывает необходимо инициализировать выполнение поиска с возвратом, чтобы найти другие решения. Visual Prolog поддерживает специальный предикат fail, вызывающий неуспешное завершение, и, следовательно, инициализирует возврат. Действие предиката fail равносильно эффекту от сравнения 2=3 или другой невозможной подцели.

fail не может быть согласован (он всегда неуспешен), поэтому Visual Prolog вынужден повторять поиск с возвратом. При поиске с возвратом он возвращается к последнему обращению, которое может произвести множественные решения. Такое обращение называют недетерминированным. Недетерминированное обращение является противоположностью детерминированному обращению, которое может производить только одно решение.

Прерывание поиска с возвратом: отсечение

Visual Prolog предусматривает возможность отсечения, которая используется для прерывания поиска с возвратом; отсечение обозначается восклицательным знаком (!). Действует отсечение просто: через него невозможно совершить откат (поиск с возвратом).

Существуют два основных случая применения отсечения:

– Если вы заранее знаете, что определенные посылки никогда не приведут к осмысленным решениям (поиск решений в этом случае будет лишней тратой времени), – примените отсечение и программа станет быстрее и экономичнее. Такой прием называют зеленым отсечением.

– Если отсечения требует сама логика программы для исключения из рассмотрения альтернативных подцелей. Это – красное отсечение.

Использование отсечений

Предотвращение поиска с возвратом к предыдущей подцели в правиле:

```
rl :- a,  b,
      !, c.
```

Такая запись является способом сообщить Visual Prolog о том, что вас удовлетворит первое решение, найденное им для подцелей **a** и **b**. Имея возможность найти множественные решения при обращении к **c** путем поиска с возвратом, Пролог при этом не может произвести откат (поиск с возвратом) через отсечение и найти альтернативное решение для обращений **a** и **b**. Он также не может возвратиться к другому предложению, определяющему предикат rl.

Предотвращение поиска с возвратом к следующему предложению

Отсечение может быть использовано, как способ сообщить Visual Prolog, что он выбрал верное предложение для определенного предиката. Например, рассмотрим следующий фрагмент:

```
r(1):-
      !,
      a, b, c.
r(2):-
      !,
      d.
 r(3):-
      !,
      c.
r(_):-  write(«This  is  a catchall clause.»).
```

Использование отсечения делает предикат r детерминированным. В данном случае, Visual Prolog выполняет обращение к r с единственным целым аргументом. Предположим, что произведено обращение r(l). Visual Prolog просматривает программу в поисках соответствия для обращения; он находит его с первым предложением, определяющим r. Поскольку имеется более чем одно возможное решение

для данного обращения, Visual Prolog проставляет точку возврата около этого предложения. Теперь Visual Prolog начинает обработку тела правила, проходит через отсечение и исключает возможность возвращения к другому предложению r. Это отменяет точки поиска с возвратом, повышая эффективность выполнения программы, а также гарантирует, что отлавливающее ошибки предложение будет выполнено лишь в том случае, если ни одно из условий не будет соответствовать обращению к r.

Обратите внимание, что конструкция такого типа весьма похожа на конструкцию case в других языках программирования; условие проверки записывается в заголовке правил.

Однако следует, по возможности, помещать проверочное условие именно в заголовок правила – это повышает эффективность программы и упрощает ее чтение.

Порядок выполнения работы

1. Какие ответы даст программа, содержащая следующие факты и правила:

p(1).

p(2) : – !.

p(3).

Цели: p(X); p(X), p(У); p(X), !, p(У).

Дать пояснения.

2. Составить программу решения двухступенчатой функции, используя бинарное отношение f(X,Y), без применения отсечений; применив зеленые отсечения в двух первых правилах; используя красные отсечения.

Правило 1: если X<3, то Y=0.

Правило 2: если X>3 и X<6, то Y=2.

Правило 3: если X>=6, то Y=4.

3. Составить программу определения наименьшего (наибольшего) из двух чисел (X,Y) без применения отсечений и с применением зеленых и красных отсечений.

4. Выполнить задания согласно вариантам, используя предикаты fail, cut (!).

Варианты заданий

1. Составить все возможные варианты выплаты заданной суммы денег денежными знаками номиналом 1, 2, 5, 10, 50 и 100 рублей.

2. Составить все возможные варианты кода для кодового замка с 4-мя кнопка (цифрами и/или буквами). Длина кода – 2 знака.

3. Составить все возможные варианты влюбленных пар в компании из 3 мальчиков и 5 девочек.

4. Написать программу, которая выводит на экран все двоичные числа от 0 до 15.

5. Составить расписание всех игр (дома и в гостях) для группы из 4-х футбольных команд.

6. Найти все трехзначные числа, сумма цифр которого равна 14.

7. Найти все двухзначные числа, произведение цифр которого равно 18.

8. Подобрать все возможные числа от 0 до 9 при которых выражение

(A+B)C=A(B-C) верно.

9. Найти все трехзначные числа, сумма цифр которого равна произведению этих же цифр.

10. Подобрать все возможные числа от 0 до 9 при которых выражение (A+B)C=A(B+C)

Контрольные вопросы

1. Зеленое и красное отсечения, их отличие.

2. Принцип работы предикатов fail и cut (!).

Практическое занятие № 5. Декларации и правила Пролога

Цель работы – построение программы с использованием стандартных доменов. Получение навыков формирования правил для программ на языке Пролог и составления запросов для выявления конечной цели.

Результатом практической работы является отчет. Отчет должен содержать листинг программы с комментариями, результаты, выдаваемые программой на каждый из запросов.

Для выполнения практической работы № 5 студент должен изучить приведенный ниже теоретический материал.

Основные стандартные домены Пролога

В Visual Prolog есть несколько встроенных стандартных доменов. Их можно использовать при декларации типов аргументов предикатов без описания в разделе domains. Основные стандартные домены приведены в табл. 1.

Таблица 1. Основные стандартные домены

Домен	Описание и реализация
short	Короткое, знаковое, количественное. Все платформы 16 бит (–32768–32767)
ushort	Короткое, беззнаковое, количественное. Все платформы 16 бит (0–65535)
long	Длинное, знаковое, количественное. Все платформы 32 бит (–2 147 483 648–2 147 483 647)
ulong	Длинное, беззнаковое, количественное. Все платформы 32 бит (0–4 294 967 295)
integer	Знаковое, количественное, имеет платформо-зависимый размер. Платформы 16 бит (-32768–32767) и 32 бит (–2 147 483 648–2 147 483 647)
unsigned	Беззнаковое, количественное, имеет платформо-зависимый размер. Платформы 16 бит (0–65 535) и 32 бит (0–4 294 967 295)
byte	Все платформы 8 бит (0–55)
word	Все платформы 16 бит (0–65535)
dword	Все платформы 32 бит (0–4 294 967 295)

Синтаксически значение, принадлежащее одному из целочисленных доменов, записывается как последовательность цифр, кото-

рой в случае знакового домена может предшествовать не отделенный от нее пробелом знак минус. Имеются также и восьмеричные и шестнадцатеричные синтаксисы для основных доменов.

Домены byte, word и dword наиболее удобны при работе с машинными числами. В основном используются типы integer и unsigned, а также short и long.

Другие часто используемые базовые домены показаны в табл. 2.

Таблица 2. Основные стандартные домены

Домен	Описание и реализация
char	Символ, реализуемый как беззнаковый byte. Синтаксически это символ, заключенный между двумя одиночными кавычками: 'a'
real	Число с плавающей запятой, реализуемой как 8 байт в соответствии с соглашениями IEEE; эквивалентен типу double в C. Синтаксически числа с необязательным знаком (+ или –), за которым следует несколько цифр DDDDDDD, затем необязательная десятичная точка (.) и еще цифры DDDDDDD, за которыми идет необязательная экспоненциальная часть (e(+ или –)DDD): <+|–> DDDDD <.> DDDDDDD <e <+|–> DDD> Примеры действительных чисел (real): 42705…999…86.72 9111.929437521e238…79.83e+21(означает 79.83×10^{21}, как в других языках). При необходимости, целые числа автоматически преобразуются в real.
string	Последовательность символов, реализуемых как указатель на байтовый массив, завершаемый нулем, как в C. Для строк допускается два формата: 1. последовательность букв, цифр и символов подчеркивания, причем первый символ должен быть строчной буквой; 2. последовательность символов, заключенных в двойные кавычки. Примеры строк: телефон_номер, «билет на поезд», «Dorid Inc» Строки могут достигать длины в 255 символов, в то время как строки, которые Visual Prolog считывает из файла или строит внутри себя, могут достигать (теоретически) до 4 Гбайт на 32-битовых платформах.
symbol	Последовательность символов, реализуемых как указатель на вход в таблице идентификаторов, хранящей строки идентификаторов. Синтаксис – как для строк.

Идентификаторы и строки взаимозаменяемы, однако Пролог хранит их раздельно. Идентификаторы хранятся в таблице идентификаторов, а для представления используются лишь их индексы в этой таблице, но не сами строки идентификаторов. Это означает, что сопоставление идентификаторов выполняется очень быстро, а в случае если они встречаются в программе несколько раз, то и хранение компактно. Строки же не хранятся в поисковой таблице, и при необходимости сопоставления Пролог проверяет их символ за символом. Поэтому нужно самому определять, какой домен лучше использовать в каждой конкретной ситуации.

В табл. 3 приводятся несколько примеров простых объектов, принадлежащих к основным стандартным доменам.

Таблица 3. Простые объекты

Простой объект	Домен
«&&», «animal», d_t_l, персона	symbol или string
–1, 3, 0	integer
4.45, 0.01, –30.5	real
'a', 'b', 'c', '/', '&'	char

Задание типов аргументов при декларации предикатов

Объявление доменов аргументов в разделе predicates называется *заданием типов аргументов*. Если в программе используются только стандартные домены, то нет необходимости использовать раздел domains.

Предположим, что вы хотите описать предикат, который сообщал бы позицию буквы в алфавите, т. е. цель:

алфавит_позиция (Буква, Позиция).

должна вернуть Позиция =1, если Буква = a, Позиция =2, если Буква = b и т. д. Предложения этого предиката могут выглядеть следующим образом:

алфавит_позиция(Символ, N).

Если при объявлении предиката используются только стандартные домены, то программе не нужен раздел domains. Предположим, что вы хотите описать предикат так, что цель будет истина, если *Символ* является *N*-м символом алфавита. Предложения этого предиката будут такими:

алфавит_позиция('a', 1).
алфавит_позиция('b', 2).
алфавит_позиция('c', 3).
...
алфавит_позиция('z', 26).

Данный предикат можно объявить следующим образом:

predicates
алфавит_позиция(char, unsigned)

и тогда не нужен раздел *domains*. Если разместить все фрагменты программы вместе, получим:

predicates
алфавит_позиция(char, integer)
clauses
алфавит_позиция('a', 1).
алфавит_позиция('b', 2).
алфавит_позиция('c', 3).
% здесь находятся остальные буквы
алфавит_позиция('z', 26).

Ниже представлено несколько простых целей, которые можно использовать:

алфавит_позиция('a', 1).
алфавит_позиция(X, 3).
алфавит_позиция('z', What).

При декларации предикатов важно знать, что предикаты могут иметь различную арность (размерность). Арность предиката – это количество аргументов, которые он принимает. Вы можете иметь два предиката с одним и тем же именем, но отличающиеся арностью. В разделах predicates и clauses версии предикатов с одним именем и разной арностью должны собираться вместе; за исключением этого ограничения, различная арность всегда понимается как полное различие предикатов. Так, например, в разделе предикатов опишем два предиката с одним именем, но разной арностью:

predicaters
студент (string, integer)
студент (string, symbol, integer)

Первый предикат будет содержать информацию о фамилии и номере зачетной книжки студента, второй – о фамилии, группе и курсе.

Автоматическое преобразование типов

Совсем необязательно, чтобы при сопоставлении двух Пролог-переменных они принадлежали одному и тому же домену. Переменные могут быть связаны с константами из различных доменов. Такое смешение допускается, т. к. Пролог автоматически выполняет преобразование типов (из одного домена в другой), но только в следующих случаях:

– между строками (string) и идентификаторами (symbol);

– между целыми, действительными и символами (char). При преобразовании символа в числовое значение этим значением является величина символа в коде ASCII.

Если основной домен – *string*, то с ним совместимы аргументы из домена *symbol*. Если основной домен *integer*, то с ним совместимы домены *real, char, word* и др.

Такое преобразование типов означает, что можно:

– вызвать предикат с аргументами типа *string,* задавая ему аргументы типа *symbol* и наоборот;

– передавать предикату с аргументами типа *real* параметры типа *integer*;

– передавать предикату с аргументами типа *char* параметры типа *integer*;

– Использовать в выражениях и сравнениях символы без необходимости получения их кодов в ASCII.

Синтаксис правил

Правила используются в Прологе в случае, когда какой-либо факт зависит от истинности другого факта или группы фактов. Правила – это связанные отношения; они позволяют Прологу логически выводить одну порцию информации из другой. Правило принимает значение «истина», если доказано, что заданный набор условий является истинным.

В Прологе все правила имеют 2 части: заголовок и тело, разделенные специальным знаком **:-**.

Заголовок – это факт, который был бы истинным, если бы были истинными несколько условий. Это называется выводом или зависимым отношением.

Тело – это ряд условий, которые должны быть истинными, чтобы Пролог мог доказать, что заголовок правила истинен.

В качестве примера напишем следующее правило: «Билл любит цветы, если цветы красные». На языке Пролог это правило запишется так:

любит(билл, цветы):- красные(цветы).

В записанном правиле выражение «любит (билл, цветы)» является заголовком правила, а «красные (цветы)» – телом правила. Символ **:-** имеет смысл «если», и служит для разделения двух частей правила: заголовка и тела.

Тело правила может состоять из одной или более подцелей. Подцели разделяются запятыми, определяя конъюнкцию, а за последней подцелью правила следует точка:

Заголовок **:-** <Подцель>,<Подцель>,…,<Подцель>.

Каждая подцель выполняет вызов другого предиката Пролога, который может быть истинным или ложным. При этом Пролог проверяет истинность вызванного предиката, и если это так, то работа продолжается, но уже со следующей подцелью. Если хоть одна из подцелей ложна, то все правило ложно.

Разделительный знак заголовка и тела правила **:-** читается как «если» (if). Однако if Пролога отличается от if, написанного в других языках, например в Pascal, где условие, содержащееся в операторе if, должно быть указано перед телом оператора, который может быть выполнен. Другими словами: *если ЗАГОЛОВОК истинен, тогда ТЕЛО истинно.* Данный тип оператора известен как *условный оператор если/тогда (if/then).*

Пролог же использует другую форму логики в таких правилах. Вывод об истинности заголовка правила Пролога делается, если (после того, как) тело правила истинно: *ЗАГОЛОВОК истинен, если ТЕЛО – истинно (или: если ТЕЛО может быть выполнено).*

Таким образом, правило Пролога соответствует условной форме *тогда/если (then/if).*

Вы можете рассматривать правило и как процедуру. С такой «процедурной» точки зрения правила могут «попросить» Пролог выполнить другие действия, отличные от доказательства фактов, такие как напечатать что-нибудь или создать файл.

Как вы уже, наверное, заметили, факты и правила – практически одно и то же, кроме того, что факты не имеют явного тела. Факты ведут себя так, как если бы они имели тело, которое всегда истинно.

Контрольные вопросы

1. Факты и правила в Прологе.
2. Типы данных в Прологе.

Практическое занятие № 6. Декларации и правила Пролога

Цель работы – написать правила на Прологе, соответствующие предложенным отношениям.

Результатом практической работы является отчет. Отчет должен содержать листинг программы с комментариями, результаты, выдаваемые программой на каждый из запросов.

Для выполнения практической работы № 6 студент должен изучить приведенный ниже теоретический материал.

Рекурсивные правила в Прологе

В процедурных языках программирования (таких как C, Delphi, Pascal и т. д.) циклические вычисления организуются в циклах. В Прологе циклические вычисления организуются при помощи рекурсии. Рекурсия обычно применяется в ситуациях, когда число возможных решений заранее неизвестно, либо, когда обрабатываются структуры данных (списки и т. п.) с произвольным количеством элементов.

Рассмотрим использование рекурсивного правила на примере отношения родитель и отношения предок:

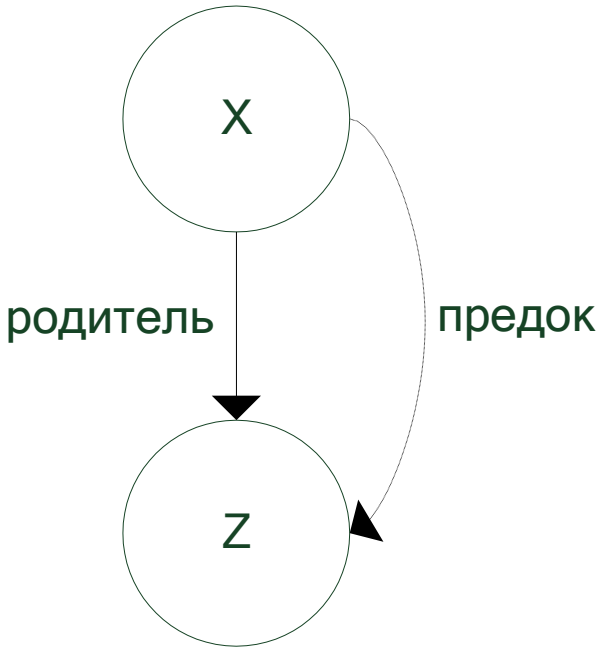

Рис. 1. X является прямым предком Z

Из рис. 1 можно выделить следующие факты и правила:

1) родитель(X, Z). Пример: Петя является родителем Васи.

2) предок(X, Z) :- родитель(X, Z). Пример: Петя является прямым предком Васи, если он является его родителем.

Данные факты и правила являются простыми, и у нас не возникает потребности в использовании рекурсии. Если посмотреть на рис. 2, то из него можно выделить более сложные факты и правила.

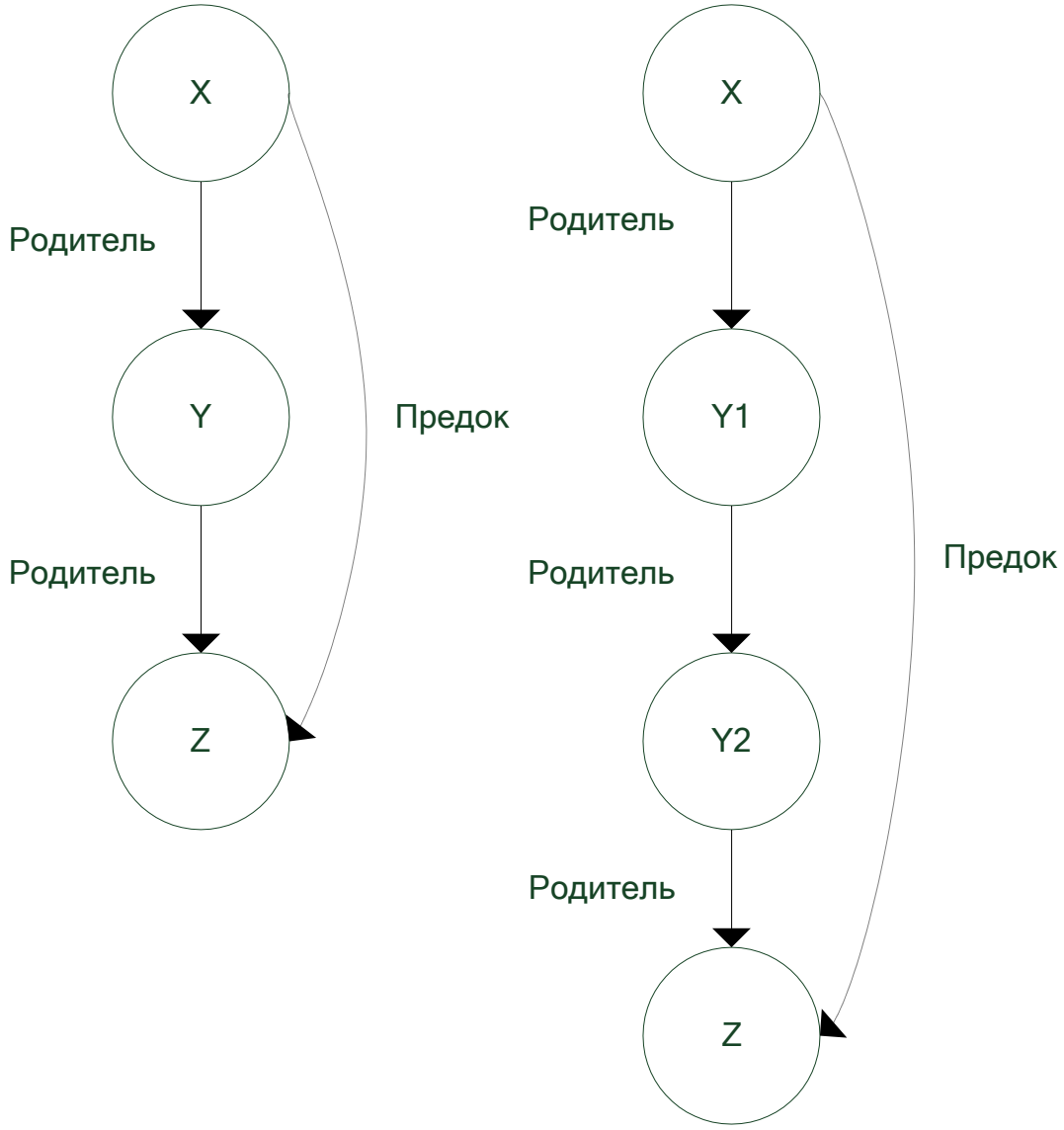

Рис. 2. X является не прямым предком Z

В частности можно выделить правило:

предок(X, Z) :- родитель(X, Y), родитель(Y, Z). Пример: Петя является не прямым предком Васи, если Петя является родителем Сережи, и Сережа в свою очередь является родителем Васи.

Также можно выделить еще правило:

предок(X, Z) :- родитель(X, Y1), родитель(Y1, Y2), родитель(Y2, Z). Пример: Петя является не прямым предком Васи, если Петя явля-

ется родителем Сережи, и Сережа в свою очередь является родителем Гены, и Гена в свою очередь является родителем Васи.

Минус данных правил заключается в том, что при увеличении числа родственных связей данные правила могут вырасти до неимоверных размеров. Что бы такого ни случилось, нужно воспользоваться рекурсией. Рассмотрим рис. 3.

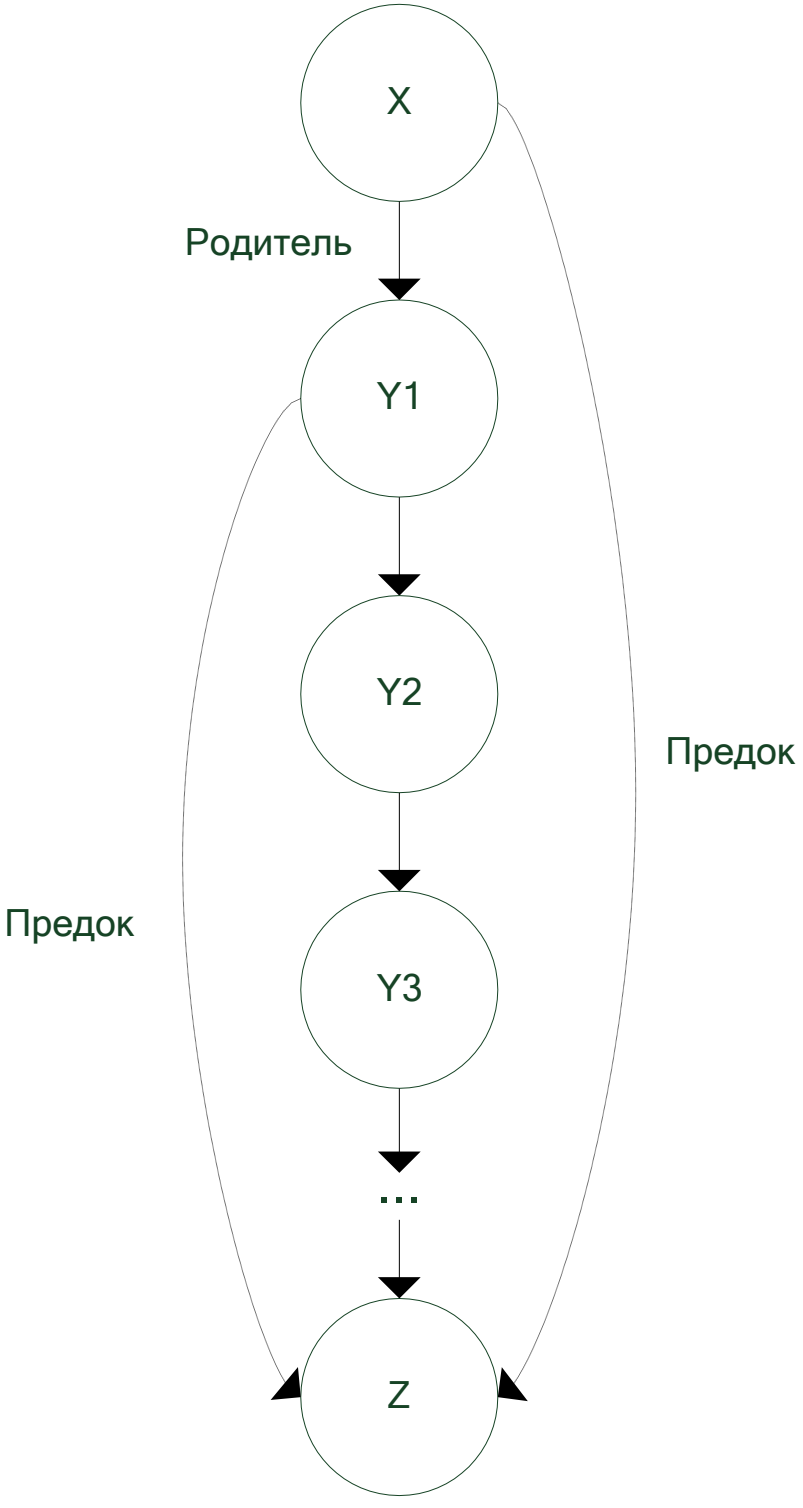

Рис. 3. Рекурсивное правило Предок

Для того что бы доказать является ли X предком Z, необходимо доказать что X является родителем Y1, и Y1 в свою очередь является предком Z. И т.д. до тех пор, пока не будет доказано, что предок Z является его родителем.

Правило «предок Z является его родителем» на языке Prolog выглядит следующим образом:

предок(X, Z) :- родитель(X, Z).

Данное правило нам необходимо для успешного завершения рекурсии. Рекурсивное правило выглядит следующим образом:

предок(X, Z) :- родитель(X, Y), предок(Y, Z).

Логика работы данного правила:

Введем в базу знаний системы Prolog следующие факты и правила:

1. родитель(«Дима», «Гриша»).
2. родитель(«Гриша», «Ваня»).
3. родитель(«Ваня», «Степа»).
4. родитель(«Степа», «Сергей»).
5. предок(X, Z) :- родитель(X, Z). (Правило 1).
6. предок(X, Z) :- родитель(X, Y), предок(Y, Z). (Правило 2).

Допустим, мы хотим определить, является ли Дима предком Сергея. Тогда наша цель будет выглядеть следующим образом:

предок(«Дима», «Сергей»).

Для того что бы доказать, что данная цель истинна, система Prolog начинает просмотр фактов и правил, в своей базе знаний, сверху вниз. Система Prolog находит правило предок(X, Z) :- родитель(X, Z), но оно не удовлетворяет условию, т. к. «Дима» не является предком «Сергея». Далее система Prolog находит правило предок(X, Z) :- родитель(X, Y), предок(Y, Z), управление переходит на первый предикат тела правила. И затем система Prolog ищет доказательство данного предиката. Система Prolog находит факт родитель(«Дима», «Гриша»). Затем управление переходит на второй предикат тела правила предок(X, Z) :- родитель(X, Y), предок(Y, Z) и пытается доказать истинность предиката предок(«Гриша», «Сергей»). Далее система Prolog вызывает рекурсию до тех пока не будет доказана истинность правила предок(X, Z) :- родитель(X, Z). Это будет успешным завершение рекурсии. Либо пока первый предикат правила предок(X, Z) :- родитель(X, Y), предок(Y, Z) будет завершен не успешно. Это будет не успешным завершением рекурсии.

Полная схема процесса достижения цели предок(«Дима», «Сергей») изображена на рис. 4.

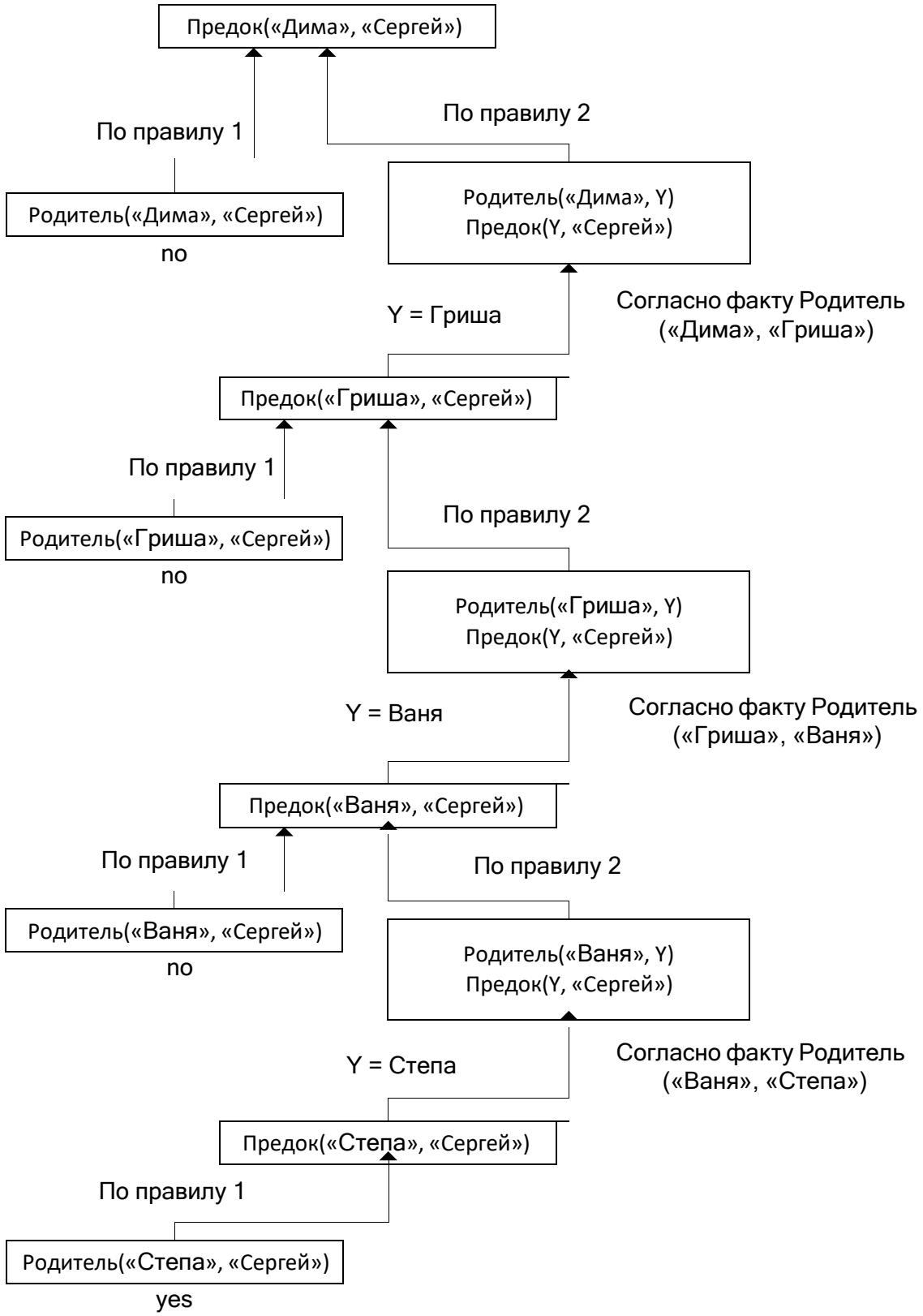

Рис. 4. Полная схема процесса достижения цели предок(«Дима», «Сергей»)

Исходный код данной задачи находится в Приложении 1.

Рекурсивное описание правила содержит в своем теле рекурсивный вызов (предикат вызывающий это же правило). В связи с этим возможны следующие разновидности рекурсии:

1. Правая рекурсия (хвостовая) – это когда рекурсивный вызов, находится в конце тела правила:

правило() :- правило11(), правило12(), … , **правило().**

2. Левая рекурсия (не хвостовая) – это когда рекурсивный вызов, находится в начале тела правила:

правило() :- **правило()**, правило21(), … , правило2N().

3. Обобщенная рекурсия (не хвостовая) – это когда рекурсивный вызов, занимает любое место в теле правила, исключая начальное и конечное положение:

правило() :- правило11(), правило12(), … , правило1N(), **правило()**, правило21(), правило22().

Для того чтобы во время выполнения рекурсивного правила не происходило зацикливания нужно условие завершения рекурсии. Их можно реализовать в Прологе двумя способами:

1. Заданием в программе альтернативного правила, не содержащего рекурсии. Выход произойдет при успешном выполнении альтернативного правила.

2. Формированием условия выхода одним из предикатов, который находится перед предикатом вызывающим рекурсию. Выход произойдет, если хотя бы один из предикатов завершится неуспехом.

Прежде чем начать создавать программы на языке Prolog необходимо разобраться, как рекурсия влияет на производительность программы и системы в целом. А именно нужно рассмотреть, как рекурсия использует память системы.

Всякий раз, когда в одном правиле вызывается другое правило, информация о вызывающем правиле должна быть сохранена в памяти компьютера, для того что бы оно (вызывающее правило) после выполнения вызванного правила, могло возобновить выполнение на том же месте где остановилось. Например: Если у нас есть рекурсивное правило, и оно вызывает само себя 100 раз, это означает что в памяти компьютера должно быть сохранено 100 различных состояний данного правила.

Если рекурсивное правило будет вызывать само себя, нужно попытаться сделать так, что бы избежать использования большого объема памяти. Есть случаи, когда правило может вызывать само себя

без сохранения информации, в памяти компьютера, о своем состоянии. Это такие правила, которые не собираются возобновлять своѐ выполнение после завершения вызванного правила. Например, рекурсивное правило которое было рассмотрено выше – предок(X, Z) :- родитель(X, Y), предок(Y, Z). Информация о таком правиле не будет сохраняться, потому что она больше не потребуется, т. к. рекурсивный вызов является последним предикатом данного правила. Данная операция (составление рекурсивных правил, которые не сохраняют информацию о своѐм предыдущем состоянии) называется оптимизацией хвостовой рекурсии.

Не хвостовая рекурсия будет всегда сохранять информацию о каждом состоянии правила, т. к. рекурсивный вызов находится не в конце тела правила, и после завершения вызванного правила, вызывающее правило должно вернуться в точку, на которой было прервано.

Для того что бы научиться правильно оптимизировать хвостовую рекурсию нужно рассмотреть ряд примеров:

1. Если рекурсивный вызов располагается не в конце тела правила – тогда рекурсия не будет являться хвостовой. Это приведет к сохранению в памяти компьютера информации о состоянии правила. Пример:

увеличение(X) :- NewX = X + 1, увеличение(NewX), write(X), nl.

При попытке запустить данное правило система Prolog выдаст ошибку: «Стек переполнен».

Для того что бы оптимизировать рекурсию необходимо привести правило к виду: увеличение(X) :- NewX = X + 1, write(X), nl, увеличение(NewX). То есть сделать рекурсию хвостовой.

2. Если к моменту выполнения рекурсивного вызова остается, какая либо непроверенная альтернатива данного правила, тогда рекурсия так же будет являться не оптимизированной. Пример:

Если поменять местами правила в примере, который был рассмотрен в начале данного методического пособия, тогда, когда происходит выполнение рекурсивного вызова, система Prolog еще не проверила второе правило. В случае неудачного завершения рекурсивного вызова, вызывающая процедура может откатиться и начать проверять альтернативы. За счет этого также расходуется память компьютера.

предок(X, Z) :- родитель(X, Y), предок(Y, Z).

предок(X, Z) :- родитель(X, Z).

3. Не оптимизированной также будет являться рекурсия, даже если непроверенная альтернатива выступает не как отдельное предложение данного правила. Непроверенная альтернатива может быть и в любом вызываемом предикате. Пример:

увеличение(X) :- NewX = X + 1, число(NewX), NewX < 1000, увеличение(NewX).

число(X) :- X >= 0, write(X, « число положительное»), nl.

число(X) :- X < 0, write(X, « число отрицательное»), nl.

Если задать цель: увеличение(-10), то при выполнении предиката число(NewX) первые несколько раз, до тех пор пока передаваемый атрибут меньше 0, информация о состоянии правила увеличение(X) не сохраняется, т. к. оба правила число(X) проверяются. Как только передаваемый атрибут станет больше либо равен 0, информация о состоянии правила увеличение(X) начнет сохраняться, т. к. вторая альтернатива правила число(X) не проверяется. И система Prolog будет выставлять точки отката.

Не смотря на то, что к моменту выполнения рекурсивного вызова остается, какая либо непроверенная альтернатива (данного правила, либо вызываемого предиката), можно сделать так, что бы правило не сохраняло информацию о своем состоянии, и система Prolog не выставляла точки отката. Отсечение (!) позволяет игнорировать все возможные альтернативы.

Оптимизация хвостовой рекурсии на примере 2: Для того что бы всякий раз система Prolog не выставляла точки отката и рекурсивное правило предок(X, Z) :- родитель(X, Y), предок(Y, Z). не сохраняло информацию о своем состоянии, необходимо поставить проверку и отсечение после предиката родитель(X, Y). Из этого выйдет: предок(X, Z) :- родитель(X, Y), Z <> Y, !, предок(Y, Z). Таким образом получается: если система Prolog находит решение для предиката родитель(X, Y) и значение переменной Z не равно значению переменной Y, тогда мы утверждаем, что нам не нужно доказывать альтернативу предок(X, Z) :- родитель(X, Z). Тогда соответственно не будет выставляться точка отката, и рекурсивное правило не будет сохранять информацию о своем состоянии.

Исходный код данной задачи находится в Приложении 2.

Оптимизация хвостовой рекурсии на примере 3: Для того что бы всякий раз система Prolog не выставляла точки отката и рекурсивное правило увеличение(X) :- NewX = X + 1, число(NewX), NewX < 1000, увеличение(NewX). не сохраняло информацию о своем состоянии,

необходимо поставить отсечение после X >= 0 в правиле число(X). Из этого выйдет: число(X) :- X >= 0, !, write(X, « число положительное «), nl. Таким образом получается: если система Prolog находит решение для предиката число(NewX) и ему соответствует правило число(X) :- X >= 0, !, write(X, « число положительное «), nl. тогда мы утверждаем, что нам не нужно доказывать альтернативу число(X) :- X < 0, write(X, « число отрицательное»), nl. Тогда соответственно не будет выставляться точка отката, и рекурсивное правило не будет сохранять информацию о своем состоянии.

Исходный код данной задачи находится в Приложении 3.

Приложение 1

```
PREDICATES
        родитель(string,string).
nondeterm предок(string,string).
CLAUSES
    родитель(«Дима», «Гриша»).
    родитель(«Гриша», «Ваня»).
    родитель(«Ваня», «Степа»).
    родитель(«Степа», «Сергей»).
    предок(X, Z) :- родитель(X, Z).
    предок(X, Z) :- родитель(X, Y), предок(Y, Z).
GOAL
    предок(«Дима», «Сергей»).
```

Приложение 2

```
PREDICATES
    родитель(string,string).
    предок(string,string).
CLAUSES
    родитель(«Дима», «Гриша»).
    родитель(«Гриша», «Ваня»).
    родитель(«Ваня», «Степа»).
    родитель(«Степа», «Сергей»).
    предок(X, Z) :- родитель(X, Y), Z <> Y, !, предок(Y, Z).
    предок(X, Z) :- родитель(X, Z).
GOAL
    предок(«Дима», «Сергей»).
```

Приложение 3

```
PREDICATES
        увеличение(integer).
        число(integer).
CLAUSES
        увеличение(X) :- NewX = X + 1, число(NewX), NewX < 1000, увеличение(NewX).
        число(X) :- X >= 0, !, write(X, « число положительное»), nl.
        число(X) :- X < 0, write(X, « число отрицательное»), nl.
GOAL
        увеличение(-10).
```

Контрольные вопросы

1. Что такое рекурсия?
2. Какие существуют типы рекурсивных правил?
3. Что такое оптимизация хвостовой рекурсии?
4. Какие существуют типы условий завершения рекурсии?

Содержание самостоятельной работы

Цель самостоятельной работы обучающихся – получить новые знания по дисциплине «Интеллектуальные системы и технологии».

Самостоятельная работа необходима для формирования у обучающихся способности самостоятельно решать задачи профессиональной деятельности, формирования умения и навыков планирования времени, формирования стремления развиваться и совершенствоваться.

Виды самостоятельной работы обучающихся указаны в табл. 1.

Таблица 1. Виды самостоятельной работы

№ п/п	Вид СРС
1	Изучение, поиск и установка свободно распространяемых интеллектуальных систем
2	Оформление отчетов по практическим работам

Обучающиеся должны изучить интернет-ресурсы и литературу по вопросам, представленным ниже.

Учебно-методические материалы по дисциплине

Основная литература

1. Боровская, Е. В. Основы искусственного интеллекта [Электронный ресурс]. – Москва: Лаборатория знаний, 2016. – 130 с. – Режим доступа: http://biblioclub.ru/index.php?page=book_red&id=440877. – Загл. с экрана.

Дополнительная литература

1. Кухаренко, Б. Г. Интеллектуальные системы и технологии [Электронный ресурс]. – Москва: Альтаир, МГАВТ, 2015. – 115 с. – Режим доступа: http://biblioclub.ru/index.php?page=book_red&id=429758. – Загл. с экрана.

Программное обеспечение и интернет-ресурсы

1. Официальный сайт Кузбасского государственного технического университета имени Т.Ф. Горбачева.

Режим доступа: www.kuzstu.ru

2. Электронные библиотечные системы:

– Университетская библиотека онлайн. Режим доступа: www.biblioclub.ru;

– Лань. Режим доступа: http://e.lanbook.com

– Электронно-библиотечная система Znanium.com

– Электронная библиотека издательства Юрайт https://biblio-online.ru/catalog/spo

3. Информатика и информационные технологии: конспект лекций. [Электронный ресурс]. – Режим доступа: http://fictionbook.ru

4. Современные тенденции развития компьютерных и информационных технологий: [Электронный ресурс]. – Режим доступа: http://www.do.sibsutis.ru

5. Единая коллекция Цифровых образовательных ресурсов [Электронный ресурс]. – Режим доступа:
http://school-collection.edu.ru/, свободный. – Загл. с экрана.

6. Единое окно доступа к информационным ресурсам [Электронный ресурс]. – Режим доступа:
http://window.edu.ru/, свободный. – Загл. с экрана.

7. Информационно-коммуникационные технологии в образовании [Электронный ресурс]. – Режим доступа: http://www.ict.edu.ru/, свободный. – Загл. с экрана.

8. Федеральный центр информационно-образовательных ресурсов [Электронный ресурс]. – Режим доступа: http://fcior.edu.ru/, свободный. – Загл. с экрана.

9. Visual Prolog 5.x и выше.