ФЕДЕРАЛЬНОЕ АГЕНТСТВО ПО ОБРАЗОВАНИЮ

Государственное образовательное учреждение
высшего профессионального образования

САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ

ЛОГИЧЕСКОЕ ПРОГРАММИРОВАНИЕ

Методические указания к выполнению лабораторных работ 1 — 4

Санкт-Петербург

2006

Составитель: А.В.Бржезовский

Рецензент:

В методические указания включены описания базовых конструкций языка PROLOG, структуры логической программы и механизма ее выполнения. В методических указаниях приведены примеры программ, решающих логические задачи, производящих рекурсивную обработку списков, работающих с динамической базой данных, осуществляющих поиск решений посредством перебора в пространстве возможных состояний (недетерминированное программирование).

Методические указания предназначены для студентов специальностей 230105 (220400)— «Программное обеспечение вычислительной техники и автоматизированных систем» и 010503 (351500) — «Математическое обеспечение и администрирование информационных систем», изучающих дисциплины «Логическое программирование», «Рекурсивно-логическое программирование».

Методические указания подготовлены кафедрой компьютерной математики и программирования и рекомендованы к изданию редакционно-издательским советом Санкт-Петербургского государственного университета аэрокосмического приборостроения.

Лабораторная работа 1. Решение логических задач

1. Константы, переменные, домены

В языке PROLOG, так же как в любом языке программирования используются понятия константы, переменной, типа данных. Вместе с тем, так как теоретической основой языка является логика предикатов первого порядка (PROLOG от PROgramming in LOGic), в языке используется терминология, заимствованная из данной теории. Первоначальные реализации языка и некоторые из существующих в настоящее время не требуют типизации аргументов предикатов, другие (например, такая система программирования как Visual Prolog — далее по тексту VIP) вводят типизацию для повышения эффективности выполнения логических программ. В таблице 1.1 приведены типы констант и перечислены соответствующие им домены (типы данных).

Таблица 1.1.

Константы и домены

Константа	**Форма записи**	**Домен**	**Примеры**
Символьная константа	Один символ, заключенный в знаки апострофа	char	‘A’, ‘a’, ‘1’
Целочисленная константа	Последовательность цифр, возможно предваряемая знаками + или –	integer	123, –5
Вещественная константа	Последовательность цифр, возможно предваряемая знаками + или –, целая часть от дробной отделяется точкой, а степень числа – символом e	real	–1.5, 0.123, 3.5e–7
Символическая константа	Последовательность букв или цифр, начинающаяся со строчной буквы и не содержащая специальных символов, за исключением знака подчеркивания	symbol	one, v_Prolog
Строковая константа	Произвольная последовательность символов заключенная в кавычки	string	“One”, “V. Prolog”

Домены, приведенные в табл. 1.1 являются стандартными (предопределенными), на их основе могут создаваться домены пользователя, определение доменов приводится в разделе логической программы, следующим за ключевым словом DOMAINS в соответствии со следующим синтаксисом:

```
domains
        <имя домена1>, …, <имя домена N> = <имя домена>
```

Здесь <имя домена> — это стандартный домен или домен, определенный ранее. В приведенном ниже примере определяются пользовательские домены на базе стандартного домена string:

```
domains
        name = string
        first_name, last_name = name
```

Разделителем в разделе объявления доменов является перевод строки, новый домен должен записываться с новой строки.

Переменные в языке PROLOG наряду с константами используются в качестве аргументов предикатов, имя переменной записывается по правилам записи символических констант, но начинается с заглавной буквы. В языке PROLOG нет явного объявления переменных, первая заглавная буква является признаком, позволяющим транслятору отличить переменную от символической константы.

2. Предикаты, факты, структура логической программы

Предикат — это логическая функция, возвращающая значение истина (И) или ложь (Л) в зависимости от значений своих аргументов. Все предикаты, используемые в логической программе (за исключением встроеных) должны быть объявлены в разделе, следующем за ключевым словом PREDICATES. Синтаксис объявления выглядит следующим образом:

```
predicates
        <имя предиката 1>(<имя домена 1,1>, < имя домена 1,2>, …)
        <имя предиката 2>(<имя домена 2,1>, < имя домена 2,2>, …)
        …
```

Здесь <имя домена I,J> указывает на тип данных, к которому относятся аргументы предиката. В приведенном ниже примере объявляются предикаты студент и оценка:

```
domains
        name, discipline = string
        group, mark = integer
predicates
        nondeterm student(name, group)
        nondeterm mark(name, discipline, mark)
```

Несмотря на то, что для домена и предиката (MARK) использовано одно и тоже имя ошибки не происходит, так как с точки зрения транслятора языка они являются различными понятиями. Так же как и в разделе DOMAINS, разделителем в разделе PREDICATES являет-

ся перевод строки. Директива NONDETERM является особенностью варианта языка, реализованного в VIP, она указывает транслятору, что данный предикат может иметь более одного решения.

Для подтверждения истинности предикатов в логической программе используются факты и правила, которые обозначаются обобщенным понятием «клозы». Факты записываются в разделе, следующем за ключевым словом CLAUSES в соответствии с приведенным ниже синтаксисом:

```
clauses
        <имя предиката 1>(<аргумент 1,1>, < аргумент 1,2>, …).
        <имя предиката 1>(< аргумент 2,1>, < аргумент 2,2>, …).
        …
        <имя предиката I>(< аргумент I,1>, < аргумент I,2>, …).
        …
```

Как следует из примера, для одного предиката может быть задано произвольное множество фактов. В качестве аргументов в фактах могут использоваться константы и переменные, область действия имени переменной ограничивается фактом, в котором оно использовано. Разделителем в разделе CLAUSES служит точка, являющаяся признаком окончания текущего факта. Ниже приведены примеры фактов для рассмотренных выше предикатов STUDENT и MARK:

```
clauses
        student("Gruzdev", 4000).
        student("Lisichkin", 4000).
        student("Volnushkin", 4001).
        student("Syroezhkin", 4001).

        mark("Gruzdev", "F&LP", 4).
        mark("Gruzdev", "DB", 5).

        mark("Lisichkin", "F&LP", 3).
        mark("Lisichkin", "DB", 3).


        mark("Volnushkin", "F&LP", 5).
        mark("Volnushkin", "DB", 4).

        mark("Syroezhkin", "F&LP", 4).
        mark("Syroezhkin", "DB", 4).
```

Выполнение логической программы состоит в доказательстве истинности цели, заданной в разделе, следующим за ключевым словом GOAL. Простая цель (содержащая один предикат) записывается по тем же правилам, что и факт. Таким образом, логическая программа может состоять из четырех разделов:

```
domains
        <объявление домена 1>
        <объявление домена 2>
        …
predicates
        <объявление предиката 1>
        <объявление предиката 2>
        …
clauses
        <факт 1>
        <факт 2>
        …
goal
        <цель>
```

3. Унификация, доказательство простых целей

Унификация в языке PROLOG используется для принятия решения, может ли клоз быть использован для подтверждения истинности предиката, и является в общем случае процедурой сопоставления образцов. Вместе с тем, для таких объектов как константы и переменные правила унификации являются достаточно простыми:

(i) две константы унифицируются, если их значения совпадают;

(ii) две свободные переменные всегда унифицируются, переменные остаются свободными;

(iii) константа и свободная переменная всегда унифицируются, результатом унификации является связывание значения константы с переменной;

(iv) связанная и свободная переменная всегда унифицируются, результатом унификации является связывание значения первой переменной со второй;

(v) две связанные переменные унифицируются, если совпадают значения связанных с ними констант.

PROLOG производит доказательство простых целей, руководствуясь следующими правилами:

(i) выбирается первый факт в разделе CLAUSES, имя которого совпадает с именем предиката, указанного в цели;

(ii) осуществляется унификация аргументов предиката цели и выбранного факта;

(iii) если унификация успешна, предикат считается истинным, а цель доказанной;

(iv) если унификация не успешна, осуществляется ОТСТУП — выбирается следующий факт для данного предиката;

(v) если унификация оказалась не успешна для всего множества фактов, цель считается ложной.

Например, для цели (см. программу в п. 2)

goal
student(“Volnushkin”, 4001).

VIP вернет значение, приведенное на рис. 1.1.

Рис. 1.1 Результат доказательства цели «student(“Volnushkin”, 4001).» .

А для цели:

goal
student(“Smorchkov”, 4001).

значение «no». В первом случае системе удалось найти факт, для которого оказалась успешной унификация аргументов предиката цели и факта, во втором случае подходящего факта найти не удалось. Если в цели указываются свободные переменные, то решений может быть несколько. Результат доказательства цели

goal
student(X, 4001).

Приведен на рис. 1.2.

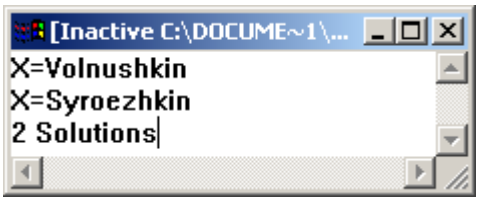

Рис. 1.2 Результат доказательства цели «student(X, 4001).).» .

С переменной X могут быть унифицированы как константа “Volnushkin”, так и константа “Syroezhkin”, поэтому два последних факта для предиката STUDENT подтверждают истинность заданной цели, при нахождении очередного истинного решения VIP выводит значения переменных, полученные ими в ходе унификации.

4. Доказательство составных целей, правила

Количество предикатов, указываемое в цели доказательства, может быть произвольным. В составной цели предикаты разделяются запятой или директивой AND, в конце цели ставится точка. PROLOG осуществляет доказательство составных целей, руководствуясь следующими правилами:

(i) выбирается i-й предикат цели;

(ii) производится поиск доказательства для i-го предиката по правилам, рассмотренным в п. 3;

(iii) если доказательство i-го предиката найдено, происходит переход к доказательству i+1-го;

(iv) если доказательство i-го предиката не успешно, осуществляется ВОЗВРАТ — поиск альтернативного решения для i–1-го предиката, при этом переменные, с которыми связались значения в ходе получения последнего доказательства i–1-го предиката, освобождаются (теряют свои значения);

(v) если доказательство найдено для всех предикатов цели, цель считается истинной, в противном случае цель считается ложной.

В качестве примера рассмотрим цель, которую можно сформулировать следующим образом «В каких группах есть студенты, получившие одинаковые оценки, как по дисциплине “Базы данных”, так и по дисциплине “Функциональное и логическое программирование”?»:

```
goal
        student(X, Y),
        mark(X,”DB”,Z),
        mark(X,”F&LP”,Z).
```

Результат доказательства приведен на рис. 1.3.ъ

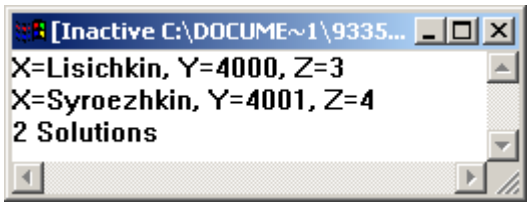

Рис. 1.3 Результат доказательства составной цели: «В каких группах есть студенты, получившие одинаковые оценки, как по дисциплине “Базы данных”, так и по дисциплине “Функциональное и логическое программирование”?».

Рассмотрим, как был получен данный результат. Первым подтверждением истинности для первого предиката цели будет «student("Gruzdev", 4000).», при этом с переменными свяжутся значения X ~ "Gruzdev", Y ~ 4000. Поэтому следующей подцелью будет доказательство «mark("Gruzdev",”DB”,Z)», ее подтверждением может служить второй факт для предиката MARK – «mark("Gruzdev", "DB", 5).», переменная Z получит значение 5. Дальнейшее доказательство подцели «mark("Gruzdev",”F&LP”,5)» потерпит неудачу, так как не найдется ни одного факта, для которого была бы успешна унификация аргументов, входящих в данную подцель. PROLOG осуществит возврат для поиска альтернативного решения, переменная Z при этом потеряет связанное с ней значение, альтернативных решений для подцели «mark("Gruzdev",”DB”,Z)» нет, поэтому произойдет возврат для поиска альтернативы для первого предиката цели, в ходе возврата будут освобождены переменные X и Y. Альтернативным решением для «student(X, Y)» будет «student("Lisichkin", 4000).», X ~ "Lisichkin", Y ~

4000. Доказательство второго предиката цели с данными значениями переменных — «mark("Lisichkin",”DB”,Z)» произойдет успешно, Z ~ 3. Так как доказательство третьего предиката цели «mark("Lisichkin",” F&LP ”,3)» существует — VIP выдаст 1-ю строку сообщения, приведенного на рис. 1.3. Процедура доказательства будет продолжаться, пока не будут найдены все другие возможные решения.

Другим примером составной цели может служить — «Кто из студентов получил по дисциплине “Функциональное и логическое программирование” оценку выше, чем Груздев?»:

```
goal
        mark("Gruzdev",”F&LP”,X),
        mark(Y,”F&LP”,Z),
        Y <> "Gruzdev", Z > X.
```

Результат доказательства цели приведен на рис. 1.4.

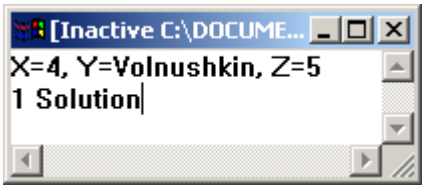

Рис. 1.4 Результат доказательства составной цели «Кто из студентов получил по дисциплине “Функциональное и логическое программирование” оценку выше, чем Груздев?».

Помимо фактов для доказательства истинности предикатов могут быть использованы правила. Синтаксис правила имеет вид:

```
clauses
        <имя предиката>(<аргумент 1>, < аргумент 2>, …) :–
                <имя предиката 1>(<аргумент 1,1>, < аргумент 1,2>, …),
                <имя предиката 2>(<аргумент 2,1>, < аргумент 2,2>, …),
                … .
```

Правило имеет заголовок и тело, разделяемые символами :– или директивой IF. Правило может быть использовано для определения истинности предиката, если успешна унификация аргументов доказываемой подцели и заголовка правила. Аргументами в заголовке и теле правила могут быть константы и переменные, при этом область действия имени переменной ограничивается данным правилом, т. е. одноименные переменных в различных правилах воспринимаются транслятором как различные переменные. Правило подтверждает истинность предиката, если успешно доказательство каждого из предикатов, входящих в тело правила. Доказательство тела правила осуществляется тем же способом, что и доказательство целей. Предикаты в теле правила разделяются запятой, которая читается как «и», вместо нее допускается использование директивы AND. Оформим цель «В каких группах есть сту-

денты, получившие одинаковые оценки, как по дисциплине “Базы данных”, так и по дисциплине “Функциональное и логическое программирование”?» в виде правила, добавим еще по одному разделу предикатов и клозов к рассматриваемой программе (в общем случае в программе может быть несколько разделов DOMAINS, PREDICATES, CLAUSES, общим требованием является следующее: определения должны быть приведены в тексте программы выше, чем конструкции, в которых они используются) и модифицируем цель:

```
predicates
        nondeterm mark_eq(group)
clauses
        mark_eq(Y):–
                student(X, Y),
                mark(X,”DB”,Z),
                mark(X,”F&LP”,Z).
goal
        mark_eq(X).
```

Результатом выполнения программы будет приведенный на рис. 1.5.

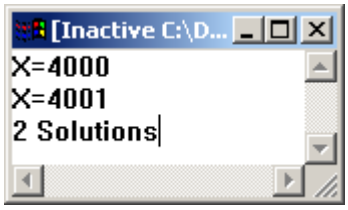

Рис. 1.5 Результат доказательства цели «mark_eq(X).».

Существуют цели, для доказательства которых необходимо убедиться не в истинности, а в ложности некоторого утверждения. Примером такой задачи является — «В каких группах есть студенты, имеющие только отличные оценки?». Для решения этой задачи можно составить правило, которое находит студентов, не являющихся отличниками, и использовать встроенный предикат NOT для нахождения требуемого решения. Предикат NOT имеет один аргумент, в качестве которого указывается другой предикат, NOT возвращает Л, если предикат – аргумент И, и наоборот. Добавим в рассматриваемую программу разделы предикатов и клозов, приведенные ниже, и модифицируем цель:

```
predicates
        nondeterm mark_less_5(name)
        nondeterm mark_5(group)
clauses
        mark_less_5(X):–
                mark(X, _, Z),
                Z < 5.

        mark_5(Y):–
                student(X, Y),
                not(mark_less_5(X)).
```

```
goal
        mark_5(X).
```

Предикат MARK_LESS_5 возвращает И, если у студента X нет оценок ниже 5, а предикат MARK _5 просматривает факты о студентах и вызывает MARK_LESS_5 для проверки. Заметим, что хотя имя студента как в правиле для предиката MARK_LESS_5, так и в правиле для предиката MARK_5 связывается с переменной X, с точки зрения транслятора это различные объекты программы. Результатом доказательства цели будет сообщение «No Solution», так как нет ни одной группы, для которой выполнялось бы сформулированное в задаче условие. В примере в правиле для MARK_LESS_5 используется знак подчеркивания, таким образом обозначаются свободные переменные, значения которых не существенны с точки зрения дальнейшего поиска доказательства.

5. Пример выполнения лабораторной работы

В лабораторной работе необходимо решить логическую задачу, составив для ее решения логическую программу на языке PROLOG. Примером логической задачи, соответствующей заданиям на выполнение лабораторной работы, может послужить следующая:

«Четыре приятеля А, Б, В и Г живут в четырех различных комнатах общежития. На вопрос “Кто, где живет?” трое из них дали по два ответа, из которых один истинен, а второй ложен:

А: “Я живу в первой комнате. Г живет во второй.”;

Б: “Я живу в третьей комнате. А живет во второй.”;

В: “Я живу во второй комнате. Б живет в четвертой.”.

Необходимо определить, в какой комнате живет каждый из приятелей».

Текст программы, находящей решение задачи, будет выглядеть следующим образом:

```
predicates
        nondeterm friend(string)
        nondeterm allocation(string,string,string,string)
        nondeterm condition1(string,string,string,string)
        nondeterm condition2(string,string,string,string)
        nondeterm condition3(string,string,string,string)
        nondeterm eq(string,string)

goal
        allocation(F1, F2, F3, F4),
        write("Комната 1 : ", F1), nl,
        write("Комната 2 : ", F2), nl,
        write("Комната 3 : ", F3), nl,
        write("Комната 4 : ", F4), nl.
```

```
clauses
        friend("А"). /* Четыре приятеля А,Б,В,Г живут в различных ком- */
        friend("Б"). /* натах общежития. На вопрос, где они живут,трое */
        friend("В"). /* дали по два ответа, из которых один истинный, */
        friend("Г"). /* другой ложный. */

 allocation(F1, F2, F3, F4):-
        friend(F1),
        friend(F2), not(eq(F1, F2)),
        friend(F3), not(eq(F1, F3)), not(eq(F2, F3)),
        friend(F4), not(eq(F1, F4)), not(eq(F2, F4)), not(eq(F3, F4)),
        condition1(F1, F2, F3, F4),
        condition2(F1, F2, F3, F4),
        condition3(F1, F2, F3, F4).

 /* А: "Я живу в первой комнате, ... */
 condition1("А", _, _, _).
 /* ... Г живет во второй". */
 condition1(_, "Г", _, _).


 /* Б: "Я живу в третьей комнате, ... */
 condition2(_, _, "Б", _).
 /* ... А – во второй". */
 condition3(_, "А", _, _).

 /* В: "Я живу во второй комнате, ... */
 condition3(_, "В", _, _).
 /* ... Б – в четвертой". */
 condition3(_, _, _, "Б").

 eq(X,X).
```

Результат работы программы приведен на рис. 1.6.

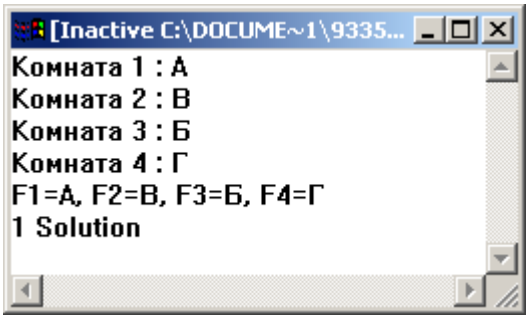

Рис. 1.6 Результат работы программы «Кто, где живет?».

В программе предикат FRIEND используется для того, что бы задать множество приятелей. Предикат ALLOCATION организует поиск решения, он имеет четыре аргумента, первый соответствует приятелю, проживающему в первой комнате, второй – во второй и т. д. При вызове предиката аргументы являются свободными переменными, вначале ALLOCATION выбирает в четыре различные переменные F1, F2, F3, F4 значения, соответст-

вующие четырем различным приятелем, после чего вызывает CONDITION1, CONDITION2, CONDITION3 для проверки ограничений. Для проверки неравенства значений F1, F2, F3, F4 используется EQ совместно с NOT. EQ, используя механизм унификации проверяет на совпадение значения первого и второго своего аргумента, а NOT инвертирует логическое значение, возвращаемое EQ. Первым набором данных связанным с переменными будет: F1 ~ "А", F2 ~ "Б", F3 ~ "В", F4 ~ "Г". Но такое распределение не удовлетворяет условиям задачи, так как для обоих клозов, соответствующих предикату CONDITION2 будет получена неудача, поэтому будет осуществлен возврат для нахождения альтернативных значений для предиката FRIEND. Следующим распределением, построенным ALLOCATION, будет F1 ~ "А", F2 ~ "Б", F3 ~ "Г", F4 ~ "В", но оно так же не удовлетворяет ограничениям. Перебор будет продолжаться, пока не будут рассмотрены все возможные варианты, единственное распределение приятелей по комнатам, соответствующее всем условиям задачи приведено на рис. 1.6. Предикаты WRITE и NL являются встроенными, они всегда возвращают логическое значение И, внелогическим результатом для WRITE является вывод его аргументов в диалоговое окно, а для NL – перевод строки. Пары символов /* и */ являются ограничителями комментариев, приводимых в программе на языке PPROLOG.

6. Варианты заданий

Составить программу для решения одной из логических задач № 1–4, 19, 20, 27, 31, 33, 40–43, 56, 58, 59, 61, 62, 64–67, 76–85, условия которых приведены в книге: Бизам Д., Герцер Я. Игра и логика. — М.: Мир, 1975 г.

Лабораторная работа 2. Обработка списков

1. Списки

Список – это последовательность элементов, заключенных в квадратные скобки и разделенных запятыми, примерами списков являются: [1, 2, 3]; [“one”, “two”]; [[‘a’, ‘b’], [‘c’, ‘d’, ‘e’], [‘f’, ‘g’]]. В первоначальной версии языка и некоторых современных системах программирования уровни вложенности и набор элементов списка ничем не ограничены, вместе с тем система программирования VIP накладывает следующее ограничение: элементы списка должны принадлежать к одному и тому же домену. Следствием этого является невозможность включать в один список числа, символы, строки и т.п., а так же смешивать в одном списке атомарные значения со списками или списки различного уровня вложенности. С точки зрения VIP следующие списки ошибочны: [1, 2, “two”]; [“one”, [‘a’, ‘b’]]; [[1, 2], [[3, 4], [5]]]. Для работы со списками должен быть объявлен списковый домен:

```
domains
        <имя спискового домена> = <имя домена элемента>*
```

Примерами списковых доменов могут служить:

```
domains
        il = integer*
        string_list = string*
        list_of_il = il*
```

Домену IL будет соответствовать список вида [1, 2, 3], домену STRING_LIST – [“one”, “two”], LIST_OF_IL – [[1, 2], [3, 4, 5], [6, 7]].

Для списков действуют следующие правила унификации – два списка унифицируются, если:

(i) списки имеют одинаковое количество элементов;

(ii) попарно унифицируются все элементы списков.

Так, например списки: [1, 2, 3] и [1, 2, 3]; [1, X, 3] и [1, 2, Y] унифицируются, в последнем случае с переменными в ходе унификации будут связаны значения: X ~ 2, Y ~ 3. Списки [[1, 2], [3]] и [X, Y] унифицируются: X ~ [1, 2], Y ~ [3]. Списки [1, 2, 3] и [1, 2]; [2, 1, 3] и [1, 2, 3]; [1, X, X] и [Y, Y, 2] не унифицируются, в последнем случае Y ~ 1, X ~ Y ~ 1, после чего унификация X и 2 становится невозможной.

Для списков в языке PROLOG предусмотрена специальная форма записи «голова-хвост». Голова списка от хвоста отделятся вертикальной чертой, например, запись [X, 2| Y] задает список, первым элементом которого является произвольный, вторым – константа 2, список может иметь 2 или более элементов. Списки, заданные в форме «голова-хвост» унифицируются, если унифицируются их головы по правилам, определенным для списков, с хвостом списка может быть связан подсписок любой длины. Так в результате унификации [X, 2| Y] и [1, 2] – X ~ 1, Y ~ [] ([] соответствует пустому списку); в результате унификации [X, 2| Y] и [1, 2, 3, 4, 5] – X ~ 1, Y ~ [3, 4, 5]; в результате унификации [X, 2| Y] и [1, 2| Z] – X ~ 1, Y ~ Z, т. е. переменные Y и Z остаются свободны. Унификация [X, 2| Y] и [2, 1| Z]; [X| Y] и []; [1, 2| X] и [1] невозможна, запись [X| Y] предполагает наличие в списке миниуму одного элемента, а [1, 2| X] – минимум двух.

2. Рекурсивные предикаты

Рекурсивными называются предикаты, для которых существуют правила, где в теле правила стоит обращение к этому же самому предикату. Примером рекурсивного предиката может служить предикат, осуществляющий вычисление факториала:

(i) факториал 0 равен 1 (0! = 1);

(ii) факториал N равен N, умноженному на факториал N–1 (N! = N × (N–1!)).

В программе на языке PROLOG эти утверждения будут выглядеть следующим образом:

```
predicates
        nondeterm factorial(integer, integer)
clauses
        factorial(0, 1).
        factorial(X, Y):– X > 0, X1 = X – 1, factorial(X1, Y1), Y = X * Y1.
goal
        factorial(5, X).
```

Результат выполнения программ приведен на рис. 2.1.

Рис. 2.1 Результат вычисления factorial(5, X).

Рекурсивные предикаты часто используются для обработки списков, одним из типичных представителей таких предикатов является MEMBER:

```
domains
        integerlist = integer*
        symbollist = symbol*
predicates
        nondeterm member(integer, integerlist)
        nondeterm member(symbol, symbollist)
clauses
        member(X, [X| _]).
        member(X, [_| Y]):– member(X, Y).
```

Прочесть клозы для предиката MEMBER можно следующим образом: «элемент входит в список, если это первый элемент списка», «элемент входит в список, если он входит в хвост списка». MEMBER, как и многие другие встроенные и определяемые пользователем предикаты, имеет несколько вариантов вызовов. Способы вызова предиката удобно показывать в форме шаблона вида (b, b, f, b, …), символ b говорит о том, что при обращении к предикату аргумент является связанным (константа или связанная переменная), а символ f означает, что аргумент свободен (свободная переменная, с которой свяжется значение в ходе доказательства истинности предиката). Предикат MEMBER имеет два варианта использования, в варианте (b, b) осуществляется проверка принадлежности элемента списку:

```
goal
        member(2, [1, 2, 3]).
```

даст результат YES, а

```
goal
        member(f, [a, b, c, d]).
```

даст результат NO. В варианте (f, b) вызов MEMBER поочередно возвращает все элементы заданного списка, для цели:

```
goal
        member(X, [1, 2, 3]).
```

результатом будет приведенный на рис. 2.2.

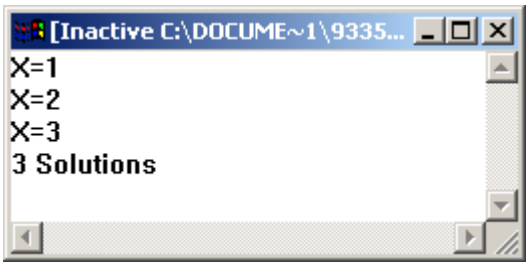

Рис. 2.2 Результат доказательства member(X, [1, 2, 3]).

Проиллюстрировать работу MEMBER можно, дополнив клозы операторами вывода:

```
clauses
        member(X, [X| Y]):– Z = [X| Y],
                write(X, " is first element of ", Z), nl.
        member(X, [_| Y]):– write("X = ", X, '\t', "Y = ", Y), nl,
                member(X, Y).
```

Тогда в ходе доказательства цели:

```
goal
        member(3, [1, 2, 3, 4, 5]).
```

Будет выдана последовательность сообщений, приведенная на рис. 2.3. Специальный символ '\t' задает табуляцию при выполнении вывода.

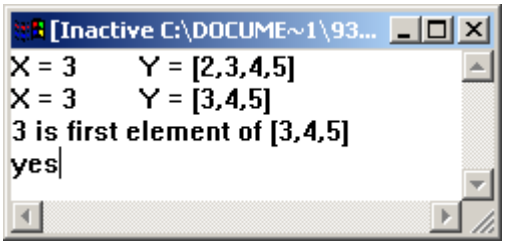

Рис. 2.3 Результат доказательства member(3, [1, 2, 3, 4, 5]).

Еще одним примером рекурсивного предиката для обработки списков может послужить APPEND, дополним приведенную выше программу разделами:

```
predicates
        nondeterm append(integerlist, integerlist, integerlist)
        nondeterm append(symbollist, symbollist, symbollist)
clauses
        append([], L, L).
        append([X|L1], L2, [X|L3]):– append(L1, L2, L3).
```

Клозы для предиката APPEND могут быть прочитаны следующим образом: «результатом слияния пустого списка с другим списком будет второй список», «для того, что бы соединить два списка, необходимо соединить хвост 1-го списка со 2-м списком и добавить первый элемент 1-го списка в голову полученного результата». Предикат APPEND допускает следующие варианты использования: (b, b, b) – проверка, 1-й и 2-й аргументы при слиянии должны давать список, указанный в качестве 3-го аргумента; (b, b, f) слияние списков, третий аргумент получает значение результата; (b, f, b) – отрезание подсписка слева; (f, b, b) – отрезание подсписка справа; (f, f, b) – генерация возможных разбиений списка на подсписки.

Результат доказательства цели:

```
goal
        append([1, 2, 3], [4, 5], X).
```

приведен на рис. 2.4,

Рис. 2.4 Результат доказательства append([1, 2, 3], [4, 5], X).

а результат доказательства цели:

```
goal
        append(X, Y, [1, 2, 3]).
```

на рис. 2.5

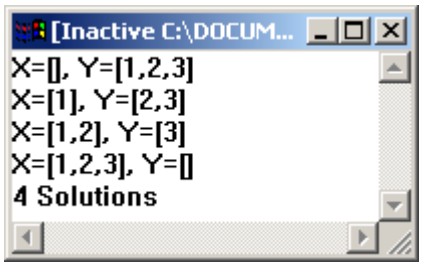

Рис. 2.5 Результат доказательства append(X, Y, [1, 2, 3]).

3. Пример выполнения лабораторной работы

Составим предикат, осуществляющий сортировку по возрастанию списка целых чисел. Текст программы будет выглядеть следующим образом:

```
domains
        integerlist = integer*
predicates
        nondeterm sort(integerlist, integerlist, integerlist)
        nondeterm incl(integer, integerlist,  integerlist)
clauses
        sort([], X, X).
        sort([X| Y], Z, W):– incl(X, Z, Z1), sort(Y, Z1, W).

        incl(X, [], [X]).
        incl(X, [Y| Z], [X, Y| Z]):– X <= Y.
        incl(X, [Y| Z], [Y| Z1]):– incl(X, Z, Z1).
goal
        sort([2, 1, 4, 3, 7, 6, 5], [], X), !.
```

Результат работы программы приведен на рис. 2.6

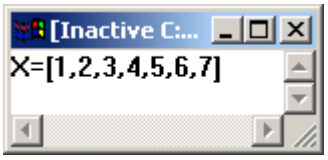

Рис. 2.6Результат работы программы

При вызове SORT в качестве первого аргумента указывается исходный список, второго аргумента – пустой список, третьего – свободная переменная, с которой будет связан результат. Первый клоз для SORT срабатывает, когда исходный список станет пустым, при этом промежуточный результат сортировки, накапливаемый во втором аргументе, за счет унификации связывается с третьим аргументом предиката. Второй клоз для SORT вызывает вспомогательный предикат INCL для того, что бы включить первый элемент списка в промежуточный результат, после чего происходит рекурсивный вызов SORT с измененными значениями аргументов: отброшен первый элемент списка [X| Y] и модифицирован список Z. Первый клоз для INCL обеспечивает включение элемента в пустой список; 2-й подставляет элемент на 1-ю позицию списка, в случае если значение включаемого элемента меньше или равно значению 1-го элемента списка; 3-й клоз обеспечивает включение элемента в хвост списка, если условие, проверяемое во 2-м клозе, оказалось ложным.

Решение задачи является единственным, для того что бы исключить попытки поиска транслятором альтернативных решений в цели использовании оператор отсечения, обозначаемый в языке PROLOG восклицательным знаком. В данном случае оператор запрещает

поиск альтернативных решений для SORT сразу, как только будет получено первое возможное решение.

7. Варианты заданий

1) Разработать предикат, объединяющий два списка в результирующий список, в котором чередуются элементы исходных списков, например: вход — [‘1’, ‘2’, ‘3’, ‘4’, ‘5’], [‘a’, ‘b’, ‘c’]; выход — [‘1’, ‘a’, ‘2’, ‘b’, ‘3’, ‘c’, ‘4’, ‘5’].
2) Разработать предикат, выделяющий из исходного списка подсписок, начиная с элемента с номером N и заканчивая элементом N + K. N и K — аргументы предиката, например: вход — [1, 2, 3, 4, 5, 6, 7, 8, 9], N = 3, K = 4; выход — [3, 4, 5, 6, 7].
3) Разработать предикат, находящий теоретико-множественное объединение двух списков, например: вход — [1, 2, 3, 4, 5], [4, 5, 6, 7]; выход — [1, 2, 3, 4, 5, 6, 7].
4) Разработать предикат, находящий теоретико-множественное пересечение двух списков, например: вход — [1, 2, 3, 4, 5], [4, 5, 6, 7]; выход — [4, 5].
5) Разработать предикат, находящий теоретико-множественную разность двух списков, например: вход — [1, 2, 3, 4, 5,], [4, 5, 6, 7]; выход — [1, 2, 3].
6) Разработать предикат, производящий удаление из исходного списка всех элементов с четными номерами, например: вход — [a, b, c, d, e]; выход — [a, c, e].
7) Разработать предикат, находящий сумму элементов с нечетными номерами в заданном списке чисел, например: вход — [2, 4, 3, 1, 7, 2, 4]; выход — 16.
8) Разработать предикат, аргументом которого является список, возвращающий список пар: [<элемент исходного списка> <количество его вхождений в исходный список>], например: вход — [1, 2, 1, 1, 3, 5, 2, 5], выход — [[1, 3], [2, 2], [3, 1], [5, 2]].
9) Разработать предикат, аргументом которого является список, возвращающий список, содержащий два подсписка. В первый подсписок включается N очередных элементов исходного, а следующие K элементов — во второй. Затем все повторяется. N и K — аргументы предиката. Например: вход — [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11] , N = 2, K = 3; выход — [1, 2, 6, 7, 11], [3, 4, 5, 8, 9, 10].
10) Разработать предикат, осуществляющую вставку в исходный список подсписка за элементом с номером N, например: вход — [‘1’, ‘2’, ‘3’, ‘4’, ‘5’, ‘6’], [‘a’, ‘b’, ‘c’,], N = 3; выход — [‘1’, ‘2’, ‘3’, ‘a’, ‘b’, ‘c’, ‘4’, ‘5’, ‘6’].
11) Разработать предикат, аргументом которого является список, а выходом список, элементами которого являются произведения 1-го и последнего элемента исходного списка, 2-го и предпоследнего и т. д., например: вход — [1, 2, 3, 4, 5, 6]; выход — [6, 10, 12,].

12) Разработать предикат, удаляющий из исходного списка элементы, порядковые номера которых заданы во втором списке, например: вход — [a, b, c, d, e, f, g], [2, 5]; выход — [a, c, d, f, g].
13) Разработать предикат, формирующий на основе исходного списка длины N список, содержащий суммы элементов с номерами 1 и N/2+1, 2 и N/2+2 и т. д. N — аргумент предиката, например: вход — [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]; выход — [7, 9, 11, 13, 15].
14) Разработать предикат, преобразующий арифметическое выражение, заданное в форме списка, в польскую обратную запись, например: вход — [‘3’, ‘*’, ‘2’, ‘-‘, ‘5’]; выход — [‘-‘, ‘*’, ‘3’, ‘2’, ‘5’].
15) Разработать предикат, выполняющий преобразование, обратное предыдущему.
16) Разработать предикат, перемещающий в исходном списке последовательность элементов, начиная с элемента с номером N и длины N + K, на позицию за элементом с номером M. N, K, M — аргументы предиката. Например: вход — [1, 2, 3, 4, 5, 6, 7, 8, 9], N = 3, K = 4, M = 8; выход — [1, 2, 7, 8, 3, 4, 5, 6, 9].
17) Разработать предикат, возвращающий количество четных [по значению] элементов в списке чисел, например: вход — [1, 2, 3, 4, 5, 6, 7, 8, 9]; выход — 4.
18) Разработать предикат, увеличивающий в исходном списке нечетные элементы [по значению] на 1 и уменьшающий четные на 2, например: вход — [1, 2, 3, 4, 5, 6, 7, 8, 9]; выход — [2, 0, 4, 2, 6, 4, 6, 6, 10].
19) Разработать предикат, инвертирующий в исходном списке последовательность элементов, начиная с элемента с номером N и заканчивая элементом с номером N + K. N и K — аргументы предиката. Например: вход — [1, 2, 3, 4, 5, 6, 7, 8, 9], N = 4, K = 3; выход — [1, 2, 3, 7, 6, 5, 4, 8, 9].
20) Разработать предикат, заменяющий в исходном списке символов и всех его подсписках последовательность … a b a … на последовательность … a b b a … . Например: вход — [a, b, c, d, a, b, a, c, e, f, d, a, b, a, b, a, b, b, a, c]; выход — [a, b, c, d, a, b, b, a, c, e, f, d, a, b, b, a, b, a, b, b, a, c].
21) Разработать предикат, возвращающий И, если в исходном списке из нулей и единиц, количество нулей больше, чем число единиц, и Л — в противном случае, например: вход — [1, 0, 0, 1, 0, 1, 1, 0, 0]; выход — yes.
22) Разработать предикат, вычисляющий среднее арифметическое для заданного списка чисел, например: вход — [1, 2, 3, 4, 5]; выход — 3.
23) Разработать предикат, транспонирующий матрицу, заданную списком списков, например: вход — [[1, 2, 3], [4, 5, 6], [7, 8, 9]]; выход — [[1, 4, 7], [2, 5, 8], [3, 6, 9]].

Лабораторная работа 3. Недетерминированное программирование

1. Понятие недетерминированного программирования

Существует класс задач, для которых невозможно предложить конкретный алгоритм их решения, но, вместе с тем, решение может быть получено последовательным перебором возможных вариантов с проверкой заданных ограничений. Метод решения таких задач иногда в литературе называется методом «построить и проверить». Подход к решению такого класса задач и получил обобщенное название недетерминированного программирования.

Примерами задач, решаемых перебором, являются игра в 8 (или 15), шашки, шахматы и др. Последние являются сложными и требуют не простого перебора, а перебора вариантов с оцениванием перспектив того или иного хода. При решении более простых задач можно использовать полный перебор вариантов. Указанные задачи традиционно относят к задачам искусственного интеллекта, такие задачи удобно реализуются на языке PROLOG, так как механизм бек-трекинга (поиска с возвратом), положенный в основу выполнения PROLOG-программы является по сути процедурой поиска решеняи в пространстве возможных состояний.

2. Пример выполнения лабораторной работы

Рассмотрим задачу, получившую в литературе название «Семь мостов через Преголю»— на реке Преголя в Калининграде есть два острова, которые соединены с берегами и между собой семью мостами, необходимо выйти из некоторой точки и вернуться в нее, пройдя по каждому из мостов в точности один раз. Задача не имеет решения, доказательство этого привел Леонард Эйлер в 1736 г. Вместе с тем можно построить программу, осуществляющую перебор вариантов (построение всех возможных путей], которая позволит убедиться, что не существует пути из точки в нее же, удовлетворяющего сформулированному выше критерию.

Фрагмент карты Калининграда приведен на рис. 3.1, а граф показывающий, какие берега и острова соединены мостами, показан на рис. 3.2.

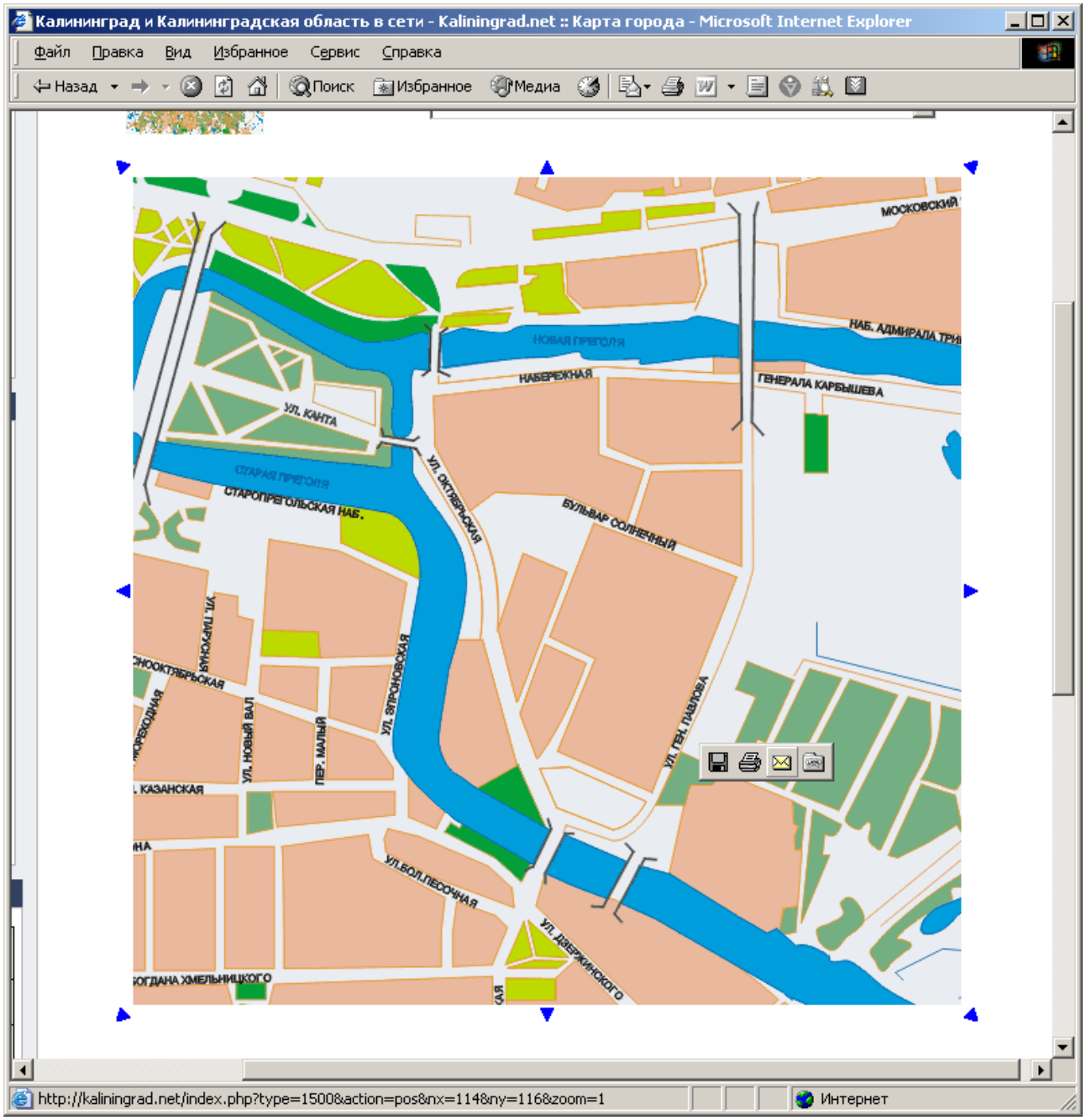

Рис. 3.1 Фрагмент карты Калининграда.

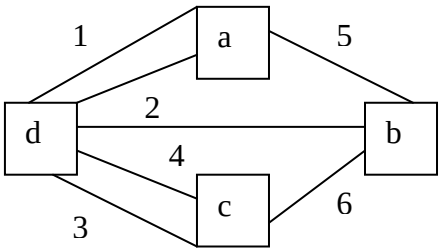

Рис. 3.2 Структура графа в задаче о мостах.

Для решения задачи может быть использован алгоритм поиска в глубину в пространстве возможных путей. Программа строит очередной возможный путь, после чего осуществля-

ется проверка на удовлетворение условиям задачи. Текст программы выглядит следующим образом:

```
domains
        integerlist=integer*

predicates
        nondeterm bridge(integer,symbol,symbol)
        nondeterm cango(integer,symbol,symbol)
        nondeterm startpoint(symbol)
        nondeterm go(symbol,symbol,integerlist)
        nondeterm member(integer,integerlist)

clauses
        bridge(1, a, d).        bridge(2, a, d).        bridge(3, c, d).
        bridge(4, c, d).        bridge(5, a, b).        bridge(6, b, c).
        bridge(7, b, d).

        startpoint(a).  startpoint(b).  startpoint(c).  startpoint(d).

        cango(X, Y, Z):– bridge(X, Y, Z).
        cango(X, Y, Z):– bridge(X, Z, Y).

        member(X, [X| _]).
        member(X, [_| Z]):– member(X, Z).


        go(X, X, [_, _, _, _, _, _, _]).
        go(X, Y, Z):– cango(Bridge, X, NewPoint),
                not(member(Bridge, Z)),
                go(NewPoint, Y, [Bridge| Z]).
goal
        startpoint(X),
        go(X, X, []).
```

С помощью фактов для предиката BRIDGE задается структура графа, 1-й аргумент BRIDGE содержит номер моста, 2-й и 3-й точки пространства, которые соединяет мост. Факты для предиката STARTPOINT определяют исходные точки для начала движения. Передвижение по мостам возможно в обоих направлениях, данную ситуацию обрабатывает CANGO, из точки Y в точку Z можно попасть, если существует мост (X, Y, Z) или мост (X, Z, Y). Предикат MEMBER совместно с NOT используется для проверки не вхождения элемента в список, он служит для предотвращения зацикливания и проверки условия задачи, мы не должны дважды проходить по одному и тому же мосту. Предикат GO осуществляет построение пути, первый клоз проверяет достижений целевого состояния когда, двигаясь из точки X мы оказались в ней же, при этом в списке пройденных мостов в точности семь элементов – [_, _, _, _, _, _, _]. Второй клоз строит продолжение пути, находя посредством обращения к CANGO точку, в которую можно переместиться из точки, в которой мы находим-

ся (она определяется 1-м аргументом CANGO). Если оказывается, что перемещение в новую точку не приводит к повторному прохождению моста (NOT(MEMBER(BRIDGE, Z))), данный мост подставляется в голову списка, связанного со 2-м аргументом CANGO, после чего осуществляется рекурсивный вызов GO. В разделе цели выбирается некоторая исходная точка, после чего осуществляется вызов CANGO для проверки существования пути.

Как было сказано выше, задача не имеет решения, поэтому результатом выполнения программы будет сообщение «No Solution». Для того, что бы убедиться в работоспособности программы, модифицируем предикат GO и цель доказательства:

```
predicates
        nondeterm go(symbol,symbol,integerlist,integerlist)

clauses
        go(X, X, Y, Y).
        go(X, Y, Z, Z1):– cango(Bridge, X, NewPoint),
                not(member(Bridge, Z)),
                go(NewPoint, Y, [Bridge| Z], Z1).
goal
        startpoint(X),
        go(X, X, [], Y).
```

Новый вариант GO будет возвращать путь, как только мы попадает в точку, из которой вышли. Программа выдаст сообщение, фрагмент которого приведен ниже:

```
X=a, Y=[]
X=a, Y=[2,1]
X=a, Y=[2,4,3,1]
…
X=d, Y=[4,3,2,5,7]
X=d, Y=[3,4,2,5,7]
164 Solutions
```

Таким образом, существует 164 варианта путей ведущих из некоторой точки в нее же, но среди них нет ни одного, в котором каждый мост проходился бы в точности один раз.

3. Варианты заданий

1) Дана схема метрополитена, найти кратчайший путь между станциями. Длина пути определяется количеством проезжаемых станций. Пересадка приравнивается к станции.
2) Игра в 8 или 15.
3) Задача о 8-ми ферзях (см. Вирт, 1989 г., с. 191].
4) Задача о Ханойской башне (см. Вирт, 1989 г., с. 210].
5) Раскрасить плоскую карту четырьмя цветами, так что бы любые две смежные области не были окрашены в один цвет.

6) Задача о коммивояжере. Коммивояжер должен посетить клиентов, находящихся в разных городах. Коммивояжер возвращается в тот же город, из которого он выехал. Коммивояжер никогда не бывает дважды в одном и том же городе. В каком порядке должен посещать города коммивояжер, что бы пройденный путь был минимальным.
7) Дана схема железных дорог, соединяющая N населенных пунктов, расписание движения поездов и стоимость проезда. Построить путь из пункта A в пункт B по критерию: а] минимального времени; б] минимального расстояния.
8) Дана карта автомобильных дорог, соединяющая N населенных пунктов, для каждой дороги задано расстояние и максимальная скорость. Построить путь из пункта A в пункт B по критерию: а] минимального времени; б] минимального расстояния.
9) Дана схема городского транспорта (маршруты автобусов, трамваев, троллейбусов]. Построить путь A в пункт B по критерию: а] минимального числа пересадок. б] минимального расстояния (количества остановок].
10) Дана сеть водопровода. Найти все участки, одновременное повреждение которых не ухудшает водоснабжение.
11) Дано M задач и N процессоров. Для задач задано отношение частичного следования и время выполнения каждой задачи. Построить оптимальную последовательность выполнения.
12) Дан нерегулируемый перекресток. Для введенного перечня подъезжающих транспортных средств (с указанием вида и направления движения], определить порядок их проезда.
13) Дан регулируемый перекресток. Для введенного перечня подъезжающих транспортных средств (с указанием вида и направления движения], определить порядок их проезда.
14) Задача об устойчивых браках. (см. Шапиро, стр. 173, п. 2].

Лабораторная работа 4. Динамическая база данных

1. Динамическая база данных

Полезной возможность языка PROLOG является добавление и удаление клозов для некоторого предиката непосредственно во время выполнения программы. Этот механизм и получил название динамической базы данных (ДБД). Существуют версии языка PROLOG в которых добавляться могут как факты, так и правила, система программирования VIP допускает добавление только фактов. Для того, что бы в программе для некоторого предиката была

возможность динамической работы с фактами, данный предикат должен быть описан в разделе программы, следующим за директивой DATABASE, синтаксис определения, такой же как и в разделе PREDICATES.

Встроенные предикаты ASSERTA(<факт>) и ASSERTZ(<факт>) добавляют новый факт в ДБД, отличие между ними состоит в том, что ASSERTA подставляет факт первым в группу фактов для данного предиката, а ASSERTZ – последним. Предикаты ASSERTA и ASSERTZ всегда возвращают логическое значение И.

Для удаления факта из ДБД используется встроенный предикат RETRACT(<факт>). Предикат удаляет первый факт в ДБД, для которого удалось произвести унификацию аргументов с аргументами факта, указанного в RETRACT. В данном случае предикат возвращает значение И, если подходящего факта в ДБД не нашлось, RETRACT возвращает Л.

Содержимое ДБД может быть сохранено в дисковом файле обращением к встроенному предикате SAVE(<имя файла ОС>), и загружено с помощью встроенного предиката CONSULT(<имя файла ОС>). Предикаты возвращают значение И, если имя файла соответствует правилам записи, принятым в операционной системе (ОС) и не произошло ошибок ввода/вывода.

Использование данных предикатов иллюстрирует следующий пример программы:

```
domains
        name, group = string
database
        student(name, group)
goal
        asserta(student("Gruzdev", "4000")),
        asserta(student("Lisichkin", "4000")),
        save("student.txt"),
        student(X, Y), write(X, ' ', Y), nl, fail.
```

Результат работы программы показан на рис. 4.1.

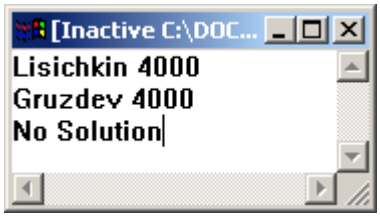

Рис. 4.1 Результат добавления фактов в ДБД.

В примере использован встроенный предикат FAIL, который всегда возвращает Л, он позволяет искусственно вызвать возврат, обеспечивая тем самым вывод всех фактов, содержащихся в ДБД. Модифицируем раздел цели в программе:

```
goal
        consult(“student.txt”),
        retract(student(“Gruzdev”, _)),
        save(“student.txt”),
        student(X, Y), write(X, ‘ ‘, Y), nl, fail.
```

Как показано на рис. 4.2., в ДБД после выполнения программы с измененной целью останется только один факт.

Рис. 4.2 Результат удаления факта из ДБД.

Попытка повторно выполнить программу даст результатом «No Solution», так к моменту выполнения в ДБД нет факта, удовлетворяющего шаблону STUDENT(“Gruzdev”, _), в результате чего RETRACT вернет Л.

В программе на языке PROLOG может быть использовано несколько разделов ДБД. Если используется несколько разделов, они именуются, а при работе с ДБД в рассмотренных выше предикатах указывается второй аргумент – имя раздела ДБД, к которому относится операция. Данный механизм иллюстрирует текст следующей программы:

```
domains
        name, group, department = string
database – stud
        student(name, group)
database – teach
        teacher(name, department)
goal
        asserta(student(“Gruzdev”, “4000”), stud),
        asserta(student(“Lisichkin”, “4000”) , stud),
        asserta(teacher(“Smorchkov”, “40”), teach),
        asserta(teacher(“Strochkov”, “40”) , teach),
        save(“student.txt” , stud),
        save(“teacher.txt” , teach).
```

В предикатах для добавления и удаления фактов второй аргумент может опускаться, так как имя факта, с которым производится операция, позволяет однозначно определить соответствующий раздел ДБД. Операция SAVE создает текстовый файл, чтение отдельных фактов из него может быть осуществлено с помощью встроенного предиката READTERM(<имя домена>, <свободная переменная>). В качестве имени домена указывается имя раздела ДБД, а если раздел единственный – DBASEDOM. Использование READTERM иллюстрирует следующая программа (предполагается, что файл “teacher.txt” был заполнен в результате выполнения предыдущей программы):

```
domains
        name, group, department = string
        file = myfile
database – stud
        student(name, group)
database – teach
        teacher(name, department)
goal
        openread(myfile, “teacher.txt”), readdevice (myfile),
        readterm (teach, X), write(X), nl,
        readterm (teach, Y), write(Y), nl,
        closefile(myfile).
```

Результат выполнения программы приведен на рис. 4.3.

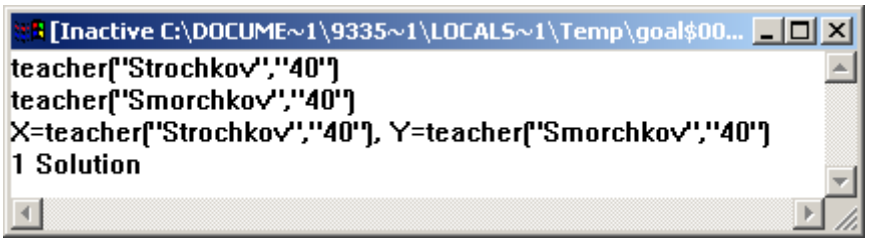

Рис. 4.3 Результат чтения с помощью READTERM.

В примере использован ряд встроенных предикатов языка: OPENREAD открывает файл для чтения, связывая имя файла ОС с файловым доменом, определяемым как FILE = MYFILE; READDEVICE переопределяет текущее устройство ввода; CLOSEFILE производит закрытие файла.

2. Составные домены

Помимо списков для хранения структурных данных в языке PROLOG могут быть использованы составные термы. Составной домен определяется следующим образом:

```
domains
        <имя составного домена> =
                <функтор 1>(<имя домена 1,1>, < имя домена 1,2>, …);
                <функтор 2>(<имя домена 2,1>, < имя домена 2,2>, …);
                …
```

Функторы, играют роль переключателей, позволяющих сопоставить одному домену несколько вариантов составных термов (механизм похож на технику использования записей с вариантами в процедурных языках программирования), точка с запятой в определениях доменов читается как «ИЛИ». Предположим, что публикацией может быть как книга, так и статья в сборнике, тогда для хранения информации о ссылках на публикации может быть предложена следующая структура данных:

```
domains
        name, article_name, author, publishing = string
        year, pages, first_page, last_page = integer
        authors = author*
        publication =  book(authors, name, publishing, year, pages);
                       article(authors, article_name, name, publishing, year,
                       first_page, last_page)
```

Таким образом, типу данных PUBLICATION будет соответствовать как составной терм book([“Booch”, “Rumbaugh”, “Jacobson”], "The Unified Modelling Language User Guide", “DMK”, 2000, 432), так и терм article([“Codd”], "A relational model of data for large shared data banks", “ACM Transactions on Database Systems”, “Comm. ACM”, 1970, 377, 387).

Используя списки, элементами которых являются составные термы, в программах на языке PROLOG можно организовывать рекурсивные структуры данных. Типовой рекурсивной структурой данных является дерево, приведенная ниже программа иллюстрирует способ представления дерева в программе на языке PROLOG:

```
domains
        tree = l(symbol); s(symbol, subtreelist)
        subtreelist=tree*
        symbollist=symbol*
predicates
        nondeterm tree_elem (symbol, tree)
        nondeterm member(tree, subtreelist)
clauses
        tree_elem(X, l(X)).
        tree_elem(X, s(X, _)).
        tree_elem(X, s(_, SubtreeList)):–
                member(Subtree, SubtreeList),
                tree_elem(X, Subtree).

        member(X, [X| _]).
        member(X, [_| Y]):– member(X, Y).
goal
        tree_elem(f,
                s(a,
                        [s(b,
                                [l(e), l(f)]),
                         s(c,
                                [l(g)]),
                         s(d,
                                [l(h), l(i)])])
        ).
```

Дерево – это либо лист (l(symbol)), который содержит один символ, либо вершина промежуточного уровня, содержащая как символ, так и список поддеревьев (s(symbol, subtreelist)). Первый клоз для TREE_ELEM соответствует случаю, когда искомый элемент

содержится в листе, второй – в вершине промежуточного уровня, третий с помощью MEMBER выбирает очередное поддерево из списка связанных с узлом поддеревьев и осуществляет рекурсивный вызов для просмотра выбранного поддерева. Дерево, заданное в разделе цели программы приведено на рис. 4.4.

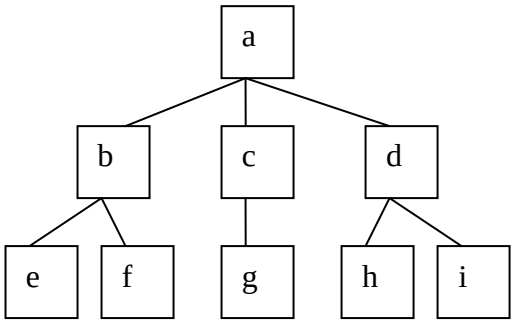

Рис. 4.4 Структура дерева.

Так как элемент f содержится в дереве, программа вернет результат «YES», если заменить в разделе цели константу f на свободную переменную X, будет произведен полный обход дерева и выданы все его элементы.

3. Пример выполнения лабораторной работы

Составим программу для ведения простого телефонного справочника. Информация о телефонах будет храниться в динамической базе данных в фактах для предиката PERSON, имеющего два аргумента: первый – составной терм с функтором NAME и двумя аргументами имя и фамилия; второй – список составных термов с функтором PHONE и двумя аргументами номер и тип (рабочий/домашний/мобильный/местный). Текст программы будет выглядеть следующим образом:

```
domains
        first_name, last_name, number, type = string
        name = n(first_name, last_name)
        phone = p(number, type)
        phonelist = phone*
        file=datafile
database
        person(name, phonelist)
predicates
        nondeterm init
        nondeterm main_menu
        nondeterm action(char)
        nondeterm add
        nondeterm add_person(first_name, last_name, number, type)
        nondeterm delete
        nondeterm delete_person(first_name, last_name, number, type, phonelist)
        nondeterm delete_phone(phone, phonelist, phonelist)
        nondeterm search_name
```

```
        nondeterm member(phone, phonelist)
        nondeterm search_number
        nondeterm all
        nondeterm eq(string,string)
        nondeterm eq(char,char)
        nondeterm sort
        nondeterm put
        nondeterm less(string)
goal
        init, main_menu.
clauses
        init:– existfile("database.dat"), consult("database.dat"), !.
        init:– write("File not found …"), nl.

        main_menu:– nl,
                write("Add, Delete, search Name, search nUmber, aLl, Sort, Quit"),
                nl, readchar(C), action(C), !, main_menu.

        action('a'):– add.
        action('d'):– delete.
        action('n'):– search_name.
        action('u'):– search_number.
        action('l'):– all.
        action('s'):– sort.
        action('q'):– save("database.dat"), exit.
        action(_).

        add:–   write("Add  …"), nl,
                write("First name:      "), readln(FN),
                write("Last name:       "), readln(LN),
                write("Phone number:            "), readln(PN),
                write("Phone type:      "), readln(PT),
                add_person(FN, LN, PN, PT).
        add:–   nl, write("Error …"), nl.

        add_person(FN, LN, PN, PT):–
                retract(person(n(FN, LN), PL)),
                asserta(person(n(FN, LN), [p(PN, PT)| PL])).
        add_person(FN, LN, PN, PT):–
                asserta(person(n(FN, LN), [p(PN, PT)])).

        delete:–
                write("Delete …"), nl,
                write("First name:      "), readln(FN),
                write("Last name:       "), readln(LN),
                write("Phone number:            "), readln(PN),
                write("Phone type:      "), readln(PT),
                retract(person(n(FN, LN), PL)),
                delete_person(FN, LN, PN, PT, PL).
        delete:–
                nl, write("Error …"), nl.
```

```
delete_person(_, _, PN, PT, [p(PN, PT)]).
delete_person(FN, LN, PN, PT, PL):–
        delete_phone(p(PN, PT), PL, PL1),
        asserta(person(n(FN, LN), PL1)).

delete_phone(_, [], []).
delete_phone(X, [X| Y], Y).
delete_phone(X, [_| Y], [X| Z]):–
        delete_phone(X, Y, Z).

search_name:–
        write("search Name …"), nl,
        write("Phone number:          "), readln(PN),
        person(n(FN, LN), PL),
        member(p(PN, _), PL),
        write("First name:     ", FN), nl,
        write("Last name:      ", LN), nl.
search_name:–
        nl, write("Error …"), nl.

member(X, [X| _]).
member(X, [_| Y]):–  member(X,Y).

search_number:–
        write("search nUmber …"), nl,
        write("Last name:      "), readln(LN),
        person(n(_, LN), PL),
        write("Phones:                ", PL), nl.
search_number:–
        nl, write("Error …"), nl.

all:–   write("aLl …"), nl,
        person(n(FN, LN), PL),
        write("First name:     ", FN), nl,
        write("Last name:      ", LN), nl,
        write("Phones:                ", PL), nl, nl,
        readchar(C), eq(C,'\27').
all.

eq(X,X).

sort:–  write("Sort …"), nl, fail.
sort:–  deletefile("database.dat"), fail.
sort:–  openwrite(datafile,"database.dat"),
        writedevice(datafile), fail.
sort:–  put.
sort:–  writedevice(screen),
        closefile(datafile),
        consult("database.dat").
sort:–  write("File not found …"), nl.

put:–   person(n(FN, LN), PL),
```

```
        not(less(LN)),
        retract(person(n(FN, LN), PL)),
        write(“person(n(“, ””', FN, ””', ',' , ””', LN, ””', “), “,PL, ‘)’), nl,
        !, put.

less(N):–
        person(n(_, N1),_), N1 < N.
```

Пример диалога с программой может выглядеть следующим образом:

```
File not found …

Add, Delete, search Name, search nUmber, aLl, Sort, Quit
Add  …
First name:         John
Last name:          Lisichkin
Phone number:       7775533
Phone type:         home

Add, Delete, search Name, search nUmber, aLl, Sort, Quit
Add  …
First name:         John
Last name:          Lisichkin
Phone number:       1112233
Phone type:         office

Add, Delete, search Name, search nUmber, aLl, Sort, Quit
search Name …
Phone number:       1112233
First name:         John
Last name:          Lisichkin

Add, Delete, search Name, search nUmber, aLl, Sort, Quit
search nUmber …
Last name:          Lisichkin
Phones:             [p("1112233","office"),p("7775533","home")]

Add, Delete, search Name, search nUmber, aLl, Sort, Quit
search nUmber …
Last name:          x

Error …
```

Работа программы начинается с вызова INIT, в первом клозе с помощью встроенного предиката EXISTFILE проверяется существование файла с данными и, если он существует, данные загружаются в ДБД. EXISTFILE возвращает Л, если файл не найден, для того что бы работа программы не прекратилась введен второй клоз для INIT, таким образом INIT будет всегда И, при этом, если для подтверждения его истинности использован 2-й клоз, будет выведено сообщений об отсутствии файла.

Предикат MAIN_MENU вызывается следующим за INIT и служит для организации диалога с пользователем, он выводит приглашение для выбора выполняемой операции, считывает введенный пользователем символ и вызывает ACTION для выполнения необходимых действий. Встроенный предикат READCHAR имеет один аргумент, в качестве которого при вызове указывается переменная, после выполнения READCHAR с переменной будет связано значение символа, введенного пользователем. Перед рекурсивным вызовом MAIN_MENU использован оператор отсечения (!) для того что бы исключить сохранение в стеке возврата информации о возможных альтернативных решениях для ACTION и предикатов, которые он вызывает.

Завершение работы программы происходит, если пользователь ввел символ 'q', для этого используется встроенный предикат EXIT, перед завершением работы содержимое ДБД сохраняется в дисковом файле. В случае, если пользователь ввел недопустимый символ, срабатывает последний клоз для предиката ACTION, никаких действий при этом не производится.

Предикат ADD запрашивает и вводит информацию о человеке и телефоне, после чего обращается к ADD_PERSON. Встроенный предикат READLN связывает указанную при его вызове переменную со строкой, введенной пользователем. Предикат возвращает значение И, если пользователь закончил ввод нажатием клавиши ENTER и Л, если была нажата ESC. В последнем случае для продолжения работы программы введен 2-й клоз для ADD.

ADD_PERSON обрабатывает две возможные ситуации, 1-й клоз удаляет информацию о человеке, имеющуюся в ДБД, добавляет новый номер в голову списка телефонных номеров и сохраняет измененную информацию в ДБД. Если факта, содержащего информацию о данном человеке в ДБД нет, то он добавляется, при этом список телефонов содержит один единственный элемент.

Предикат DELETE работает аналогично ADD, и вызывает DELETE_PERSON, который обрабатывает как случай, когда удаляется последний телефонный номер (1-й клоз), так случай, когда в списке телефонов есть несколько элементов (2-й клоз). Для удаления номера из списка используется вспомогательный предикат DELETE_PHONE, который удаляет номер, указанный при его вызове в качестве 1-го аргумента, из списка, указанного во 2-м аргументе, после завершения работы DELETE_PHONE с 3-м аргументом связывается результат обработки. DELETE_PHONE возвращает И независимо от того, найден удаляемый элемент в списке или нет.

Предикат SEARCH_NAME находит информацию о человеке по заданному телефонному номеру, для просмотра списка телефонных номеров используется вспомогательный предикат MEMBER (см. лабораторную работу №2). Предикат SEARCH_NUMBER выполняет

симметричную операцию – находит список телефонных номеров по введенной пользователем фамилии.

Предикат ALL служит для просмотра всего содержимого ДБД. Вывод сведений об очередном человеке происходит после нажатия пользователем клавиши на клавиатуре. Если код нажатой клавиши – 27, что соответствует ESC, просмотр завершается, в противном случае EQ(C,'\27') возвращает Л, происходит ВОЗВРАТ и выдается очередной факт ДБД. По исчерпании всех имеющихся фактов сработает второй клоз для ALL и выполнение программы будет продолжено.

Предикат SORT осуществляет сортировку по фамилии посредство удаления фактов из ДБД, записи их в файл и последующей загрузки ДБД из файла. 1-й клоз для SORT выводит служебное сообщение, 2-й обращается к встроенному предикату для удаления файла, 3-й открывает файл для записи и объявляет его текущим устройством вывода. В конце каждого из этих клозов стоит обращение к FAIL, всегда возвращающему Л, поэтому клозы будут срабатывать последовательно в порядке их записи в программе.

В 4-м клозе для SORT происходит обращение к PUT, он, используя LESS, находит в ДБД факт с фамилией, для которой выполняется условие – нет других фактов, где фамилия была бы меньше в алфавитном порядке. LESS возвращает И, если такие факты существуют, а NOT, инвертирует полученное значение. Далее PUT удаляет факт из ДБД и осуществляет запись в дисковый файл. Так как рано или поздно все факты ДБД будут удалены, PUT возвратит Л, после чего сработает 5-й клоз для SORT, в котором произойдет закрытие файла и загрузка его в ДБД. 6-й клоз для SORT срабатывает только в случае, если CONSULT возвратит Л, что соответствует ошибке ввода-вывода.

8. Варианты заданий

Используя динамическую базу данных, списки и составные термы реализовать программу для обработки предложенного ниже варианта набора данных, выполняющую:

— включение/исключение/замену данных;

— 4-5 поисковых запроса (разработать самостоятельно и согласовать с преподавателем].

1) Расписание авиарейсов.
2) Железнодорожное расписание.
3) Купля/продажа/сдача/наем/обмен недвижимости.
4) Телефонный справочник организации.
5) Учет результатов экзаменационных сессий.
6) Подбор персонала (вакансии и резюме].

7) Учет технических осмотров автомобилей в автопарке.
8) Планы изданий и заказ книг.
9) Бронирование номеров в гостиницах.
10) Учет выданных книг в библиотеке.
11) Учет выдачи фильмов в прокате.
12) Инвентаризация лабораторий университета (аудитории, установленная техника].
13) Маршруты городского транспорта.
14) Учет сотрудников в отделе кадров.
15) Репертуар театров.
16) Учет услуг абонентам мобильной связи.
17) Котировки ценных бумаг.
18) Запись к врачам в поликлинике.
19) Проведение научной конференции.
20) Учет товаров на складе.

Библиографический список

1. Братко И. Алгоритмы искусственного интеллекта на языке PROLOG./Пер. с англ. — М.: Вильямс, 2004. —640 с.
2. Братко И. Программирование на языке Пролог для искусственного интеллекта./Пер. с англ. — М.: Мир, 1990. — 560 с.
3. Стерлинг Л., Шапиро Э. Искусство программирования на языке Пролог./Пер. с англ. — М.: Мир, 1990. — 335 с.
4. Лорьер Ж.Л. Системы искусственного интеллекта./Пер. с франц. — М.: Мир, 1991 г. — 568 с.

Содержание