ФЕДЕРАЛЬНОЕ АГЕНТСТВО ПО ОБРАЗОВАНИЮ
Белгородский государственный технологический университет
им. В.Г. Шухова

В.В. Шаптала

Представление знаний в информационных системах

Методические указания

Белгород
2007

ФЕДЕРАЛЬНОЕ АГЕНТСТВО ПО ОБРАЗОВАНИЮ
Белгородский государственный технологический университет
им. В.Г. Шухова

В.В. Шаптала

Представление знаний в информационных системах

Утверждено советом университета в качестве методических указаний для студентов специальности 230201 – Информационные системы и технологии

Белгород
2007

УДК 681.3.016 (075)
ББК 32.973.233-018я7
Ш24

Рецензенты:

Кандидат технических наук, доцент кафедры математического и программного обеспечения информационных систем Белгородского государственного университета *Тубольцев М.Ф.*

Шаптала В.В.
Ш24 Представление знаний: методические указания/ В.В. Шаптала. – Белгород: Изд-во БГТУ им. В.Г. Шухова, 2007. – 33 с.

Методические указания посвящены изучению основ языка программирования Пролог как формализма, предназначенного для представления знаний.

Данные методические указания по курсу “Представление знаний в информационных системах” предназначены для студентов 230201 – Информационные системы и технологии.

УДК 681.3.016(075)
ББК 32.973.233-018я7

Содержание

Введение

Предположим, что известна логическая спецификация, описывающая подлежащую решению задачу или структуру системы, которую необходимо представить. При использовании процедурного языка требуется проведение большой работы по переводу спецификации сложной системы в работоспособную программу. При использовании функциональных языков программист может мыслить в терминах вычисления значений, а не поведения компьютера. Семантика конструкций реляционных языков наиболее близка к спецификации задачи. Смысл конструкции в реляционном языке определяется как отношение между отдельными сущностями или классами сущностей. С декларативной точки зрения реляционную конструкцию можно интерпретировать как *формулировку того, что существует отношение между аргументами, представленное именем этого отношения*. Одним из реляционных языков является Пролог. При должном использовании языка Пролог взгляд программиста на мир может подняться до уровня логической спецификации. Если программист сможет мыслить целиком в терминах структуры отношений, не заботясь о реализации, то у него освободится время для более важных целей. Пролог – наиболее известный язык логического программирования. Логическое программирование – это один из подходов к информатике, при котором в качестве языка высокого уровня используется логика предикатов первого порядка в форме фраз Хорна. Логика предикатов первого порядка – это универсальный абстрактный язык предназначенный для представления знаний и для решения задач. Его можно рассматривать как общую теорию отношений. Логическое программирование дает возможность описывать ситуацию при помощи формул логики предикатов, а затем, для выполнения выводов из этих формул применить автоматический решатель задач.

Язык Пролог можно успешно использовать как для решения многих традиционных задач (программная инженерия, интерфейсы баз данных, системы помощи в принятии решений), так и для решения задач искусственного интеллекта (экспертные системы, системы обработки естественного языка).

Лабораторная работа №1

Факты и правила

Основные понятия

Программа на Прологе состоит из множества фраз, и ее можно рассматривать как сеть отношений, существующих между термами. *Терм* обозначает некоторую сущность, принадлежащую миру. *Фраза* – это либо факт, либо правило. *Факт* – это утверждение о том, что соблюдается некоторое конкретное отношение.

Пример 1: знает (оля, витя).

Данный факт состоит из имени предиката *знает* и списка термов, заключенного в скобки.

Одним из видов термов являются атомы. *Атом* – это константа, которая обычно записывается в виде некоторого слова, начинающегося с маленькой буквы. Термы “оля” и “витя” являются атомами.

Предикат может обладать произвольным количеством аргументов.

Пример 2.

Нижеследующий факт показывает, что БГТУ расположен по адресу ул. Костюкова 48.

расположение(бгту, костюкова, 48).

Простейшая Пролог-программа - это множество фактов, которое неформально называют *базой данных*.

Пример 3.

Данная база данных, состоит из фактов “знает”:

знает(оля, витя).
знает(коля, света).
знает(коля, витя).

Ввод программы

Пролог-программу, можно ввести в компьютер одним из двух способов: 1) при помощи текстового редактора создается файл с программой, которая затем загружается интерпретатором Пролога; 2) программа вводится во время сеанса работы с интерпретатором Пролога.

Непосредственный ввод программы будет происходить так:

Пример 4.

```
?- consult(user).
знает(оля, витя).
знает(коля, света).
знает(коля, витя).
quit.
?-
```

Ввод команды consult(user), что в переводе на русский язык означает “просмотр(пользователь)”, переключает интерпретатор в режим ввода программы. Ввод команды quit возвращает интерпретатор обратно в командный режим, о чем свидетельствует появление сообщения подсказки ?-.

Запросы к базе данных

Простой запрос состоит из имени предиката, за которым располагается список аргументов. (Синонимом слова *запрос* является слово *цель*). Если база данных “знает” просмотрена интерпретатором, то можно написать, например, такой запрос к ней:

```
?- знает (оля, витя).
```

Это означает вопрос: “Знает ли Оля Витю?”. Интерпретатор найдет соответствующий факт в базе данных и ответит:

```
yes.
```

Запросы с переменными

Переменная – это вид терма, который записывается как слово, начинающееся с большой латинской буквы.

Пример 5.

```
?-знает(оля, X).
```

означает: “Кого знает Оля?”

В запросе, содержащем переменную, неявно спрашивается о том, существует ли хотя бы одно значение этой переменной, при котором запрос будет истинным. Приведенный запрос можно прочесть так: “Существует ли хотя бы один человек, которого знает Оля?”

Этот запрос будет истинным, если такое лицо будет найдено в текущей базе данных “знает”.

Интерпретатор пытается со- гласовать аргументы запроса с аргументами фактов, входящих в базу данных “знает”.

В рассматриваемом случае запрос окажется успешным при сопоставлении запроса с первым же фактом, поскольку атом “оля” в запросе унифицируется с атомом “оля” в факте, а переменная X унифицируется с атомом “витя”, входящем в факт. В результате переменная X примет значение “витя” и интерпретатор ответит:

X = витя.

Составной запрос образуется из нескольких простых запросов, соединенных между собой символом “,” (запятая), что означает “и” или символом “;” (точка с запятой), что означает “или”. Каждый простой запрос, входящий в составной, называется “подцелью”. Истинность составного запроса зависит от истинности подцелей.

Пример 6.

?- знает(оля, A), знает(коля, A).

Данный запрос соответствует вопросу: “Есть ли такой человек, которого знают одновременно и Оля и Коля?”. Одна и та же переменная A входит в обе подцели этого запроса, а это означает, что для истинности всего составного запроса вторые аргументы обеих подцелей должны принимать одно и то же значение. Интерпретатор ответит:

A = витя ,

no

Первый ответ A = витя, показывает, что Витю одновременно знают и Оля и Коля. Второй ответ no свидетельствует, что Витя – единственный общий знакомый.

Символ подчеркивания _ выступает в качестве анонимной переменной, которая предписывает интерпретатору проигнорировать значение аргумента. Анонимная переменная унифицируется с чем угодно, но не обеспечивает выдачу выходных данных.

Пример 7.

В результате запроса:

?- знает(B, _).

будут выданы все возможные значе- ния первого аргумента предиката “знает”, вне зависимости от того, какие значения будет принимать второй аргумент:

B = оля,
B= коля,
B=коля,
no

Правила

Правило – это факт, значение истинности которого зависит от истинных значений условий, образующих тело правила.
Форма записи правила:

заголовок :- тело.

Пример 8.

начальник(Familia, Oklad) :- служ(Familia, Oklad), Oklad > 100.

Заголовок правила имеет такую же форму как и факт. Обозначение :- читается как “если”, затем следует тело правила. Каждое условие, входящее в тело, называется подцелью. Для того чтобы заголовок правила оказался истинным, необходимо, чтобы каждая подцель, входящая в тело была истинной.

Пример 9.

Предположим, что создана база данных “раб_смена”. Каждый факт этой базы определяет смену, в которую работает служащий.

раб_смена(оля, дневная).
раб_смена(коля, вечерняя).
раб_смена(витя, вечерняя).
раб_смена(толя, дневная).

Следующее правило устанавливает, что два человека знают друг друга, если они работают в одну и ту же смену:

знает2(A,B):- раб_смена(A, Smena), раб_смена(B, Smena).

Пример 10.

В программе, текст которой приведен ниже, устанавливается: есть ли дуга между вершинами графа; связаны ли две вершины противоположно

направленными дугами; образуют ли три следующие друг за другом дуги треугольник:

```
% описание существующих дуг в графе
arc_(a,c).
arc_(b,c). arc_(c,d). arc_(d,a). arc_(d,e). arc_(c,b). arc_(a,f).
% условие того, что вершины A и B связаны двумя противоположно направленными дугами
line_(A,B):-arc_(A,B),arc(B,A).
% условие того, что три следующих друг за другом дуги образуют треугольник
tr(X,Y,Z):-arc_(X,Y),arc_(Y,Z),arc_(Z,X).
```

После запуска программы на выполнение в окне Dialog экрана появится строка ?- .

Введем, например line(X,Y). В результате получим:

X=b,Y=c
X=c,Y=b

т.е. найдено два решения. Точно так же можно ставить вопросы для отношений arc_ и tr. Например, если ввести: arc_(a,b), результатом будет *no* (нет), а если ввести *arc(a,f), - yes (да)*. На вопрос *arc_(F,c)* Пролог выдаст два решения: F = a и F = b.

Процедуры

Смысл фразы языка Пролог может быть понят либо с позиций декларативного подхода, либо с позиций процедурного подхода. Декларативный смысл подчеркивает статическое существование отношений. Порядок следования подцелей в правиле не влияет на декларативный смысл этого правила.

При процедурной трактовке подчеркивается последовательность шагов, которые выполняет интерпретатор при обработке запроса. Таким образом приобретает значение порядок следования подцелей в правиле.

Пример11.

База данных “путешествие” содержит факты, каждый из которых имеет по три аргумента. Каждый факт устанавливает, что можно совершить путешествие из одного города (1-й аргумент) в другой город (2-й аргумент) воспользовавшись некоторым видом транспорта (3-й аргумент).

```
путешествие(белгород,          орел,   поезд).
путешествие(брянск, дятьково, автобус).
путешествие (губкин, белгород, автобус).
путешествие(орел, брянск, поезд).
```

Два правила “можно_путешествовать” устанавливают либо прямую, либо косвенную связь между двумя городами. Косвенное отношение будет соблюдаться в том случае, если возможно путешествие из одного города в другой через третий – промежуточный город.

```
можно_путешествовать(A,B):- путешествие (A, B,_).
можно_путешествовать(A,B):- путешествие(A,C,_),
                            путешествие (C, B,_).
```

Считается, что между этими двумя правилами неявно присутствует соединитель “или”. С декларативных позиций оба этих правила можно прочесть так: путешествие из города А в город В будет возможным, если существует прямая транспортная связь между этими городами, либо можно совершить путешествие из города А в некоторый промежуточный пункт С, а затем добраться из города С в город В.

Процедурная трактовка данных правил будет иной:

Для того, чтобы найти способ добраться из города А в город В, необходимо либо найти прямую транспортную связь между городами, либо найти вид транспорта, связывающий город А и город С, а затем найти транспортную связь между городами С и В.

Пример 12.

Следующий запрос к базе данных можно_путешествовать:

Есть ли транспортная связь между городами Губкин и Орел?

```
?- можно_путешествовать(губкин, орел).
Yes
```

Ответ получен по второй фразе.

При обработке этого запроса интерпретатор вначале проверяет первую фразу. Если первая фраза дает отрицательный ответ, то интерпретатор переходит к проверке второй фразы.

Обработку последнего запроса можно представить в виде эквивалентной последовательности запросов:

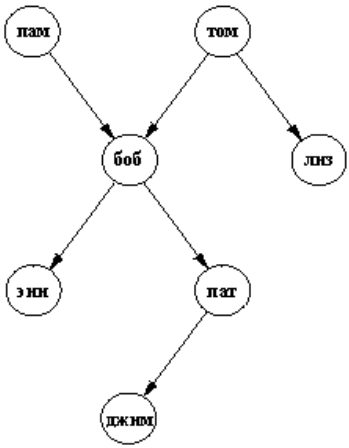
```
?-можно_путешествовать(губкин, орел).

    % первая подцель первого правила “можно_путешествовать”
```

?- путешествие (губкин, орел,_).
no
% первая подцель второго правила "можно_путешествовать"
?- путешествие (губкин, С,_).
С = белгород
% вторая подцель второго правила "можно_путешествовать"
?- путешествие (белгород, орел,_).
yes

Рекурсивные процедуры

Классическим примером рекурсивного определения в Прологе может служить программа «предок».

Предположим, имеем дерево родственных отношений:



Создадим соответствующую базу данных "родитель"
родитель(пам,боб).
родитель(том,боб).
родитель(том,лиз).
родитель(боб,энн).
родитель(боб,пат).
родитель(пат, джим).

Создадим также два правила:

```
предок(A,B):-  родитель(A,B).   % (1)
предок(A,B):- родитель(C,B), предок(A, C).          % (2)
```

Первое правило – для ближайших предков. Второе для отдаленных предков.

Любая рекурсивная процедура должна включать по крайней мере по одной из компонент:

1. Нерекурсивную фразу, определяющую исходный вид процедуры, т.е. вид процедуры в момент прекращения рекурсии.
2. Рекурсивное правило. Первая подцель, располагающаяся в теле этого правила, вырабатывает новые значения аргументов. Далее размещается рекурсивная подцель, в которой используются новые значения аргументов.

Фраза (1) процедуры предок определяет исходный вид этой процедуры. Как только данная фраза станет истинной, дальнейшая рекурсия прекратится. Фраза (2) – это рекурсивное правило. При каждом вызове данное правило поднимается на одно поколение вверх. Подцель родитель(C, B), входящая в тело этого правила, вырабатывает значение переменной C. Затем располагается рекурсивная подцель предок(A, C), в которой используется этот новый аргумент.

Содержание работы

1. Запустите интерпретатор Пролога. Воспользуйтесь командой “consult” для того, чтобы в диалоговом режиме ввести несколько фраз со сведениями о Вашей семье, либо со сведениями из какой-нибудь другой прикладной области, с которой Вы хорошо знакомы. Придумайте запросы к этим фразам, используйте в этих запросах и константы и переменные.
2. Воспользуйтесь редактором для создания текстового файла под названием “путешествие”. Внесите в этот файл приведенные выше факты “путешествие” и процедуру “можно_путешествовать”. Напишите запрос, в котором выясняется возможность путешествия из Белгорода в Брянск. Добавьте в программу правило, которое бы описывало возможность обратного путешествия.
3. Создайте рекурсивную процедуру “можно_путешествовать”. Напишите запрос, в котором выясняется возможность путешествия из Губкина в Дятьково.

Контрольные задания

1.Родственные отношения.

Кроме родственных отношений parent (родитель) и ancestor (предок) программа должна содержать хотя бы одно из следующих отношений:

1.1. brother (брат);

1.2. sister (сестра);

1.3. grand-father (дедушка);

1.4. grand-mother (бабушка);

1.5. uncle (дядя);

2.Ориентированные графы.

Описать граф. Задать отношения, позволяющие определить наличие в графе:

2.1. путей между произвольной парой вершин;

2.2. многоугольников с заданным числом сторон (например, четырехугольников).

3.Отношения likes ("нравится") и can_buy ("может купить").

Описать указанные отношения для следующих комбинаций "субъекты - предметы":

3.1.субъекты - фрукты;

3.2.субъекты - марки автомобилей;

3.3.субъекты - фильмы;

3.4.субъекты - книги.

4. Диагностика неисправностей

4. 1. в телевизоре

4.2. в автомобиле

4.3. в компьютере

5. Описать предметные области:

5.1. Торговый бизнес

5.2. Спортивный магазин

5.3. Аптека

5.4. Библиотека

Лабораторная работа №2

Арифметика. Списки.

Основные понятия

В языке Пролог имеется ряд встроенных предикатов. Предназначенных для вычисления арифметических выражений, выполнения арифметических сравнений. Обозначение операции записывается между аргументами без употребления скобок.

Предикат «is» (есть), предназначен для унификации первого аргумента и результата вычисления второго аргумента.

Пример 1.

```
?- X is 3, Y is (5*X)/2.
X = 3
Y = 7;
нет
```

Пример2.

```
?- 12 = < 24.
да

?- X is 50, X =\= 50.
нет
```

Все переменные входящие в арифметические выражения должны быть конкретизированы.

Структуры данных

Существует три типа термов: константа, переменная, составной терм.

Составные термы Пролога аналогичны записям Паскаля или структурам Си.

Пример 3.

```
сделка(клиент(петров,100,4), дата(2006,10,5)).
```

Факт сделка связывает информацию о сделке с клиентом.

Структура «клиент», содержит информацию о клиенте, берущем автомашину напрокат. В структуре содержится информация о фамилии клиента, суточном тарифе и количестве дней, на которое взята машина. Структура «дата» содержит информацию о годе, месяце и дне заключения сделки.

Пример 4.

Приведем правило, в котором аргументом служит структура «клиент», а в результате обработки правила подсчитывается сумма, которую должен заплатить клиент:

```
итого(A,Summa) :- сделка(клиент(A, Tarif, Dni), _), Summa is Tarif*Dni.
```

Запрос к правилу “итого” можно записать так:

```
?- итого(петров,S).

S = 400
```

Встроенный предикат “=”, имеет два аргумента. Этот предикат проверяет, унифицируются ли друг с другом его аргументы. Если в одном из аргументов содержатся неконкретизированные переменные, то в случае успешной унификации они конкретизируются.

Пример 5.

```
?- X is 10, Y is X+2, Y = 10.
no
```

Пример 6.

```
?- клиент(смит,29,4) = клиент(X,Y,Z).

X = смит
Y = 29
Z = 4
```

Рекурсивные структуры

Рекурсивные структуры аналогичны односвязным спискам, в которых один из аргументов каждой записи указывает на следующую запись.

Рекурсивная процедура *можно_путешествовать* предназначена для построения рекурсивной структуры, описывающей все промежуточные пункты, посещаемые по пути из исходного города в пункт назначения.

```
путешествие(губкин,белгород,автобус).
путешествие(белгород,орел,поезд).
путешествие(орел, брянск,поезд).
путешествие(брянск,дятьково,автобус).

можно_путешествовать(A,B,путь(A,VidTr,B)):-путешествие(A,B,VidTr).
можно_путешествовать(A,B,путь(A,VidTr,Put)):-путешествие(A,C,VidTr),
можно_путешествовать(C,B,Put).
```

Можно ли из Белгорода добраться до Дятьково? Если да, то какими видами транспорта?

```
?- можно_путешествовать(белгород, дятьково, X).

X = путь(белгород, поезд, путь(орел, поезд, путь(брянск, автобус, дятьково)))
```

Списки

Списки обозначаются следующим образом:
[терм1,терм2 | X]
Символ | разделяет список на две части: начало списка и остаток списка.

Пример 7. Унификация списка.

```
?- [F|R]=[1,2,3,4,5].

F = 1
R = [2, 3, 4, 5].  % остаток списка является списком
yes

?- [F1,F2|R]=[1,2,3,4,5].

F1 = 1
F2 = 2
```

```
R = [3, 4, 5].
yes
```

Применение рекурсивных процедур для обработки списков

Рекурсивная процедура должна состоять из фразы, определяющей условие окончания рекурсии, а затем следует фраза, выполняющая действия с заголовком списка, которая далее вызывает рекурсию сама себя с аргументом, которым служит остаток списка.

Пример 8. Вывод элементов списка.

```
% фраза, определяющая условие окончания рекурсии
печатать_элементы([ ]).

% рекурсивное правило
печатать_элементы(Первый|Остаток]):-
  write(Первый), nl,                    % вывод первого элемента
  печатать_элементы(Остаток).           % рекурсивный вызов
```

Пример. Процедура Найти_слово

```
найти_слово(Slovo,[]):-fail. % Поиск заканчивается если список пуст или
найти_слово(Slovo,[Slovo|Y]). % слово совпадает с первым элементом
списка
найти_слово(Slovo,[X|Y]):-найти_слово(Slovo,Y).


?-найти_слово(стул,[в, доме, есть, стул]).
yes.

?-найти_слово(забор,[в, доме, есть, стул]).
no.
```

Пример 9. Подсчет суммы элементов списка.

```
%  фраза, определяющая условие окончания рекурсии
сумма([],0):-!.

% рекурсивное правило
сумма([A|B],S):-сумма(B,S1),S is S1+A.
```

?- сумма([1,2,3],X).

X = 6

Пример 10. Объединение двух списков в третий.

присоединить([],S,S).
присоединить([X|S1], S2, [X|S3]):- присоединить (S1, S2,S3).

Содержание работы

1. Напишите составной запрос, в котором конкретизируются переменная X, получая значение 10, а затем конкретизируется переменная Y, которой присваивается значение, получаемое в результате умножения X на 3.
2. Напишите заново тот же составной запрос, добавив в него третью подцель, в которой проверяется, равно ли значение переменной Y числу 300.
3. Напишите правило, позволяющее вычислить площадь прямоугольника. У правила должно быть три аргумента: основание, высота, площадь.
4. Напишите запрос к процедуре «присоединить», в котором два существующих списка объединяются в один – третий список. Запишите запрос, генерирующий все возможные комбинации подсписков, которые могут быть сформированы из заданного списка.
5. Пусть имеется список структур “кл”: [(кл(a,29,3), кл(b,29,6), кл(c,40,2)]
   Первым аргументом каждой структуры служит имя клиента, вторым – суточный тариф, третьим – количество дней, на которое взята машина. Напишите правило, позволяющее вычислять итоговую сумму оплаты, объединяющую выплаты всех клиентов, данные о которых содержатся в списке.
6. Напишите новую версию процедуры “предок”, которая вырабатывает список представителей всех промежуточных поколений, располагающихся между предком и потомком.

Контрольные задания

1. Определить максимальный элемент в списке.
2. Определить число элементов в списке.
3. Определить произведение элементов списка.
4. Исключить из списка отрицательные элементы.
5. Выполнить сортировку элементов списка по возрастанию.
6. Даны два списка, имеющие ненулевое пересечение. Построить список, включающий все элементы указанных двух списков без повторений.
7. Определить отношение

Обращение(Список, Обращенный список),

которое располагает элементы списка в обратном порядке.

8. Определить отношение

перевод(Список1, Список2)

для перевода списка чисел от 0 до 9 в список соответствующих слов.

9. Определить отношение

разбиение_списка(Список, Список1, Список2)

так, чтобы оно распределяло элементы списка между двумя списками Список1 и Список2, длины которых отличаются друг от друга не более чем на единицу.

10. Определить отношение

пересечение(Список1, Список2, Список3),

где элементы списка Список3 являются общими для списков Список1 и Список2.

11. Определить отношение

разность(Список1, Список2, Список3),

где элементы списка Список3 принадлежат Списку1, но не принадлежат Списку2.

12. Определить отношение

element_mult(List1,List2,List3),

в котором элементы списка List3 равны произведениям соответствующих элементов списков List1 и List.

13. Определить отношение

shift(List1,List2)

таким образом, чтобы список List2 представлял собой список List1, "циклически сдвинутый" влево на один символ.

14. Треугольное число с индексом N - это сумма всех натуральных чисел до N включительно. Напишите программу, задающую отношение

triangle(N,T),

истинное, если T - треугольное число с индексом N.

15. Написать программу, задающую предикат

power(X,N,V),

истинный, если $V = X^N$.

16. Написать программу, задающую отношение

fib(N,F),

для определения N - го числа Фибоначчи F.

17. Написать программу вычисления скалярного произведения векторов inner_product(X,Y,V), где X и Y - списки целых чисел, V - скалярное произведение.

18. Написать программу

fact(N,F),

вычисляющую факториал F числа N.

Лабораторная работа № 3

Управление ходом выполнения программы

Основные понятия

Работу интерпретатора языка Пролог можно трактовать как рекурсивный циклический процесс унификации и вычисления подцелей. Действия интерпретатора инициируются запросом. В ходе выполнения этих действий интерпретатор «опустится» в структуру текущей программы настолько глубоко, насколько это окажется необходимым для того, чтобы найти факты, требующиеся для определения истинностного значения запроса. Затем интерпретатор вернется в исходное состояние, доказав или оказавшись не в состоянии доказать истинность запроса.

После того как пользователь вводит запрос интерпретатору, этот запрос активизируется. Интерпретатор приступает к анализу фраз текущей программы в поисках первой фразы, заголовок которой будет унифицироваться с запросом.

Неудача запроса и возврат назад

Если активный запрос достигает конца соответствующего множества фраз, то он завершится неудачей. Если такой активный запрос служит частью составного запроса и не является первой подцелью этого составного запроса, то интерпретатор возвратится назад, чтобы повторно

проанализировать предыдущую подцель составного запроса. Если активный запрос является первой подцелью составного запроса, то неудача активного запроса приводит к неудаче всего составного запроса. Когда интерпретатор возвращается назад, ликвидируются все конкретизации переменных, выполненные последним активным запросом.

Указание интерпретатору вернуться назад

После того как интерпретатор найдет один ответ на запрос, пользователь может попросить найти еще один ответ. Для этого вводится символ ; , который означает отказ от только что полученного ответа. Это заставляет интерпретатор возвратиться назад и приступить к поиску другого ответа. Точнее, ввод символа ; приводит к неудаче запроса, активизированного самым последним (т.е. запроса, расположенного в вершине стека).

Предикат «Сократить»

Пространство поиска запроса – это множество всех возможных ответов, рассматриваемых интерпретатором при выполнении запроса. Существует специальный встроенный предикат «сократить», который дает указание интерпретатору не возвращаться назад далее той точки, где стоит этот предикат.

***Пример** 1.*

?- a(X), b(Y), !, c(X,Y,Z).

При выполнении данного запроса интерпретатор пройдет через предикат «сократить» только в том случае, если подцель a(X) и b(X) окажутся успешными. После того как предикат «сократить» будет обработан интерпретатор не сможет возвратиться назад для повторного рассмотрения подцелей «a» и «b», если подцель «c» потерпит неудачу при текущих значениях переменных X и Y.

Проверка типа терма

В языке Пролог имеются встроенные предикаты, предназначенные для проверки типа терма.

var(X) - предикат даст значение истина, если его аргумент будет не конкретизированной переменной.

Пример2 .

?- var(X)
да
?- X = лондон, var(X).
нет

nonvar(X) - предикат будет истинным, если его аргумент будет термом любого вида, кроме не конкретизированной переменной.

Пример 3.

?- X = [париж, лондон, нью-йорк, токио], nonvar(X).
да

Действия с текущей программой

Существуют встроенные предикаты, позволяющие программными средствами изменять текущее множество фраз программы.
Предикат assert (принять) добавляет к текущей программе фразу X.

Пример 4.

?- assert(король(людовиг, франция)).
да
?- король(людовиг, X).
X = франция

Предикат retract (удалить) удаляет из текущей программы первую фразу, которая унифицируется с X.

Пример 5.

?- retract(король(людовиг, франция)).
да
?- король(людовиг, X).
нет

Задание итерации

Цель repeat порождает новую ветвь вычислений.

Пример 6.

Данная программа изменяет значение счетчика counter(i) в базе данных от i=0 до i=100.

```
count :- assert(counter(0)),fail.
count :- repeat, counter(X), Y=X+1, retract(counter(X)),
aserta(counter(Y)), write(Y), Y = 100.

?- counter(X).
X = 100
```

Содержание работы

1. Напишите при помощи предиката «сократить» составной запрос к базе данных «путешествие», который будет находить только один город, в который можно добраться из Белгорода, а затем будет отыскивать все города, в которые можно отправиться из найденного города на автобусе.
2. Напишите обратимую версию процедуры, вычисляющей площадь прямоугольника. Используйте предикаты var(X) и nonvar(X).
3. Напишите с использованием предиката «repeat» составной запрос, который спрашивает у пользователя имена школьных товарищей и добавляет каждое имя в базу данных в виде факта: школьный_товарищ(Имя).
   После того, как пользователь введет слово «конец», запрос должен прекратить задавать вопросы и выдать на экран все только что введенные имена.
4. Выберите некоторую форму представления базы данных, в которой содержатся сведения об операциях с кредитными карточками. Каждая запись должна содержать сведения об имени лица, тратящего деньги, о типе операции и о сумме денег. Напишите процедуру, которая будет выдавать значение итоговой суммы всех операций для конкретного лица.

Контрольные задания

1.Создать БД, содержащую сведения о пассажирах:
Ф.И.О., количество мест, вес багажа.
Определить, есть ли пассажиры, багаж которых занимает 1 место и вес багажа больше 30 кг.

2.Создать БД о студентах вашей группы:
Фамилия, Имя, Год рождения.
Получить список студентов старше 20 лет.

3.Соэдать БД, содержащую сведения:
Ф.И.О., профессия, оклад.
Найти среднемесячную заработную плату для инженеров.

4.Создать БД о группе студентов:
Фамилия, Имя.
Выяснить, имеются ли в группе однофамильцы.

5.Создать БД о металлах:
Наименование, Удельная проводимость, Удельная стоимость.
Найти металлы с максимальной проводимостью и минимальной стоимостью.

6.Создать БД с расписанием движения поездов:
Номер поезда,
Пункт назначения,
Время отправления,
Время в пути,
Стоимость билета.
Найти номер и время отправления самого скорого поезда до Москвы.

7.Создать БД с расписанием движения самолетов:
Номер рейса,
Пункт отправления,
Пункт прибытия,
Время отправления,
Время в пути,
Стоимость билета.
Определить маршрут движения из Новосибирска в Нью-Йорк, время в пути и стоимость проезда.

8. Создать БД с таблицей игр чемпионата по футболу:
Первая команда, Вторая команда, Счет игры.
Определить чемпиона.

9. Создать БД с книжным каталогом:
Ф.И. автора, Название книги, Издательство, Год издания.
Найти все книги, изданные в издательстве "Наука" после 1990 года.

10. Создать БД со сведениями о стоимости товаров:
Наименование товара, Стоимость товара.
Определить суммарную стоимость указанных в БД товаров, найти товары с максимальной и минимальной стоимостями.

Лабораторная работа № 4.

Операции на графах

Представление ориентированных графов в Прологе

Способ 1.

Каждая дуга графа записывается в виде отдельного предложения.

```
arca(a,b). arca(b,c).
```

или (граф с взвешенными дугами)

```
arca(s,t,1). arca(t,v,3). arca(v,u,2).
```

Способ 2.

Граф представляется в виде списка дуг. Например,

```
G = [ arca(a,b), arca(b,c), arca(b,d), arca(c,d)]
```

или

```
G = [ arca(s,t,3), arca(t,v,1), arca(v,u,2), arca(u,t,5), arca(t,u,2)]
```

Способ 3.

Граф представляется как один объект. Графу соответствует пара множеств - множество вершин и множество дуг. Для объединения множеств в пару будем применять функтор graph, а для записи дуги - arca. Например,

```
G = graph([a,b,c,d], [ arca(a,b), arca(b,d), arca(b,c), arca(c,d)])
```

Всюду, где это возможно, для простоты записи программы будем представлять графы способом 1 или способом 2.

Операции на графах

Типичными операциями на графе являются следующие:

- найти путь между двумя заданными вершинами графа;
- найти подграф, обладающий заданными свойствами (например, построить остовное дерево графа).

Поиск пути в графе

Определим отношение path(A,Z,P),
где P - ациклический путь между вершинами A и Z в графе G, представленном следующими дугами:

arca(a,b). arca(b,c). arca(c,d). arca(b,d).

Один из методов поиска пути основан на следующих соображениях:
- если A = Z, то положим P = [A];
- иначе нужно найти ациклический путь P1 из произвольной вершины Y в Z, а затем найти путь из A в Y, не содержащий вершин из P1.

Введем отношение path1(A,P1,P),
означающее, что P1 - завершающий участок пути P.

Между path и path1 имеет место соотношение:

path(A,Z,P) :- path1(A,[Z],P).

Рекурсивное определение отношения path1 вытекает из следующих посылок:
- "граничный случай": начальная вершина пути P1 совпадает с начальной вершиной A пути P;
-в противном случае должна существовать такая вершина X, что:) Y - вершина, смежная с X, 2) X - не содержится в P1, 3) для P выполняется отношение path(A,[Y|P1],P).

Пример 1. Программа нахождения пути.

```
arca(a,b). arca(b,c). arca(c,d). arca(b,d).
member(X,[X|_]).
member(X,[_|Tail]):- member(X,Tail).
path(A,Z,Path):- path1(A,[Z],Path).
path1(A,[A|Path1],[A|Path1]).
path1(A,[Y|Path1],Path):- arca(X,Y), not(member(X,Path1)),
        path1(A,[X,Y|Path1],Path).
```

Чтобы отношения path и path1 могли работать со стоимостями (весами), их нужно модифицировать введением дополнительного аргумента для каждого пути:

```
path(A,Z,P,C),
path1(A,P1,C1,P,C),
```

где C и C1 - стоимости путей P и P1 соответственно.

Отношение смежности arca включает дополнительный аргумент - стоимость дуги:

```
arca(X,Y,C).
```

Пример 2. Программа построения пути минимальной стоимости между заданными вершинами графа.

```
arca(a,b,1).    arca(b,c,3).    arca(c,d,1).    arca(b,d,7).
arca(a,d,1).
member(X,[X|_]).
member(X,[_|Tail]):- member(X,Tail).
path(A,Z,Path,C):- path1(A,[Z],0,Path,C).
path1(A,[A|Path1],C,[A|Path1],C).
path1(A,[Y|Path1],C1,Path,C):- arca(X,Y,CXY),
not(member(X,Path1)),
        C2=C1+CXY, path1(A,[X,Y|Path1],C2,Path,C).

% поиск пути минимальной стоимости между вершинами X и Y
db0(X,Y) :-path(X,Y,P,C), assert(db_path(X,Y,P,C)).
db(X,Y):-db_path(X,Y,P,C), path(X,Y,MP,MC), MC<C,!,
        retract(db_path(X,Y,P,C)), assert(db_path(X,Y,MP,MC)),
db(X,Y).
db(_,_).
```

Следующий запрос определяется путь MP минимальной стоимости MC между вершинами a и b:

```
db0(a,d), db(a,d), db_path(a,d,MP,MC),
write("\nMP=",MP,"\nMC=",MC).
```

Построение остовного дерева

Граф называется связным, если между любыми двумя его вершинами существует путь. Остовное дерево графа G=(V,E) - это связный граф T=(V,E1) без циклов,в котором E1 - подмножество E.

Определим процедуру

osttree(e,T),

где T - остовное дерево графа G (G - связный граф),e - произвольно выбранное ребро графа G.

Остовное дерево можно строить так: 1) начать с произвольного ребра графа G; 2) добавлять новые ребра, постоянно следя за тем, чтобы не образовывались циклы; 3) продолжать этот процесс до тех пор, пока не обнаружится, что нельзя присоединить ни одного ребра, поскольку любое новое ребро порождает цикл. Отсутствие цикла обеспечивается так: ребро присоединяется к дереву лишь в том случае, когда одна из его вершин уже содержится в строящемся дереве, а другая пока еще не включена в него.

Основное отношение, используемое в программе,

expand(Tree1,Tree).

Здесь Tree - остовное дерево, полученное добавлением множества ребер из G к дереву Tree1.

Пример 3. Построение остовного дерева.

```
arca(a,b). arca(b,c). arca(c,d).        arca(b,d).
member(X,[X|_]).
member(X,[_|Tail]):-member(X,Tail).
osttree(arca(A,B),Tree) :- expand([arca(A,B)],Tree).
expand(Tree1,Tree) :- ap_arca(Tree1,Tree2), expand(Tree2,Tree).
expand(Tree,Tree) :-not(ap_arca(Tree,_)).
ap_arca(Tree,[arca(A,B)|Tree]) :- arca(A,B), node(A,Tree),
not(node(B,Tree));
            arca(A,B),node(B,Tree), not(node(A,Tree)).
node(A,Tree) :- member(arca(A,_),Tree); member(arca(_,A),Tree).
```

Контрольные задания

1. Определить, является ли связным заданный граф.
2. Найти все вершины графа, к которым существует путь заданной длины от выделенной вершины графа.
3. Найти все вершины графа, достижимые из заданной.
4. Подсчитать количество компонент связности заданного графа.
5. Найти диаметр графа, т.е. максимум расстояний между всевозможными парами его вершин.
6. Найти такую нумерацию вершин орграфа, при которой всякая дуга ведет от вершины с меньшим номером к вершине с большим номером.
7. Дан взвешенный граф. Построить остовное дерево минимальной стоимости.
8. Определить является ли граф гамильтоновым. Найти гамильтонов цикл, т. е. цикл, проходящий через все вершины графа.
9. Задана система односторонних дорог. Найти путь, соединяющий города A и B и не проходящий через заданное множество городов.
10. В заданном графе указать все его четырехвершинные полные подграфы.
11. Известно, что заданный граф - не дерево. Проверить, можно ли удалить из него одну вершину (вместе с инцидентными ей ребрами), чтобы в результате получилось дерево.

Лабораторная работа № 5.

Основные стратегии решения задач

Основные понятия

Пространством состояний задачи является граф, вершины которого соответствуют ситуациям, встречающимся в задаче (" проблемные ситуации"), а решение задачи сводится к поиску пути в этом графе.

Пространство состояний задачи определяет "правила игры":

- вершины пространства состояний соответствуют ситуациям;
- дуги соответствуют разрешенным ходам или действиям, или шагам решения задачи.

Конкретная задача определяется: пространством состояний; стартовой вершиной; целевым условием (т. е. условием, к выполнению которого нужно стремиться).

Целевыми называются вершины, удовлетворяющие целевым условиям.

Каждому разрешенному ходу или действию можно приписать его стоимость. В тех случаях, когда каждый ход имеет стоимость, мы заинтересованы в отыскании решения минимальной стоимости.

Стоимость решения - это сумма стоимостей дуг, из которых состоит решающий путь - путь из стартовой вершины в целевую.

Будем представлять пространство состояний при помощи отношения *after(X,Y)*, которое истинно тогда, когда в пространстве состояний существует разрешенный ход из вершины X в вершину Y. Будем говорить, что Y – это преемник вершины X. Если с ходами связаны их стоимости, мы добавим третий аргумент, стоимость хода C: *after(X,Y,C)*.

Эти отношения можно задавать в программе явным образом при помощи набора соответствующих фактов. Однако, такой принцип непрактичен, поэтому отношение следования *after* обычно определяется неявно, при помощи правил вычисления вершин преемников некоторой заданной вершины.

Пример 1. Задача манипулирования кубиками.

Проблемная ситуация - список столбиков. Каждый столбик - список кубиков, из которых он составлен. Кубики упорядочены в списке таким образом, что самый верхний кубик находится в голове списка. "Пустые" столбики изображаются как пустые списки.

Отношение следования вытекает из правила:

Ситуация Sit2 - преемник ситуации Sit1, если в Sit1 имеется два столбика Stolb1 и Stolb2 такие, что верхний кубик из Stolb1 можно поставить сверху на Stolb2 и получить тем самым Sit2.

Поскольку все ситуации - списки столбиков, правило транслируется на Пролог так:

```
after(Stolbs,[Stolb1, [Up1|Stolb2],Rest]) :-
        delete([Up1|Stolb1],Stolbs,Stolbs1),
        delete(Stolb2,Stolbs1,Rest).
delete(X,[X|L],L).
delete(X,[Y|L],[Y|L1]) :- delete(X,L,L1).
```

Здесь:

- Stolbs - множество столбиков в ситуации Sit1;
- Stolbs1 - множество столбиков без первого столбика;
- Rest - множество столбиков без первого и второго.

Целевое условие в данном примере имеет вид:

```
goal(Sit) :- member([a,b,c],Sit).
```

Алгоритм поиска программируется как отношение

```
solve(Start,Solution),
```

где

- Start - стартовая вершина пространства состояний,
- Solution - путь, ведущий из вершины Start в любую целевую вершину.

Для конкретного примера обращение к Пролог- системе имеет вид:

```
solve([[c,a,b],[],[]],Solution).
```

Исходная ситуация

```
[[c,a,b],[],[]].
```

Целевые ситуации:

```
[[a,b,c],[],[]]
[[],[a,b,c],[]]
[[],[],[a,b,c]]
```

Список Solution представляет собой план преобразования исходного состояния в состояние, в котором три кубика поставлены друг на друга в указанном порядке : [a,b,c].

Стратегия поиска в глубину

Основные стратегии поиска решающего пути - поиск в глубину и поиск в ширину. Идея алгоритма поиска в глубину заключается в следующем. Чтобы найти путь Solution из заданной вершины B в некоторую целевую вершину, необходимо:

- если B - целевая вершина, то положить Solution = [B], или
- если для исходной вершины B существует вершина - преемник B1, такая, что можно провести путь Solution1 из B1 в целевую вершину, то положить

```
Solution = [B|Solution1].
```

На Прологе это правило имеет вид:

```
solve(B,[B]) :- goal(B).
solve(B,[B|Solution1]) :- after(B,B1), solve(B1,Solution1).
```

Эта программа реализует по- иск в глубину, т.е. когда алгоритму поиска в глубину надлежит выбрать из нескольких вершин ту, в которую следует перейти для продолжения поиска, он предпочитает самую глубокую из них. Самая глубокая вершина - та, которая расположена дальше других от стартовой вершины.

Поиск в глубину наиболее адекватен рекурсивному стилю программирования, принятому в Прологе. Обрабатывая цели Пролог - система сама просматривает альтернативы именно в глубину.

Описанная процедура поиска в глубину страдает одним серьезным недостатком - она не работает в пространстве состояний, имеющем циклы.

Добавим к нашей процедуре механизм обнаружения циклов. Ни одну из вершин, уже содержащихся в пути, построенном из стартовой вершины в текущую, не следует вторично рассматривать в качестве возможной альтернативы продолжения поиска. Это правило можно сформулировать в виде отношения

in_depth(Path,Node,Solution).

Здесь:

- Node - вершина (состояние), из которой необходимо найти путь до цели;
- Path - список вершин (путь) между стартовой вершиной и Node;
- Solution - путь, продолженный до целевой вершины.

Для облегчения программирования вершины в списках, представляющих пути, будут расставляться в обратном порядке. Аргумент Path нужен для того,

(1) чтобы не рассматривать тех преемников вершины Node, которые уже встречались (обнаружение циклов);

(2) чтобы облегчить построение решающего пути Solution.

Программа поиска в глубину без зацикливания имеет вид:

```
solve(Node,Solution) :-in_depth([],Node,Solution).
in_depth(Path,Node,[Node|Path]) :- goal(Node).
in_depth(Path,Node,Solution) :- after(Node,Node1),
not(member(Node1,Path)),
                    in_depth([Node|Path],Node1,Solution).
```

Предложенная процедура, снабженная механизмом обнаружения циклов, будет успешно находить пути в конечных пространствах состояний, т. е. в пространствах с конечным числом вершин. Если же пространство состояний бесконечно, то алгоритм поиска в глубину может "потерять" цель, двигаясь вдоль бесконечной ветви графа. Чтобы предот-

вратить бесцельное блуждание, добавим в процедуру поиска в глубину ограничение на глубину поиска. Тогда эта процедура будет иметь следующие аргументы:

in_depth2(Node,Solution,MaxDepth).

Не разрешается вести поиск на глубине большей, чем MaxDepth. Программная реализация этого ограничения сводится к уменьшению на единицу величины предела глубины при каждом рекурсивном обращении к in_depth2 и к проверке, что этот предел не стал отрицательным.

В результате получаем следующую программу:

```
in_depth2(Node,[Node],_) :- goal(Node).
in_depth2(Node,[Node|Solution],MaxDepth) :-
        MaxDepth > 0, after(Node,Node1),
        Max1=MaxDepth - 1, in_depth2(Node1,Solution,Max1).
```

В отличие от предыдущей процедуры, где порядок вершин обратный, здесь порядок вершин - прямой.

Пример 2. Решение задачи манипулирования кубиками методом поиска в глубину.

```
tgoal([[a,b,c],[],[]]).    % целевые ситуации
tgoal([[],[a,b,c],[]]). tgoal([[],[],[a,b,c]]).
after(Stolbs,[Stolb1,[Up1|Stolb2]|Rest]):-
      delete([Up1|Stolb1],Stolbs,Stolbs1), delete(Stolb2,Stolbs1,Rest).
      delete(X,[X|L],L).
      delete(X,[Y|L],[Y|L1]) :- delete(X,L,L1).
      member(X,[X|_]).
      member(X,[_|Tail]):- member(X,Tail).
      solve(Node,Solution) :- in_depth([],Node,Solution).
      in_depth(Path,Node,[Node|Path]) :- tgoal(Node).
      in_depth(Path,Node,Solution) :-
            after(Node,Node1), not(member(Node1,Path)),
            in_depth([Node|Path],Node1,Solution).

  write_list([X|Rest]):-  % вывод решения на терминал
            write("\n",X),readchar(_), write_list(Rest).
      write_list([]).
```

```
?-  solve([[c,a,b],[],[]],Solution),          write_list(Solution).
```

На терминал выводится последовательность ситуаций, начиная с целевой и кончая стартовой (обратный порядок ситуаций).

Стратегия поиска в ширину

Стратегия поиска в ширину предусматривает переход в первую очередь к вершинам, ближайшим к стартовой. При поиске в ширину приходится сохранять все множество альтернативных путей-кандидатов. Таким образом, цель

in_width(Paths,Solution)

истинна только тогда, когда в множестве Paths существует такой путь,который может быть продолжен вплоть до целевой вершины.

Представим каждый путь списком вершин, перечисленных в обратном порядке, т.е. головой списка будет самая последняя из порожденных вершин, а последним элементом списка будет стартовая вершина. Поиск начинается с одноэлементного множества кандидатов

[[StartNode]].

Чтобы выполнить поиск в ширину при заданном множестве путей-кандидатов, нужно:

- если голова первого пути - это целевая вершина, то взять этот путь в качестве решения;
- иначе удалить первый путь из множества кандидатов и породить множество всех возможных продолжений этого пути на один шаг; множество продолжений добавить в конец множества кандидатов, а затем выполнить поиск в ширину с полученным новым множеством.

Пример 3. Решение задачи манипулирования кубиками методом поиска в ширину.

```
tgoal([[a,b,c],[],[]]). tgoal([[],[a,b,c],[]]). tgoal([[],[],[a,b,c]]).
after(Stolbs,[Stolb1,[Up1|Stolb2]|Rest]):-
        delete([Up1|Stolb1],Stolbs,Stolbs1), delete(Stolb2,Stolbs1,Rest).
        delete(X,[X|L],L).
        delete(X,[Y|L],[Y|L1]) :- delete(X,L,L1).
        member(X,[X|_]).
        member(X,[_|Tail]):- member(X,Tail).
        conc([],L,L).
        conc([X|L1],L2,[X|L3]) :- conc(L1,L2,L3).
solve(Start,Solution) :- in_width([[Start]],Solution).
```

```
in_width([[Node|Path]|_],[Node|          Path]) :- tgoal(Node).
in_width([[B|Path]|Paths],Solution) :- findall(S, new_node(B,S,Path),   New-
Paths),
      % NewPaths - ациклические продолжения пути [B|Path]
           conc(Paths,NewPaths,Paths1),!, in_width(Paths1,Solution);
      in_width(Paths,Solution).  % случай, когда у B нет преемника
new_node(B,S,Path) :- after(B,B1), not(member(B1,[B|Path])), S=[B1,B|
Path].
write_list([X|Rest]):-write("\n",X),readchar(_), write_list(Rest).
write_list([]).

?- solve([[c,a,b],[],[]],Solution), write_list(Solution).
```

Контрольные задания

1. Решить задачу о перевозке через реку волка, козы и капусты методом поиска в глубину. (Предполагается, что вместе с человеком в лодке помещается только один объект и что человеку приходится охранять козу от волка и капусту от козы.)
2. Решить предыдущую задачу методом поиска в ширину.
3. Решить задачу о коммивояжере методом поиска в глубину.
4. Решить головоломку "игра в восемь" методом поиска в ширину.
5. Решить задачу о восьми ферзях методом поиска в ширину. Расставить 8 ферзей на шахматной доске таким образом, что ни один из ферзей не находится под боем другого.)
6. Задача раскраски карты состоит в приписывании каждой стране на заданной карте одного из четырех заданных цветов с таким расчетом, чтобы ни одна пара соседних стран не была окрашена в одинаковый цвет. Решить задачу методом поиска в глубину.
7. Задача о ханойской башне (упрощенный вариант):

Имеется три колышка 1, 2 и 3 и три диска a,b и c (a - наименьший из них, c - наибольший). Изначально все диски находятся на колышке 1. Задача состоит в том, чтобы переложить все диски на колышек 3. На каждом шагу можно перекладывать только один диск, причем никогда нельзя помещать больший диск на меньший. Решить эту задачу методом поиска в ширину.

Библиографический список

1. *Братко И.* Программирование на языке Пролог для искусственного интеллекта. - М.: Мир, 1990.- 560 с.
2. *Ин Ц., Соломон Д.* Использование Турбо-Пролога. - М.: Мир, 1993. - 608 с.
3. *Малпас Дж.* Реляционный язык Пролог и его применение. -М.: Наука., 1990, 464 с.