Севастопольский государственный университет
Кафедра «Информационные системы»

Курс лекций по дисциплине

"МЕТОДЫ И СИСТЕМЫ ИСКУССТВЕННОГО ИНТЕЛЛЕКТА"

(МиСИИ)

Лектор: Бондарев Владимир Николаевич

Лекция18

Организация циклов. Ввод вывод.Предикаты проверки типов и декомпозиции термов. Предикаты для работы с базой данных.

В.Бондарев

Организация циклов

Рассмотрим организацию циклов в Прологе с использованием **механизма возврата.**

Пусть имеется БД, состоящая из фактов: **отец(Отец, Ребенок).**

Определим предикат **дети(Отец)**, обеспечивающий вывод на экран имен всех детей некоторого отца.

Чтобы заставить программу перебирать факты базы данных, ***организуем состояние искусственной неудачи*** (**fail**). Это приведет к тому, что пролог-система выполнит возврат к иному варианту (если такой имеется) унификации предиката **отец(Отец, Ребенок)**.

```
дети(Отец): –
        отец(Отец, Ребенок), write(Ребенок), fail;
        true.
? – дети('Иван').
```

Пусть имеется два варианта унификации предиката **отец('Иван',Ребенок)**. Тогда Пролог проверит оба варианта.

Организация циклов

Дерево вывода целевого утверждения **дети('Иван')**

```
отец('Иван','Сергей').
отец('Иван','Анна').
отец('Петр','Николай').
дети(О):-отец(О,Р),write(Р),nl,fail;true.
```

```
?-дети('Иван').
Сергей
Анна
true.
```

После вывода на экран первого значения переменной **Ребенок**, пролог-система встречает условие fail и переходит к варианту **Ребенок='Анна'**. В конце второго пути пролог-система опять встречает условие **fail** и переходит к третьему варианту, который завершается условием **true**.

Организация циклов

Схема выполнения пролог-программы, основанная на создании искусственной неудачи, называется ***циклом с возвратом.***

Рассмотренный предикат **дети(Отец)** можно **обобщить.**

Определим предикат, который выполняет **поиск всех решений** некоторого метаусловия:

```
все_решения(Условие, X): –
                    Условие, write(X), fail;
                    true.
```

Переменная **X** также является аргументом метаусловия **Условие**.

Теперь все решения, соответствующие предикату **дети(Отец)**, можно получить с помощью вызова:

все_решения(отец(Отец, X), X).

```
?-все_решения(отец('Иван',X),X).
Сергей
Анна
true.
```

```
?-все_решения(отец(O,X),X).
Сергей
Анна
Николай
true.
```

Организация циклов

С помощью рассмотренной схемы организации циклов легко организовывать вычисления, соответствующие **вложенным циклам**.

Требуется вывести на экран общие элементы списков **L1** и **L2**.

```
общие_элементы(L1, L2): –
        member(X, L1), member(X, L2),write(X), fail;
        true.
```

```
?-общие_элементы([1, 2, 3], [2, 3, 4]).
23
true.
```

В рассмотренных примерах пролог-система возвращалась к точкам выбора, порождаемым предикатами: **отец(Отец, Ребенок), member(X, L1), member(X, L2)**.

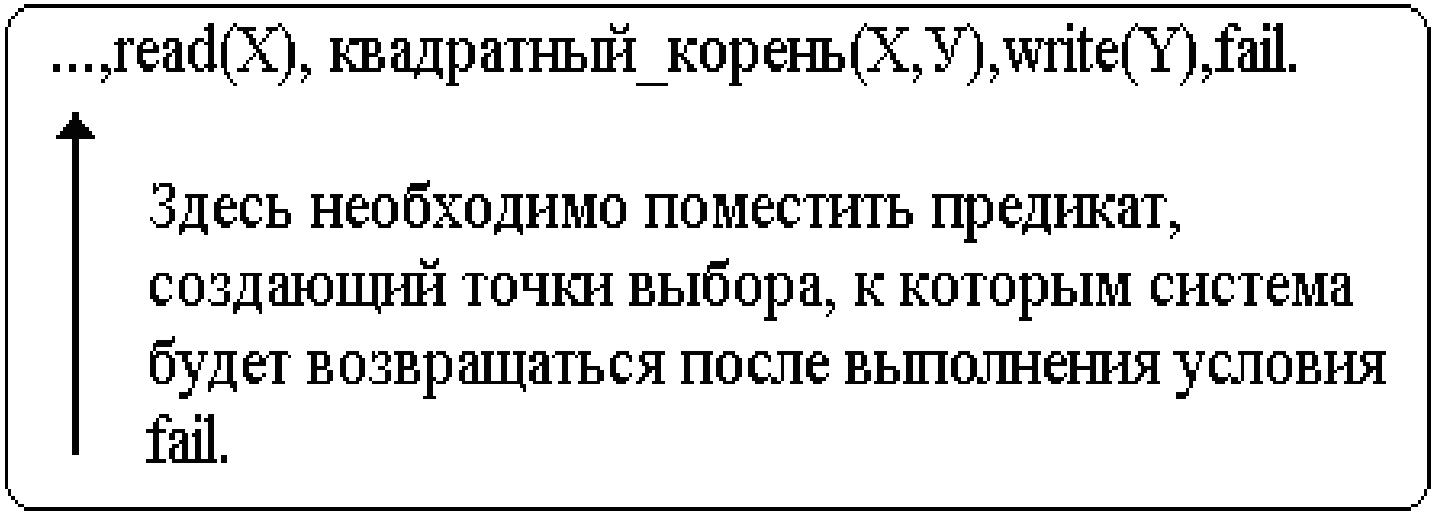
Предикаты, создающие точки выбора, называют **недетерминированными.** Иногда необходимо осуществлять повторное выполнение предикатов, не создающих точек выбора (**детерминированных**).

Для этого можно организовать цикл с возвратом, создав **искусственные точки выбора.**

Организация циклов

Пусть требуется последовательно вводить с клавиатуры числа и вычислять для каждого числа квадратный корень.

Создание искусственных точек выбора

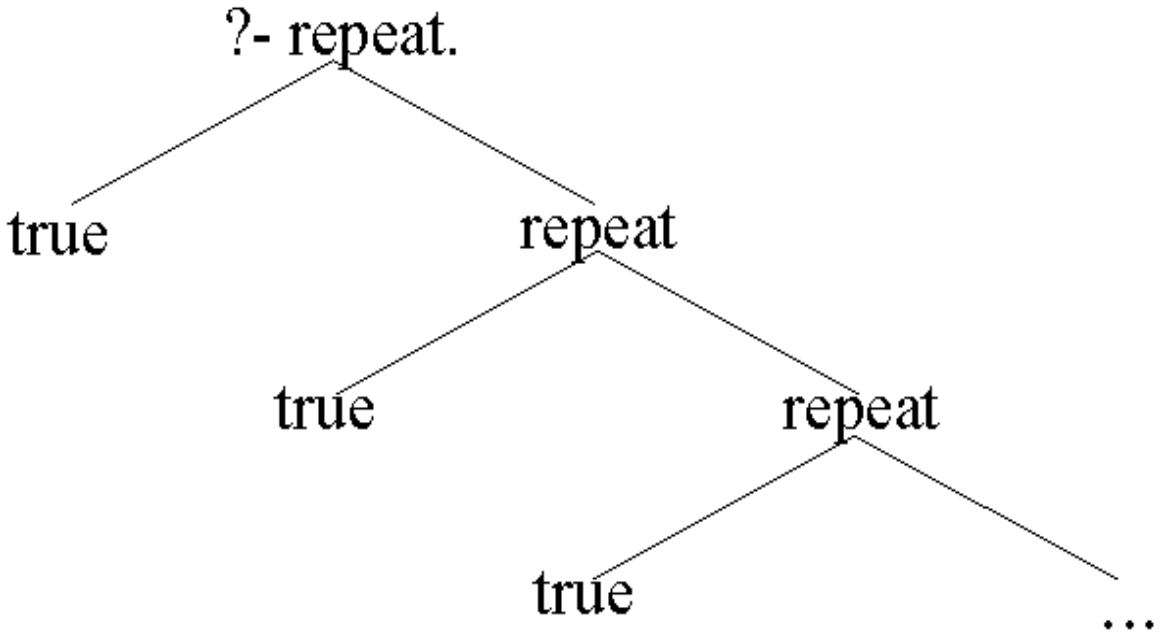


Создать точки выбора можно с помощью предиката **repeat**:

```
repeat: – true;
        repeat.
```

Организация циклов

Каждый вызов предиката **repeat** порождает новую ветвь вычислений.



Воспользуемся предикатом **repeat** и напишем циклическую программу, вычисляющую значение квадратного корня очередного числа, введенного с клавиатуры. Если вместо числа с клавиатуры будет введен атом **'стоп'**, то вычисления прекращаются:

Организация циклов

```
цикл: –
        repeat,nl,write('число?'), read(X),
        (X='стоп', !;
        квадратный_корень(X, Y),
        write('корень='), write(Y),fail).
```

Предикат **repeat** можно переопределить так, чтобы он обеспечивал построение **циклов с заданным числом повторений**.

```
repeat(1): – !.
repeat(N): – true;
                M is N-1, repeat(M).
```

Циклические вычисления можно выполнять и **рекурсивно:**

```
цикл: –
        nl, write('число?'), read(X),
        (X='стоп', !;
        квадратный_корень(X, Y),
        write('корень='),write(Y),цикл).
```

Предикаты ввода-вывода

Предикаты управления потоками

Входной поток	Выходной поток	Интерпретация предиката
see(<имя файла>)	**tell(<имя файла>)**	Определяет в качестве текущего входного или выходного потока соответствующий файл.
seeing(<имя файла>)	**telling(<имя файла>)**	Отождествляет имя файла с текущим входным или выходным потоком.
seen	**told**	Закрывает текущий входной или выходной поток и опять связывает с текущим входным или выходным потоком пользовательский терминал.

Предикаты ввода-вывода

Для считывания значений из входных потоков и записи их в выходные потоки применяют следующие предикаты:

read(X) – вызывает чтение очередного терма из входного потока и сопоставление его с **X**, вводимые термы должны заканчиваться точкой, в конце файла **X** конкретизируется атомом **end_of_file**;

write (X) – выводит терм X в текущий выходной поток;

nl – переход на новую строку;

tab(N) – выводит в выходной поток N-пробелов;

put(X) – выводит символ с ASCII кодом X на терминал;

get0(X) – считывает один символ с клавиатуры и сопоставляет его **X**;

get(X) – сопоставляет **X** с первым символом, отличным от пробела;

display(X) – выводит терм **X** в точечно-скобочной записи в текущий выходной поток. В частности: **[a,b,c]** → **.(a,.(в,.(c, [])))**

Файлы Пролога являются **текстовыми** и обрабатываются последовательно.

Предикаты ввода-вывода

Типовая последовательность вызова встроенных предикатов при выполнении ввода из файла является следующей:

```
…,
see('имя файла'),          % открытие файла
read(X),                   % чтение терма X
…,                         % обработка
seen.                      % закрытие файла
```

Аналогично строится работа с выходным потоком:

```
…,
tell('имя файла'),         % открытие файла
write(X),                  % запись терма X
…,
told.                      % закрытие файла
```

Предикаты ввода-вывода

Пример 1. Вывод списка на экран.

```
вывод _списка([ ]).
вывод _списка([H|L]):–write(H), nl, вывод _списка(L).
```

Пример 2. Обработка запроса к базе данных.

Пусть имеется база данных, содержащая факты:

```
адрес (Фамилия, Адрес).
```

Определим предикат, запрашивающий фамилию и осуществляющий вывод адреса:

```
вывод _адреса:–
        repeat, write ('Фамилия?'), nl,
        read (X),
        (X= 'стоп', ! ;
        адрес( X, Адрес), write('Адрес:'),
        write( Адрес), nl, fail).
```

Предикаты ввода-вывода

Пример 3. Определим предикат, вычисляющий значения арифметических выражений, вводимых с клавиатуры:

```
калькулятор:–
        write('Введите-выражение'), nl,
        read(X),
        (X= 'стоп', ! ;
        Y is X, write(Y), nl,
        калькулятор).
```

В этом случае в ответ на вопрос **?-калькулятор.** пролог-система предлагает пользователю ввести арифметическое выражение. После ввода выражения, например,

107+(302*3)/4.

оно будет сопоставлено переменной **X** и вычислено с помощью предиката **is**. В результате на экран будет выведено значение **333.5.**

Предикаты ввода-вывода

Пример 4. В примере реализовано копирование термов из файла F1 в файл F2. Термы в файле F1 должны завершаться точкой.

```
обработка_файлов(F1,F2):-
   see(F1),              %F1- входной поток
   tell(F2),             % F2- выходной поток
   чтение_запись,        %копирование термов до конца файла
   seen,told.            %переключение потоков на терминал

чтение_запись:-
  read(T1),                              %чтение терма
  копирование(T1,T2),                    %копирование
  write_term(T2),                        %запись терма в новой строке
  T1=\=end_of_file,чтение_запись.        %рекурсивный вызов
чтение_запись:-!.                        %завершить чтение-запись
```

Предикаты ввода-вывода

копирование(end_of_file,end_of_file):-!.
копирование(T1,T2):-T2=T1,!.

write_term(end_of_file):-!. %F2 =EOF ничего не записывать
write_term(Term):-write(Term),write('.'),nl,!.

Сначала создайте в папке проекта файл “**f1.txt**”.
Например, со следующим содержимым:

(5+3*2).
программирование_на_Прологе.
start(f1,f2).

Затем введите запрос:

обработка_файлов('d:/Eclipse_prolog/vvod_vyvod/f1.txt',
'd:/Eclipse_prolog/vvod_vyvod/f2.txt').

Просмотрите созданный файл “**f2.txt**”, предварительно обновив содержимое папки проекта в среде Eclipse: **Файл > Обновить.**
Содержимое “**f2.txt**” должно полностью совпадать с содержимым “**f1.txt**”

Предикаты проверки типов термов

var(X) – возвращает значение значение истина, если **X** – не конкретизированная переменная;
nonvar(X) – предикат **nonvar** будет истинным, если **X** – терм любого вида, кроме не конкретизированной переменной;
atom(X) – предикат atom будет истинным, если **X** – атом;
atomic(X) – предикат **atomic** даст значение истина, если **X** обозначает целое число или атом;
integer(X) – предикат **integer** будет истинным, если **X** обозначает целое число.

? – var(X).
Yes
? – X=пролог, var(X).
No
? – X= пролог, integer(X).
No

? – atom(пролог).
Yes
? – atom(220).
No
? – atomic(220).
Yes

Предикаты декомпозиции термов

name вернёт значение истина, если **L** – список ASCII-кодов символов, образующих атом **A**, например:

? – name (X, [97,105]).
X = ai
? – name (ab,L).
L = [97,98]

Предикат **functor(T,F,A)** будет истинным, если главный функтор **F** есть терм **T** с арностью **A**, например:

? – functor(f(a, x), F, A).
F = f, A = 2

Предикат **T = ..L** будет истинным, если **L**– список, где первый элемент - главный функтор терма **T**, за которым следуют аргументы терма **T**, например:

? – f(a, b) = ..L.
L = [f, a, b]

? – T = ..[быстро, автомобиль].
T = быстро(автомобиль).

Предикаты работы с базой данных

В Прологе имеется ряд встроенных предикатов, которые позволяют корректировать базу данных в процессе выполнения программ. К ним относят следующие предикаты: **assert(X), asserta(X), assertz(X), retract(X)**.

Предикат **assert(X)** добавляет в базу данных утверждение **X**.

? – assert(столица('Украина', 'Киев')).
Yes
? – столица('Украина', X).
X = Киев

Предикат **asserta(X)** добавляет утверждение **X** в базу данных и помещает его перед другими утверждениями, соответствующими этому же факту или правилу. Предикат **assertz(X)** определяется аналогично **asserta(X)**, но утверждения помещаются после других утверждений. Предикат **retract(X)** выполняет поиск утверждения **X** в базе данных и удаляет первое найденное утверждение.

Предикаты работы с базой данных

Если предикаты группы **assert** используются для **добавления в базу данных правила**, то добавляемое правило дополнительно заключается в скобки:

? – assert((a(X): – b(X), c(X))).

Предикат **assert** позволяет **накапливать в базе данных** уже вычисленные ответы на вопросы. Например, пусть в программе используется предикат **решить(Задача,Решение).** Тогда можно задать вопрос и добавить полученное решение в базу данных:

? – решить(Задача1, Решение),
asserta(решить(Задача1, Решение)).

Если теперь задавать вопросы, использующие добавленный факт **решить(Задача1, Решение)**, то ответ будет получен быстрее, так как никаких вычислений выполнять уже не требуется.

Предикаты работы с базой данных

Пользователь может **вводить новые утверждения в базу данных** из файла, используя предикат **consult(Файл)**. Например, запрос

? – consult('prog1.pl').

считывает содержимое файла "**prog1.pl**". Если при чтении утверждений из файла обнаруживается синтаксическая ошибка, то выдается соответствующее сообщение, но обработка оставшихся утверждений продолжается.

Утверждения, считываемые предикатом **consult(Файл)** добавляются в конец существующей базы данных. Допускается **замена существующих утверждений базы данных новыми утверждениями** из файла. В этом случае используется предикат **reconsult(Файл)**.

Можно запомнить текущее состояние системы в файле. Для этого применяется предикат **save(Файл)**. Восстановить запомненное состояние системы можно, воспользовавшись предикатом **restore(Файл)**.