Тема 4. Управление перебором, ввод-вывод и работа с файлами

Несколько полезных встроенных предикатов.

repeat – предикат, который всегда верен. Он всегда доказывается, независимо от направления доказательства. Так как доказательство производится слева направо, то движение слева направо обозначим как успешное, обратное, как откат:

1. → repeat → (доказывается успешно)

1. → repeat → ← (попытка отката до repeat)
2. → repeat → ← → (доказывается успешно)

Предикат repeat может быть записан на Прологе в виде следующей программы:

```
repeat.
repeat :- repeat.
```

Что произойдет, если строчки поменять местами?

fail – предикат, который всегда НЕверен. Он НИКОГДА не доказывается.

1. → fail (не доказывается)
2. ← fail (не доказывается)

Следствие: всё, что будет перечислено через запятую после fail никогда не будет доказываться (мы туда просто не попадём).

Предикат fail может быть записан на Прологе в виде следующей программы:

```
fail :- 1 == 0.
```

Из repeat и fail может быть получен бесконечный цикл:

```
?- ..., repeat, ... , fail.
```

Из этого цикла программа выйдет по Out of memory, т.к. при доказательстве repeat генерируется новая ветвь доказательства, которая размещается в памяти.

Рассмотрим программу определения минимума из двух чисел:

```
1. min(X, Y, X) :- X < Y.
2. min(X, Y, Y).
```

Если X меньше Y, значит минимальный – X, иначе – минимальный – Y. Проверяем программу:

```
?- min(2, 3, Min).
Min = 2
yes
```

В чем ошибка в программе?

```
?- min(2, 3, Min), Min > 2.
Min = 3
yes
```

Т.е. минимальное число равно 3? Правильное решение:

```
1. min(X, Y, X) :- X < Y.
2. min(X, Y, Y) :- X >= Y.
```

Второй вариант правильного решения с «отсечением»:

```
1. min(X, Y, X) :- X < Y, !.
2. min(X, Y, Y).
```

Оператор отсечения в Прологе обозначается восклицательным знаком – «!». Отсечение всегда доказывается при прямом доказательстве, а при попытке отката запрещает передоказательство (поиск альтернатив) того правила, в котором указано отсечение.

Как можно задать конструкцию **if-then-else** на Прологе:

```
A :- B, !, C.
A :- D.
```

Если удалось доказать B, то доказывается C, иначе доказывается D. Если убрать отсечение, то при неуспешном доказательстве C может произойти попытка передоказательства B, а это уже будет не if-then-else.

Предикат member(Item, List) давал возможность находить несколько решений по вхождению элемента в список. Можно его исправить, чтобы он искал только первое вхождение:

```
member(Elem, [Elem|_]) :- !.
member(Elem, [Head|Tail]) :- member(Elem, Tail).
```

Выделяют два вида отсечений: красные и зеленые. Деление условное.

Зеленые отсечения не влияют на логику выполнения программы, а только отсекают ветви перебора, в рамках которых решения быть не может. Красные – влияют и могут отсекать решения.

С использованием отсечения может быть описан предикат отрицания **not**.

```
not(X) :- X, !, fail.
not(_).
```

Что он делает: если X удаётся доказать, то not – не верен, т.к. сочетание отсечения и fail приведет к неуспешности доказательства not. Если X не удаётся доказать, то происходит поиск альтернативы первому правилу, которая оказывается верна, независимо от того, какой у нас X.

Пример:

```
?- X = 2, not(X == 3).
X = 2
yes
?- X = 2, not(X == 2).
no
```

Программа вычитания одного списка из другого **minus(L1, L2, Diff)**. Т.е. в Diff находятся только те элементы, которые встречаются в L1 и не встречаются в L2.

```
minus([], _, []).
minus([X | L1], L2, L) :- member(X, L2), !, minus(L1, L2, L).
minus([X | L1], L2, [X | L]) :- minus(L1, L2, L).
```

Если исходный список пуст, то и результат вычитания – пуст. Если «голова» исходного (уменьшаемого) списка есть в вычитаемом списке, то про «голову» забываем и осуществляем вычитание из «хвоста». Если «головы» нет в вычитаемом списке, то при возврате нужно «голову» поместить в результирующий список.

Пример:

```
?- minus([a, b], [b], L).
L = [a]
yes
```

Краткая справка для выполнения практических и лабораторных работ:
read(X) – предикат для чтения X из текущего входного потока.
write(X) – предикат вывода X в текущий выходной поток.
nl – выводит в выходной поток знак перевода каретки.

Пример использования:

```
?- write('Enter value: '), read(X), write('result='),
write(X).
Enter value: 12345.
result=12345
X = 12345
yes
```

Обратите внимание, что после ввода 12345 введена **точка**!!! после которой выполнен **перевод каретки** (нажата клавиша Enter).

Предикаты для работы с файлами:
see(ИмяФайла) – перенаправляет входной поток на чтение данных из файла.
seen – закрывает входной поток, если читали из файла.
see(user) - перенаправляет входной поток на чтение данных из стандартной консоли.
tell(ИмяФайла) – перенаправляет выходной поток на вывод данных в файл.
told - закрывает выходной поток, если выводили в файл.
tell(user) - перенаправляет выходной поток вывод данных в стандартную консоль.

Файлы могут обрабатываться только последовательно. Каждый запрос на чтение из входного файла приводит к чтению в текущей позиции текущего входного потока. После этого чтения текущая позиция, будет перемещена на следующий, еще не прочитанный элемент данных. Следующий запрос на чтение приведет к считыванию, начиная с этой новой текущей позиции. Если запрос на чтение делается в конце файла, то в качестве ответа на такой запрос выдается атом **end_of_file** (конец файла). **Считанную один раз информацию считать вторично невозможно.**

В файле data.txt находится 12345 *без точки и перевода каретки*. Результат запроса:

```
?- see('data.txt'), write('Enter value: '), read(X),
write('result='), write(X), seen.
Enter value:
uncaught exception: error(syntax_error('data.txt:1 (char:6) .
or operator expected after expression'),read/1)
```

В файле *data.txt* находится 12345. *с точкой без перевода каретки*. Результат запроса:

```
?- see('data.txt'), write('Enter value: '), read(X),
write('result='), write(X), seen.
Enter value: result=end_of_file
X = end_of_file
yes
```

В файле *data.txt* находится 12345. *с точкой и переводом каретки*. Результат запроса:

```
?- see('data.txt'), write('Enter value: '), read(X),
write('result='), write(X), seen.
Enter value: result=12345
X = 12345
yes
```

В файле данные должны быть с маленькой буквы или в одинарных кавычках!!!