Функциональное и логическое программирование

Лекция 5

2.3 Графический отладчик

Включение графической трассировки: guitracer.

или через меню Тест - GUI-Tracer.

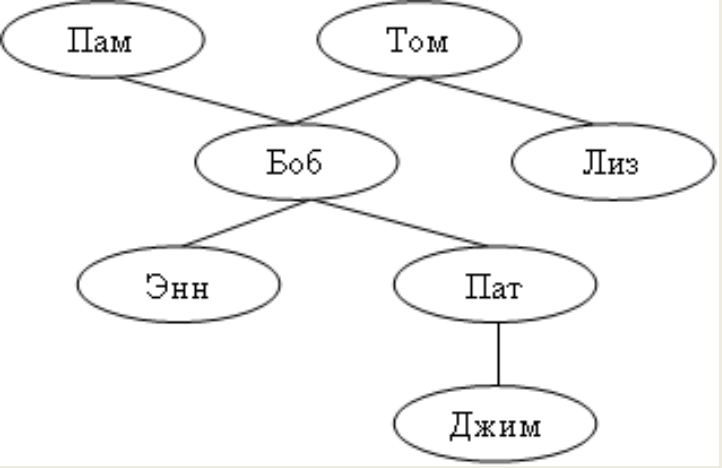
Выключение графической трассировки: noguitracer.

Или можно использовать пиктограмму  на панели инструментов в SWI-Prolog_Edit.

После включения графической трассировки включаем трассировку:

?-trace, <имя предиката>.

Откроется дополнительное окно графического отладчика. Оно разделено на 3 части: верхнее левое окно показывает текущие значения переменных, верхнее правое окно показывает дерево вызовов, а нижнее – текст программы. Для пошагового выполнения следует нажимать значок стрелки, направленной вправо или пробел.



Пример:

?-trace, дед(том,X).

c:/users/марина/onedrive/рабочий стол/файл1.pl

Tool Edit View Compile Help

Bindings

X = том

Call Stack

11 дед/2

```
родитель(пам,боб).
родитель(том,боб).
родитель(том,лиз).
родитель(боб,энн).
родитель(боб,пат).
родитель(пат,джим).
мужчина(том).
мужчина(боб).
мужчина(джим).
дед(X,Y):-мужчина(X),родитель(X,Z),родитель(Z,Y).
```

Call: дед/2

2.4 Некоторые операции в SWI-Prolog

=	Унификация (присваивание значения несвязанной переменной)
<, =<, >=, >	Арифметические (только для чисел) операции сравнения
=:=	Арифметическое равенство
=\=	Арифметическое неравенство
is	Вычисление арифметического выражения
@<, @=<, @>=, @>	Операции сравнения для констант и переменных любого типа (чисел, строк, списков и т.д.)
==	Равенство констант и переменных любого типа
\==	Неравенство констант и переменных любого типа

Примеры:

?- X=3+2.

X = 3+2.

?- X is 3+2.

X = 5.

?- 3+2=2+3.

false.

?- 3+2==2+3.

false.

?- 3+2=:=2+3.

true.

2.5 Предикаты ввода-вывода

read(A)	Чтение значения с клавиатуры в переменную A
write(A)	Вывод значения A на экран без перевода строки
writeln(A)	Вывод значения A на экран с переводом курсора в начало следующей строки
nl	Перевод курсора в начало следующей строки
format(‘<строка~w>’,X)	<строка> <значение X>
format(‘<строка~w~w>’, [X,Y])	<строка> <значение X> <значениеY>
format(‘<строка1~w\n строка2~w>’,[X,Y])	<строка1> <значение X> <строка2> <значениеY>

Некоторые полезные сведения при работе с интерпретатором SWI-Prolog

При ожидании ввода выводится приглашение |:

Комментарии заключаются между /* и */ или % комментируется строка (есть иконка на панели инструментов в SWI-Prolog-Edit).

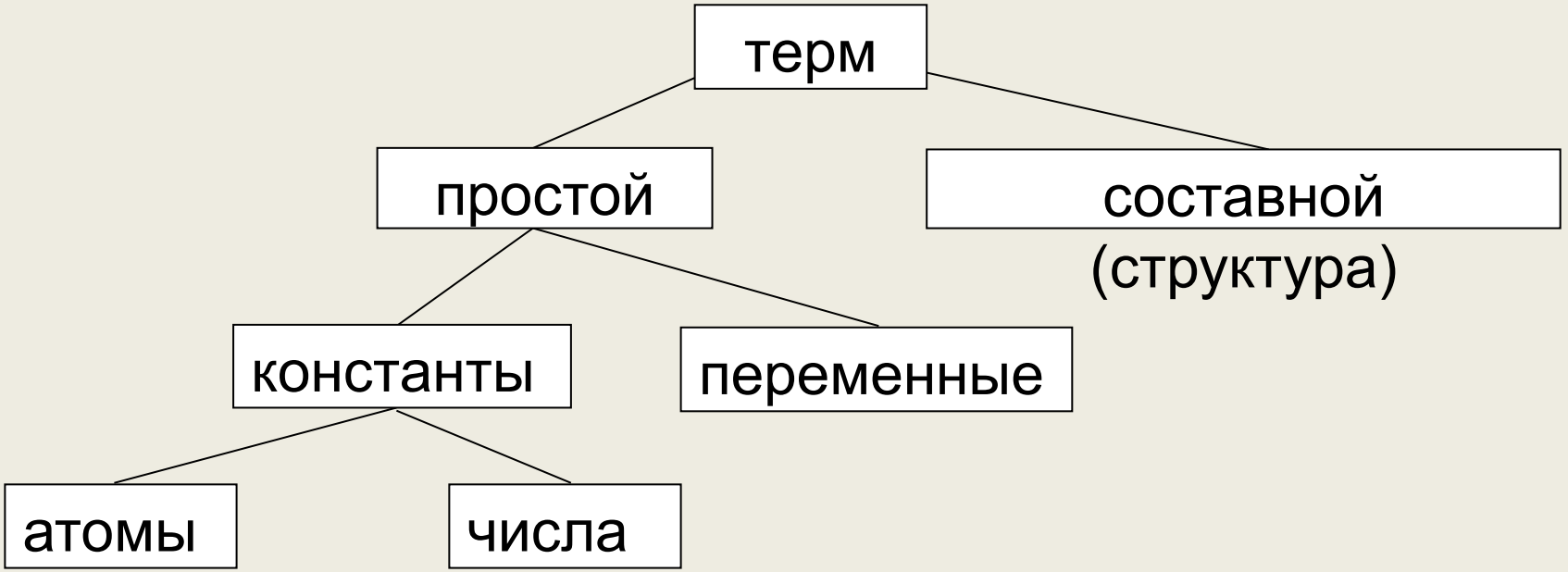
pwd. – текущая папка;

ls. – содержимое текущей папки;

cd(‘<путь к папке, слэши дублируются’>). – сменить текущую рабочую папку.

2.6 Структуры

В соответствии с теорией предикатов первого порядка единственная структура данных в логических программах - *термы*.



Атом - последовательность латинских букв, цифр, начинающаяся со строчной буквы, любая последовательность символов, заключенных в апострофы, или специальный символ.

Составной терм:

$$f(t_1,t_2,...,t_n),$$

где f - имя n-арного функтора, t_i – аргументы-термы.

Единообразие программ и данных также, как в Лиспе.

Примеры составных термов:

дата(30, ноябрь,2021).

дата(ноябрь,2021).

Каждый функтор и предикат определяется двумя параметрами: именем и арностью (количеством аргументов).

В Пролог-программах допускается использование одного и того же предикатного (или функционального символа) с разным числом аргументов.

Арифметические выражения – составные термы, записанные в инфиксной форме. Но их можно также записать в префиксной форме.

Пример:

?- X is 5*4+8/2.

X = 24.

?- X is +(*(5,4),/(8,2)).

X = 24.

Предикаты для проверки типа терма:

var(Term) - свободная переменная

nonvar(Term) - несвободная (конкретизированная) переменная

number(Term) - целое или вещественное число

atom(Term) – атом

atomic(Term) - атом или число

ground(Term) -терм не содержит свободных переменных

2.7 Рекурсия

Рекурсия в Прологе может быть алгоритмическая и по данным. Основное отличие от Лиспа заключается в том, что возвращаемое значение должно находиться в аргументе предиката.

Передача параметров осуществляется по значению.

Пример 1:

Определим одноместный предикат, который заданное количество раз выводит на экран строку **********.

```
w(N):-N>0,write('*********’),nl,N1 is N-1,w(N1).
```

Результат работы:

```
?- w(2).
*********
*********
*********
false.
```

Предикат завершился неуспешно. Так быть не должно. Причина в том, что при N=0 нет успешного сопоставления текущей цели w(0) (можно увидеть, включив трассировку).

Поправим код:

```
w(0).
w(N):-N>0,write('**********'),nl,N1 is N-1,w(N1).
```

Результат работы:

```
?- w(2).
**********
**********
**********
true .
```

Пример 2:

Определим одноместный предикат, вычисляющий n!.

Ввод n с клавиатуры и вывод результата будет осуществлять предикат goal.

```
goal:-writeln('N=?'),read(N),fact(N,P),format('~w!=~w',[N,P]).
fact(0,1).
fact(N,P):-N1 is N-1,fact(N1,P1),P is N*P1.
```

Результат работы:

```
?- goal.
N=?
|: 5.
5!=120
true .
```

2.7 Семантика Пролога

2.7.1 Порядок предложений и целей

Программу на Прологе можно понимать по-разному: с декларативной и процедурной точки зрения.

Декларативная семантика касается только отношений, описанных в программе, и определяет, что является ли поставленная цель достижимой и если да, то определяются значения переменных, при которых эта цель достижима.

Процедурная семантика определяет, как должен быть получен результат, т.е. как Пролог-система отвечает на вопросы.

Для правила вида P:–Q,R. декларативная семантика определяет, что из истинности Q и R следует истинность P, а процедурная семантика определяет, что для решения P следует сначала решить Q, а потом R (важен порядок обработки целей).

P:-P. Верное по декларативной семантике, но бесконечный цикл – по процедурной семантике.

Пример 1:

Изменим порядок следования правил и подцелей в предикате предок, определенном на прошлой лекции:

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

предок1(X,Y):-родитель(X,Z),предок1(Z,Y).
предок1(X,Y):-родитель(X,Y).

предок2(X,Y):-родитель(X,Y).
предок2(X,Y):-предок2(Z,Y),родитель(X,Z).

предок3(X,Y):-предок3(Z,Y),родитель(X,Z).
предок3(X,Y):-родитель(X,Y).
```

Посмотрим, как Пролог будет искать решения при ответе на следующие вопросы:
?-предок(том,боб).
true .
Ответ найден быстро (смотрим трассировку).

?-предок1(том,боб).
true .
Ответ долго ищется, сначала перебираются все предки Боба (смотрим по трассировке).

?-предок2(лиз,боб).
Переполнение стека, бесконечная рекурсия, хотя делаются попытки найти решение (смотрим по трассировке).

?-предок3(том,боб).
Переполнение стека, бесконечная рекурсия сразу (смотрим по трассировке).

Таким образом, правильные с декларативной точки зрения программы могут работать неправильно. При составлении правил следует руководствоваться следующим:

- более простое правило следует ставить на первое место;
- по возможности избегать левой рекурсии.

2.7.2 Пример декларативного создания программы

Пример 2 (задача об обезьяне и банане):

Возле двери комнаты стоит обезьяна. В середине комнаты к потолку подвешен банан. Обезьяна голодна и хочет съесть банан, но не может до него дотянуться, находясь на полу. Около окна этой комнаты находиться ящик, которым обезьяна может воспользоваться.

Обезьяна может предпринимать следующие действия: ходить по полу, залазить на ящик, двигать ящик (если обезьяна находится возле ящика), схватить банан (если обезьяна находится на ящике под бананом).

Сможет ли обезьяна добраться до банана?

Мир обезьяны всегда находится в некотором состоянии, которое может изменяться со временем.

Состояние обезьяньего мира определяется четырьмя компонентами: горизонтальная позиция обезьяны, вертикальная позиция обезьяны, позиция ящика, наличие у обезьяны банана. Объединим эти компоненты в структуру функтором состояние.

Например, может быть такое состояние:

состояние(уокна, наполу, вцентре, нет).

Задачу можно рассматривать как игру для одного игрока – обезьяны. Формализуем правила этой игры: начальное состояние, цель игры и ходы игры.

Начальное состояние игры:

состояние(удвери, наполу, уокна, нет).

Цель игры: состояние(_,_,_, да).

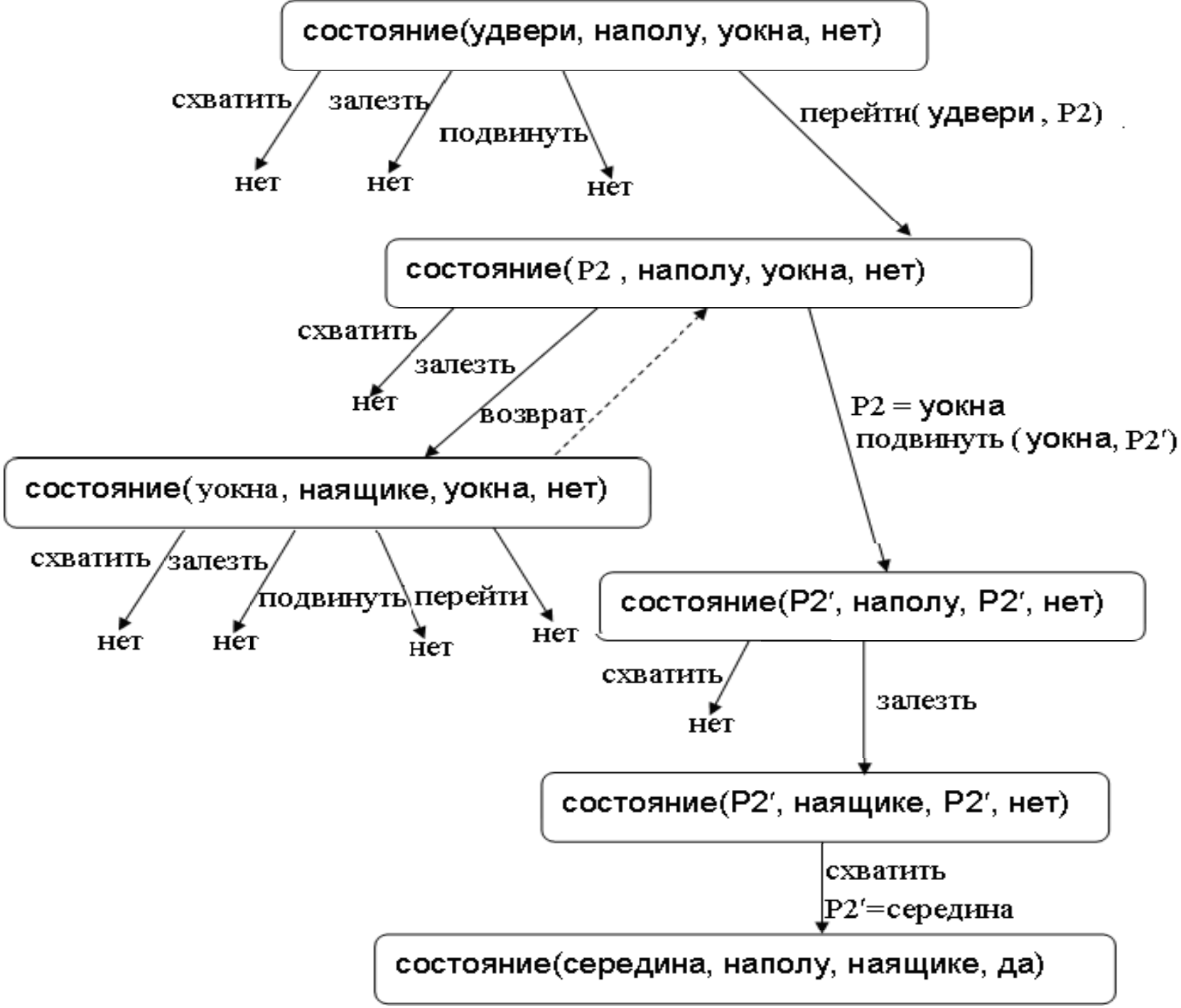
Ходы обезьяны: перейти в другое место, подвинуть ящик, залезть на ящик, схватить банан.

В результате хода меняется состояние обезьяньего мира, система переходит из одного состояния в другое. Для изменений определим предикат:

ход(состояние, действие, состояние).

```
goal:- может_достать(состояние(удвери,наполу,уокна,нет)).

ход(состояние(всередине,наящике,всередине,нет),
      схватить,
      состояние(всередине,наящике,всередине,да)).
ход(состояние(P,наполу,P,H),
      залезть,
      состояние(P,наящике,P,H)).
ход(состояние(P1,наполу,P1,H),
      подвинуть(P1,P2),
      состояние(P2,наполу,P2,H)).
ход(состояние(P1,наполу,P,H),
      перейти(P1,P2),
      состояние(P2,наполу,P,H)).
может_достать(состояние(_,_,_,да)).
может_достать(S1):-ход(S1,_,S2),
                              может_достать(S2).
```


состояние(удвери, наполу, уокна, нет)
схватить
залезть
подвинуть
перейти(удвери , P2)
нет
нет
нет
состояние(P2 , наполу, уокна, нет)
схватить
залезть
возврат
нет
P2 = уокна
подвинуть (уокна, P2′)
состояние(уокна , наящике, уокна, нет)
схватить
залезть
подвинуть
перейти
нет
нет
нет
нет
состояние(P2′, наполу, P2′, нет)
схватить
нет
залезть
состояние(P2′, наящике, P2′, нет)
схватить
P2′=середина
состояние(середина, наполу, наящике, да)

Для того чтобы ответить на вопрос, Пролог-системе пришлось сделать лишь один возврат. Причина такой эффективности – правильно выбранный порядок следования предложений, описывающих ходы.

Однако возможен и другой порядок, когда обезьяна будет ходить туда-сюда, не касаясь ящика, или бесцельно двигать ящик в разные стороны. Порядок предложений и целей важен в программе. Например, если первым поставить ход – перемещение обезьяны, то она будет ходить туда-сюда.