

1. ЗНАКОМСТВО С SWI/PROLOG. ЗАПУСК ПРОСТОЙ ПРОГРАММЫ

ЦЕЛЬ: Знакомство с интерпретатором SWI/PROLOG, включая использование меню, создание программных файлов, запуск и трассировку программ на SWI/PROLOG.

1.1. ГЛАВНОЕ МЕНЮ

В папке с установленным SWI/PROLOG войдите в директорию *pl/bin*, содержащую файл *plwin.exe*, и запустите его. На экране появится главное меню и главное (диалоговое) окно с приглашением SWI/PROLOG (см рис.1).

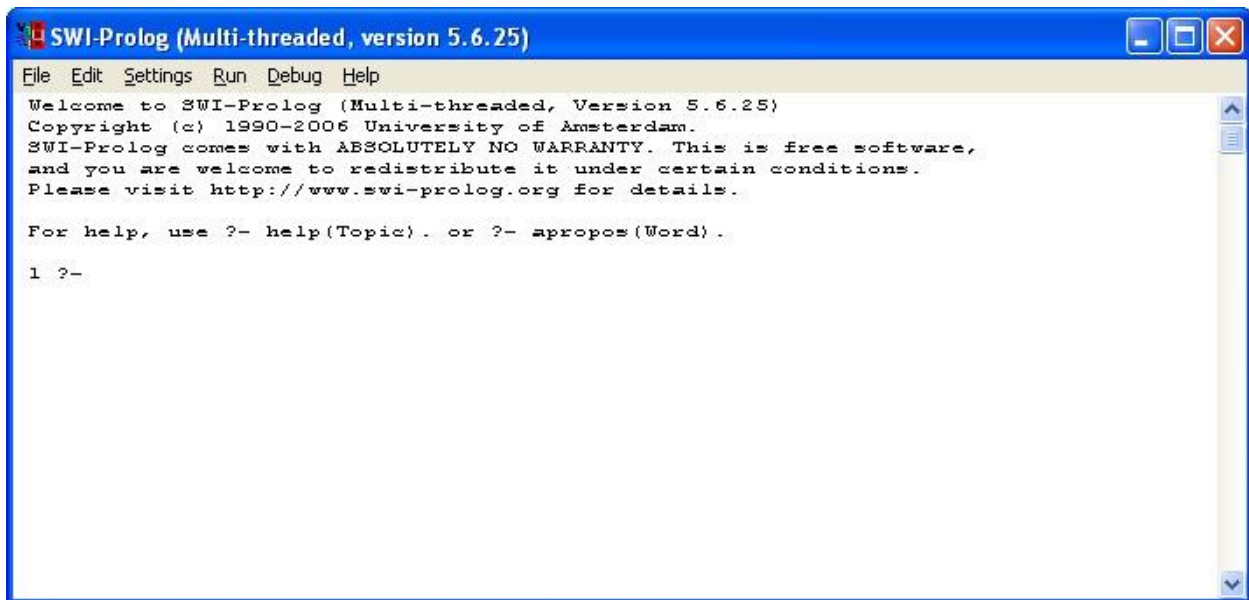


Рис.1 Вид диалогового окна SWI/PROLOG

Главное меню можно сделать активным, нажав **F10** или **Alt**. Когда главное меню активно, его элементы можно выбрать с помощью клавиш управления курсором (**→**, **←**) и последующим нажатием клавиши **Enter**. Выбирать элементы главного меню можно также и мышью.

1.2. ПЕРВАЯ ПРОГРАММА

Программа

на Прологе состоит из фактов и правил, которые образуют базу знаний Пролог-программы, и запроса к этой базе, который задает цель поиска решений.

Предикат

описывают отношение между объектами, которые являются аргументами предиката.

Факты

Констатируют наличие заданного предикатом отношения между указанными объектами.

ПРИМЕР

Констатация факта в предложении
Эллен любит теннис.

в синтаксисе Пролога выглядит так:



Имя предиката (функтора) и объекта должно начинаться с маленькой буквы и может содержать латинские буквы, кириллицу, цифры и символ подчеркивания (_). Кириллица используется наравне с латинскими буквами. Обычно предикатам дают такие имена, чтобы они отражали смысл отношения. Например: **main**, **add_file_name**. Два предиката могут иметь одинаковые имена, тогда система распознает их как разные предикаты, если они имеют различное число аргументов (арность). Например, **любит/2**, **любит/3**.

Имя предиката может совпадать с именем какого-либо встроенного предиката SWI/PROLOG-а. Однако, если совпали имена пользовательского и встроенного предиката, то при обращении к нему (либо из интерпретатора, либо из программы), будет вызван пользовательский предикат, т.е. пользовательское определение «перекроет» предопределенное в SWI/PROLOG-е.

Правила

описывают связи между предикатами.

ПРИМЕР

Билл любит все, что любит Том.

в синтаксисе Пролога

любит('Билл',Нечто):- любит('Том',Нечто).

Правило **B:-A** соответствует импликации $A \rightarrow B$ («ЕСЛИ A , ТО B»).

В общем виде правило - это конструкция вида:

$P_0:-P_1,P_2,\dots,P_n$

которая читается « P_0 истинно, если P_1 и P_2 и ... P_n истинны».

Предикат P_0 называется заголовком правила, выражение P_1,P_2,\dots,P_n - телом правила, а предикаты P_i - подцелями правила. **Запятая** означает логическое "И".

Факты и правила называются также утверждениями или клозами. Факт можно рассматривать как правило, имеющее заголовок и пустое тело.

Процедура

- это совокупность утверждений, заголовки которых имеют одинаковый функтор и одну и ту же арность. Процедура задает определение предиката.

Конец предложения всегда отмечается точкой, поэтому все факты, правила и запросы должны заканчиваться точкой. Заметим также, что между именем предиката и скобкой не должно быть пробелов.

Переменная

- *поименованная область памяти, где может храниться значение.*

Если переменная не связана со значением – она называется **свободной переменной**.

Унификация

- процесс получения свободной переменной значения в результате сопоставления при логическом выводе в SWI/PROLOG-е.

Понятие переменной в логическом программировании отличается от базового понятия переменной, которое вводится в структурном программировании. Прежде всего, это отличие заключается в том, что переменная в SWI/PROLOG-е, однажды получив свое значение при унификации в процессе работы программы, не может его изменить, т.е. она скорее является аналогом математического понятия «переменная» – неизвестная величина. Переменная в SWI/PROLOG не имеет предопределенного типа данных и может быть связана с значением любого типа данных.

Переменная в SWI/PROLOG обозначается как последовательность латинских букв, кириллицы и цифр, начинающаяся с заглавной буквы или символа подчеркивания (_). Заметим, что если значение аргумента предиката или его имя начинается с заглавной буквы, то оно пишется в апострофах (см. предыдущий пример).

В SWI/PROLOG различаются строчные и заглавные буквы.

Рассмотрим следующую программу на SWI/PROLOG, которую будем использовать для иллюстрации процессов создания, выполнения и редактирования Пролог-программ.

ПРОГРАММА 1. /* кто что любит */

любит('Эллен',теннис). %Эллен любит теннис

любит('Джон',футбол). %Джон любит футбол

любит('Том',бейсбол). %Том любит бейсбол

любит('Эрик',плавание). %Эрик любит плавание

любит('Марк',теннис). %Марк любит теннис

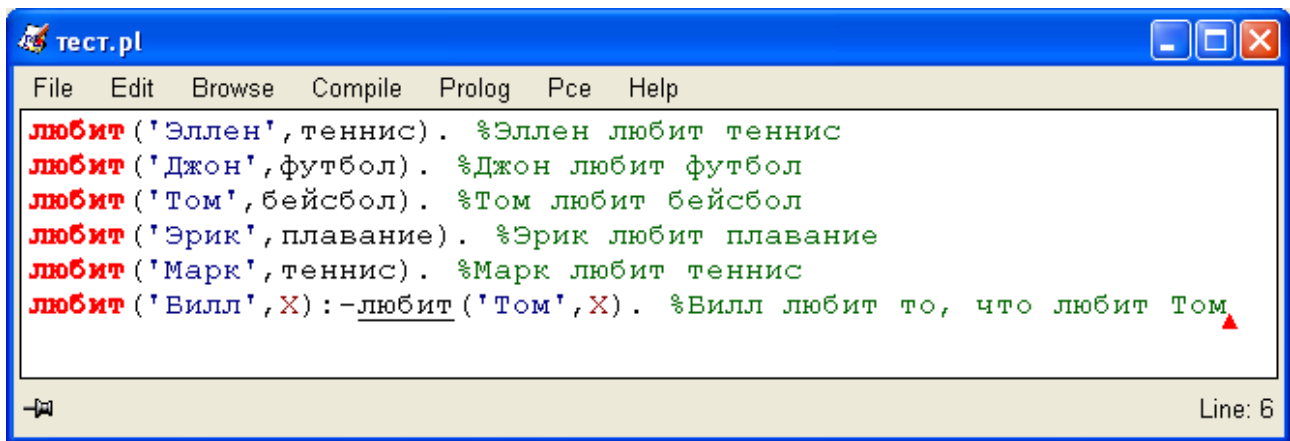
любит('Билл',X):-любит('Том',X). %Билл любит то, что любит Том

Комментарий в строке программы начинается с символа % и заканчивается концом строки. Блок комментариев выделяется специальными скобками: /* (начало) и */ (конец).

ЗАДАНИЕ 1.1

Для того чтобы набрать текст программы воспользуйтесь встроенным текстовым редактором. Чтобы создать новый файл выберете команду **File/New**, в диалоговом окне укажите имя нового файла, например, **тест**. Для редактирования уже созданного файла с использованием встроенного редактора можно воспользоваться командой меню **File/Edit**. Набейте программу 1 (текст программы выше по тексту) и сохраните ее (**File/Save buffer**).

Внешний вид редактора



Красным цветом подсвечиваются предикаты в заголовках предложений, которые с точки зрения синтаксиса SWI/PROLOGa корректны. Указатель “курсор” можно использовать для выверки (например, корректности) расстановки скобок. Зелёным цветом выделяются комментарии, темно-красным цветом – переменные. Подчеркиванием выделяются предикаты в теле правила, которые совпадают с предикатом заголовка, – таким образом акцентируется внимание на возможном заиклиивании программы.

Чтобы запустить программу, сначала необходимо ее загрузить в SWI/PROLOG для выполнения. Это делается выбором опции **Compile/Compile buffer** из окна редактора. Результат компиляции отображается в окне интерпретатора SWI/PROLOGa. Там же указываются ошибки, возникшие при компиляции, чаще всего они отображаются и во всплывающем окне ошибок. Обычно перед компиляцией предлагается сохранить файл.

Другой способ загрузить уже существующий файл – это выполнение команды **Consult** в подменю **File** диалогового окна SWI/PROLOG. На экране появится диалоговое окно.



Укажите имя файла, который вы хотите загрузить, и выберите **Открыть**. Если вы попытаетесь загрузить для выполнения файл, в котором есть синтаксические ошибки, то он не загрузится, а вы получите сообщение об ошибке в главном окне. Угловые скобки << >> будут выделять место, где встретила

ошибка. По умолчанию файлы, ассоциируемые с SWI/PROLOG имеют расширение **.pl**.

Файлы также можно загрузить, используя встроенный предикат:

consult(Имя файла или имена нескольких файлов).

ПРИМЕРЫ

consult(Test). % test – имя файла
consult([Test1,Test2]). % Загрузка двух файлов.
consult('test.pl').¹

Для выполнения загрузки этот предикат нужно написать в главном окне после приглашения интерпретатора (?-), которое означает, что интерпретатор ждет запрос.

Запрос

- это конструкция вида:

?- P₁,P₂,...,P_n.

которая читается "Верно ли P₁ и P₂ и ... P_n ?". Предикаты P_i называются подцелями запроса.

Запрос является способом запуска механизма логического вывода, т.е фактически запускает Пролог-программу.

Для просмотра предложений загруженной базы знаний можно использовать встроенный предикат **listing**.

Проверьте загрузку исходного файла, задайте запрос

?-listing.

Введите запрос:

?-любит('Билл',бейсбол). % Любит ли Билл бейсбол?

Получите ответ **yes** (да) и новое приглашение к запросу.

Введите следующие запросы и посмотрите на результаты.

?-любит('Билл', теннис). %Любит ли Билл теннис?

?-любит(Кто, теннис). %Кто любит теннис?

?-любит('Марк',Что),любит('Эллен',Что).%Что любят Марк и Эллен?

?-любит(Кто, Что). %Кто что любит?

?-любит(Кто, _). %Кто любит?

При поиске решений в базе Пролога выдается первое решение.

ПРИМЕР

?-любит(Кто,теннис).

Кто = 'Эллен'

Если необходимо продолжить поиск в базе по этому же запросу и получить альтернативные решения, то вводится точка с запятой **;**.

Если необходимо прервать выполнение запроса, (например, нужно набрать другой запрос), используйте клавишу **b**.

¹ Не вводите имя файла с расширением без апострофов.

Если Вы хотите повторить один из предыдущих запросов, воспользуйтесь клавишами "стрелка вверх" или "стрелка вниз".

Перезагрузить, измененные во внешнем редакторе, файлы можно, используя встроенный предикат **make**. Например так:

?-make.

```

SWI-Prolog (Multi-threaded, version 5.6.25)
File Edit Settings Run Debug Help
1 ?- Warning: (f:/логическое программирование/swi/методичка swiprolog+примеры программы/тест.pl:2):
Singleton variables: [Нечто]
% f:/логическое программирование/SWI/Методичка SWIprolog+примеры программы/тест.pl compiled 0.00 sec, 1,704 bytes
% f:/логическое программирование/SWI/Методичка SWIprolog+примеры программы/a.pl.txt compiled 0.00 sec, 1,324 bytes
1 ?- % SWI-Prolog version 5.6.25 by Jan Wielemaker (jan@swi-prolog.org)
% Copyright (c) 1990-2006 University of Amsterdam.
% SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software,
% and you are welcome to redistribute it under certain conditions.
% Please visit http://www.swi-prolog.org for details.
1 ?- make.
Warning: (f:/логическое программирование/swi/методичка swiprolog+примеры программы/тест.pl:2):
Singleton variables: [Нечто]
% f:/логическое программирование/swi/методичка swiprolog+примеры программы/тест.pl compiled 0.02 sec, 196 bytes
% f:/логическое программирование/swi/методичка swiprolog+примеры программы/a.pl.txt compiled 0.00 sec, 396 bytes
% Scanning references for 1 possibly undefined predicates
Yes
2 ?-

```

Перезагружаются все измененные файлы и файл начальной инициализации `pl.ini`; о его назначении будет оговорено позднее.

1.3. ИСПОЛЬЗОВАНИЕ ВСТРОЕННЫХ ПРЕДИКАТОВ

В интерпретаторе SWI/PROLOG имеются, как было отмечено выше, множество встроенных предикатов. Опишем некоторые из них:

consult(Имя файла).

– загрузка одного или нескольких файлов.

ПРИМЕР ИСПОЛЬЗОВАНИЯ

?- consult(test). test – имя файла

pwd

– показывает текущий рабочий каталог.

ПРИМЕР ИСПОЛЬЗОВАНИЯ

?- pwd.

c:/prolog_workspace

make

– перезагрузить все измененные файлы (*.pl). В том числе и **pl.ini**. Аналог **consult**. Удобна при редактировании файлов во внешнем редакторе.

ПРИМЕР ИСПОЛЬЗОВАНИЯ

?- make.

working_directory(X,Y)

– смена рабочего каталога, где *X* – текущий рабочий каталог, *Y* – новый рабочий каталог.

ls

– просмотр списка файлов в текущем рабочем каталоге.

ПРИМЕР ИСПОЛЬЗОВАНИЯ

?- ls.

a.pl

ЗАДАНИЕ 1.2

Переведите предложения русского языка в предикатную форму, создайте, сохраните и загрузите базу знаний.

Эллен любит чтение.

Марк любит компьютеры.

Джон любит бадминтон.

Эрик любит чтение.

Бадминтон - это вид спорта.

Теннис - это вид спорта.

Футбол - это вид спорта.

Бейсбол - это вид спорта

Спортсмен - это тот, кто любит какой-нибудь вид спорта.

Объедините эту базу с базой из Задания 1.1.

1.4. ВЫПОЛНЕНИЕ И ТРАССИРОВКА ПРОГРАММЫ

Алгоритм выполнения Пролог-программы основан на механизме прямого перебора с возвратом и операции сопоставления (унификации).

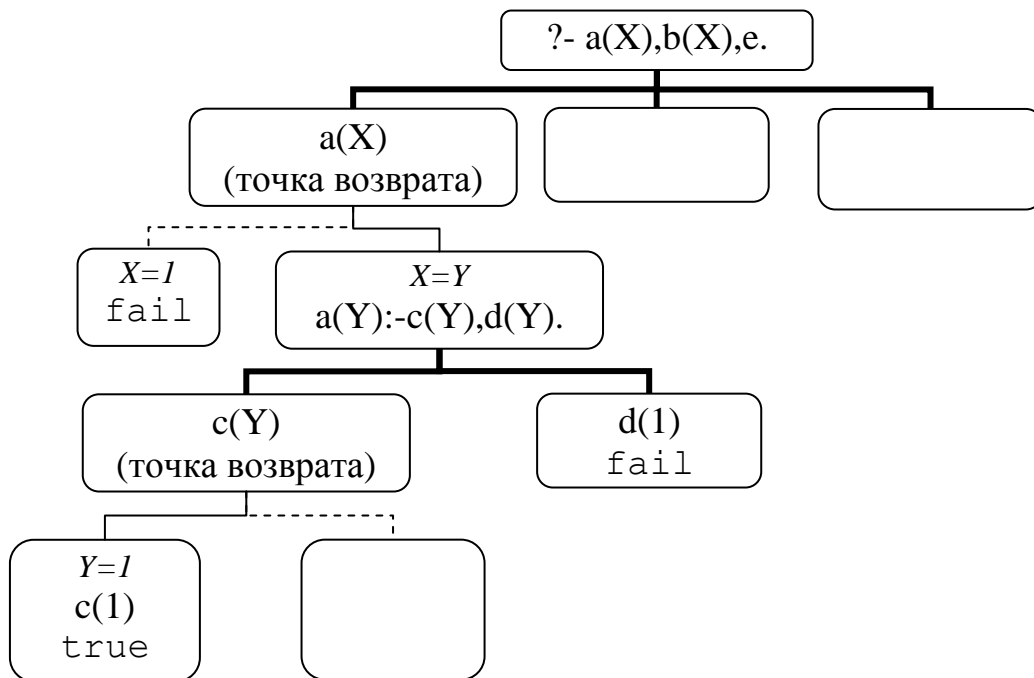
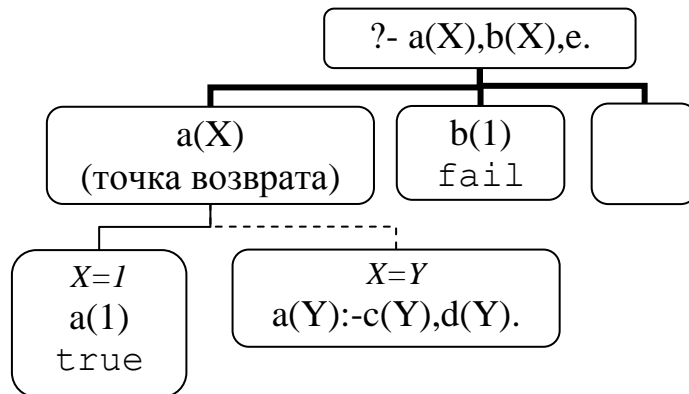
Выполнение программы начинается с запроса. Пролог-система берет первую подцель запроса и пытается доказать ее истинность. Для этого просматриваются программа (сверху вниз) и ищется первое утверждение, функтор и аргументы которого совпадают с функтором и аргументами этой подцели.

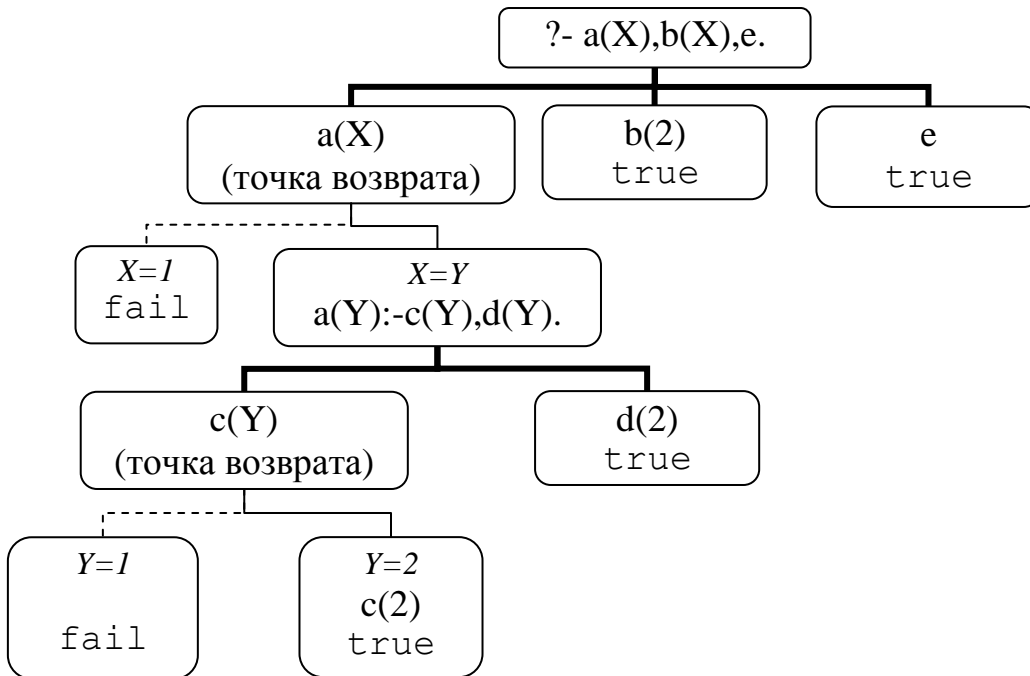
Если такое утверждение существует и происходит успешное сопоставление аргументов, тогда в случае утверждения-факта - подцель доказана, и Пролог-система переходит к доказательству следующей подцели. В случае утверждения-правила Пролог-система пытается доказать истинность подцелей тела правила слева направо.

Если при поиске решения обнаружено несколько вариантов доказательства истинности цели, то Пролог-система запоминает альтернативные варианты решения - **точки возврата**.

Если для некоторой цели нет ни одного утверждения, с которым ее можно сопоставить, то цель считается неуспешной. Если при этом остались непройденные точки возврата, то выполняется возврат (**бектрекинг**) и еще раз делается попытка доказательства подцели.

РАССМОТРИМ ПРИМЕР ВЫПОЛНЕНИЯ ПРОЛОГ-ПРОГРАММЫ

База знаний:**?- a(X),b(X),e.****a(1).****a(Y):- c(Y),d(Y).****b(2).****c(1).****c(2).****d(2).****e.****Запрос****?- a(X),b(X),e.****ДЕРЕВО ВЫВОДА**



На рисунках представлено дерево вывода. Жирными линиями в дереве обозначены **И-деревья** – для доказательства такого дерева необходимо, чтобы каждая его ветка «опиралась» на истинное утверждение. **ИЛИ-деревья** исходят из вершин с точками возврата, такое дерево истинно, если хотя бы одна ветка опирается на истинное утверждение. Пунктирные линии – альтернативные ветки, не рассматриваемые на данном этапе (либо ложные, либо еще не рассмотренные).

В данном примере получение решения складывается из трех этапов. Вначале рассматривается первая подцель – она имеет два предложения в базе, заголовки которых сопоставимы с ней, поэтому она является точкой возврата. Выбирается первое сверху предложение – это факт $a(1)$, при этом свободная переменная X получает значение 1. Первая цель запроса доказана. Вторая цель - $b(1)$ (X уже связана со значением 1) не сопоставляется ни с одним правилом базы, следовательно, является ложной.

Бэктрекинг – возврат к ближайшей точке возврата, т.е. $a(X)$. На втором дереве отображен вывод по второй альтернативе. Связываются две переменные X из запроса и Y из правила для предиката **a**. Фактически запрос **?-a(X),b(X),e** подменяется на новый набор целей **?-c(Y),d(Y),b(Y),e**. Первая подцель – это $c(Y)$ – она является новой точкой ветвления, так как унифицируется с двумя предложениями в базе. Первое предложение – факт приводит к унификации Y с 1, но следующая в наборе подцель $d(1)$ – ложна.

Третье дерево отражает альтернативу для $c(Y)$ с унификацией $Y=2$. Для этой альтернативы истинны все последующие цели из набора (они унифицируются с соответствующими фактами).

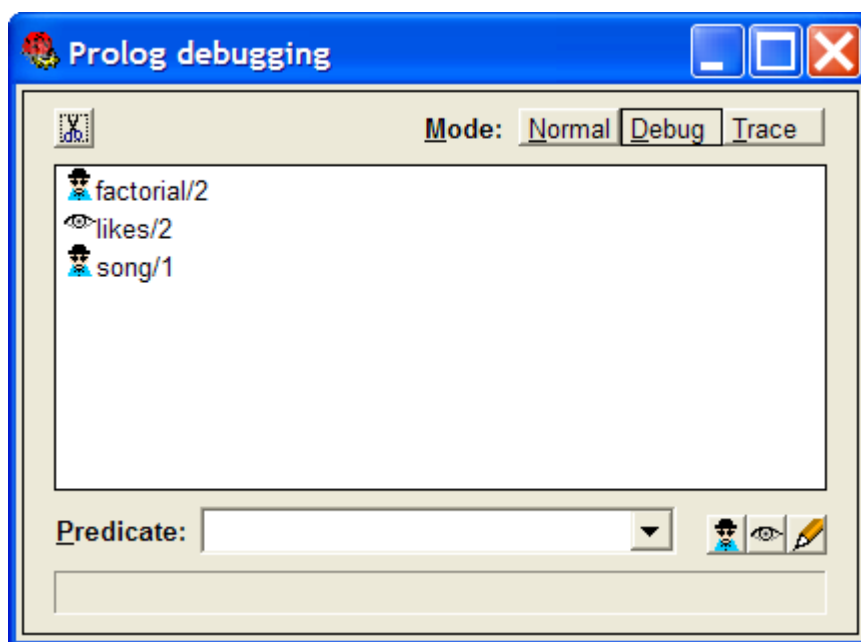
ЗАДАНИЕ 1.3

Загрузите Пролог-программу из задания 1.2.

В SWI/PROLOG есть два вида отладчиков: командный² и графический³, в последнем трассировка выглядит более наглядно.

Для перехода в режим трассировки необходимо набрать встроенный предикат **debug**. Для выхода из режима трассировки – предикат **nodebug**⁴.

Перейдите в режим трассировки. Для трассировки необходимо указать контрольные точки (*Spy points*) в меню *Debug->Edit spy points->Predicate*, где указать имена предикатов. В данном случае укажите предикат *любим*. Обратите внимание, что справа внизу находятся три кнопки, где указываются как контрольные точки, так и точки трассировки. Контрольные точки (*Spy points*) позволяют вызвать графический отладчик. Точки трассировки (*Trace points*) – консольный отладчик. Третий выбор — просмотр и редактирование файла программы, содержащей, введенный в поле *Predicate*, предикат. Внешний вид изображен на снимке ниже.



ЗАМЕЧАНИЕ. На момент написания методички графический отладчик не поддерживал русских предикатов (не работает корректно указание контрольных точек и точек трассировки через графический интерфейс). В связи с этим опишем предикаты относящиеся к отладке программы с использованием командной строки.

Предикат	Описание
trace	Включить режим трассировки. Если не указаны точки трассировки, будет производиться полная пошаговая трассировка.

² Отладка происходит в командной строке в диалоговом окне SWI/PROLOG-а и отображает последовательность выполняемых подцелей и результат выполнения.

³ Графический отладчик отображает информацию в собственном – отдельном окне.

⁴ Не забывайте каждый раз ставить точку после предикатов или группы предикатов (записанных через запятую) – каждое предложение заканчивается точкой.

tracing	Успешен, если вызывается из режима трассировки.
notrace	Выход из режима трассировки.
guitracer, gtrace	Включить графического режима трассировки. Окно графического режима трассировки включается при встрече первой контрольной точки (spy-point).
noguitracer	Отключить графического режима трассировки.
trace(Pred)	Установить точку трассировки предиката с именем Pred (то есть отображаться при трассировке будет каждое событие связанное с предикатом Pred).
trace(Pred,Ports)	<p>Установить точку трассировки предиката с именем Pred и активирующейся по событиям (при прохождении по портам) Ports. События могут быть следующих типов:</p> <ol style="list-style-type: none"> 1. fail – событие соответствующее неудачному вызову предиката с именем Pred; 2. call – событие соответствующее первому вызову предиката с именем Pred; 3. redo – событие соответствующее повторному вызову предиката с именем Pred; 4. exit – событие соответствующее успешному окончанию вызова предиката с именем Pred. <p>Сам же параметр Ports может принимать значения типов сообщений с префиксами соответствующими добавлению и удалению события (порта) из точки трассировки а так же списки таких значений. Например, запрос</p> <p>?- trace(foo/2, +fail), trace(foo/2, [+call,-fail]).</p> <p>добавляет в точку трассировки связанную с предикатом foo/2 событие fail. Для добавления события используют префикс +, для изъятия -. Кроме того можно оперировать сразу со всеми типами событий, используя обозначение all. Пример: trace(foo/2, +all).</p>
debug	Включить режим отладки. При запросах к программе в режиме отладки выводятся все события, происходящие при выполнении запроса, связанные с установленными точками отладки.
nodebug	Отключить режим отладки.
debugging	Показать отслеживающиеся точки трассировки и контрольные точки
spy(Pred)	Установить контрольную точку, связанную с предикатом Pred .

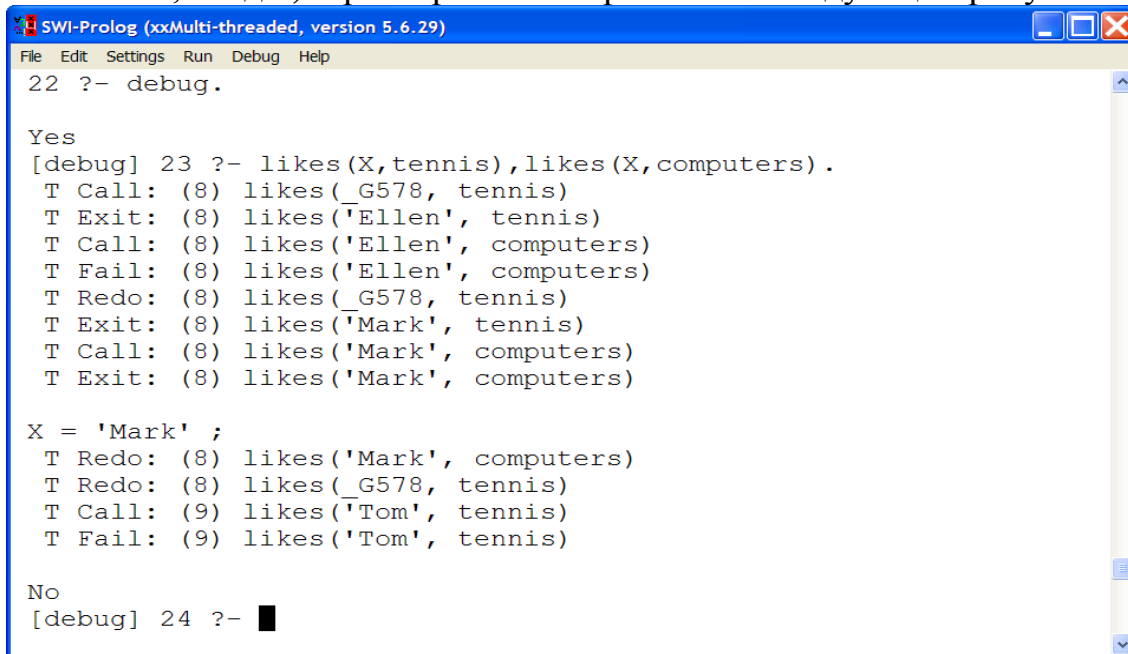
nospy(Pred)	Убрать контрольную точку, связанную с предикатом Pred.
nospyall	Убрать все контрольные точки.

Тогда именно на указанных предикатах трассировка будет останавливаться и показывать промежуточный результат. Остальные не указанные предикаты не будут показаны (если не указаны никакие контрольные точки, то режим трассировки будет неотличим от обычного режима работы интерпретатора).

Задайте запрос

?-любит(X,теннис),любит(X,компьютеры).

и посмотрите, что произойдет. После получения первого решения получите все остальные, вводя ;. Трассировка изображена на следующем рисунке:



```

SWI-Prolog (xxMulti-threaded, version 5.6.29)
File Edit Settings Run Debug Help
22 ?- debug.

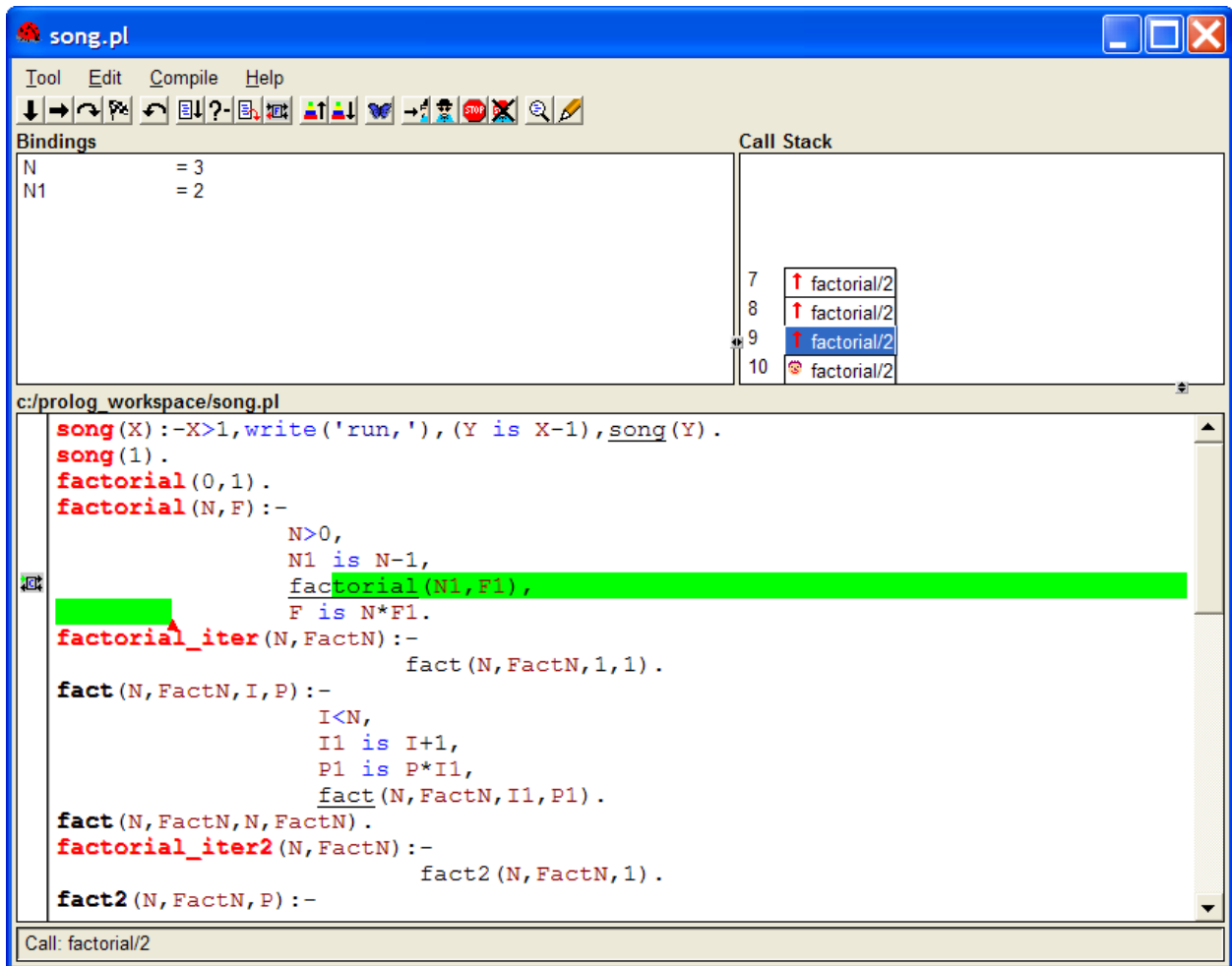
Yes
[debug] 23 ?- likes(X,tennis),likes(X,computers) .
  T Call: (8) likes(_G578, tennis)
  T Exit: (8) likes('Ellen', tennis)
  T Call: (8) likes('Ellen', computers)
  T Fail: (8) likes('Ellen', computers)
  T Redo: (8) likes(_G578, tennis)
  T Exit: (8) likes('Mark', tennis)
  T Call: (8) likes('Mark', computers)
  T Exit: (8) likes('Mark', computers)

X = 'Mark' ;
  T Redo: (8) likes('Mark', computers)
  T Redo: (8) likes(_G578, tennis)
  T Call: (9) likes('Tom', tennis)
  T Fail: (9) likes('Tom', tennis)

No
[debug] 24 ?- █

```

В левой верхней части окна графического отладчика отображены текущие значения переменных текущего выполняемого предиката, справа стек вызовов предикатов, в нижней части факты и правила, где цветом отмечен следующий вызываемый предикат. Внешний вид графического отладчика изображен на рисунке. Сверху, в виде кнопок, находятся допустимые действия. Вторая кнопка (стрелка вправо) позволяет выполнять программу пошагово (предикат за предикатом). Цвет выделения также имеет значение. Красный цвет обозначает неудачу, ложность решения.



ЗАДАНИЕ 1.4

Используя команды внешнего редактора, заменить всюду *'Eric'* на *'Tom'*. Построить дерево вывода для построенной в задании 1.2 базы данных и запроса `?-sportsman('Mark').`

ЗАДАНИЕ 1.5

Построить базу знаний на основе текста:

Живет зебра на земле. Живет собака на земле. Живет карп в воде. Живет кит в воде. Кошка живет там же, где живет собака. Живет крокодил в воде и на земле. Живет лягушка в воде и на земле. Живет утка в воде, на земле и в воздухе. Живет орел в воздухе и на земле. Живет буревестник в воде и в воздухе.

Задайте к этой базе запросы (воспользуйтесь предикатом **help** для поиска отношения "неравно").

Кто где живет?

Кто живет на земле, но не является собакой?

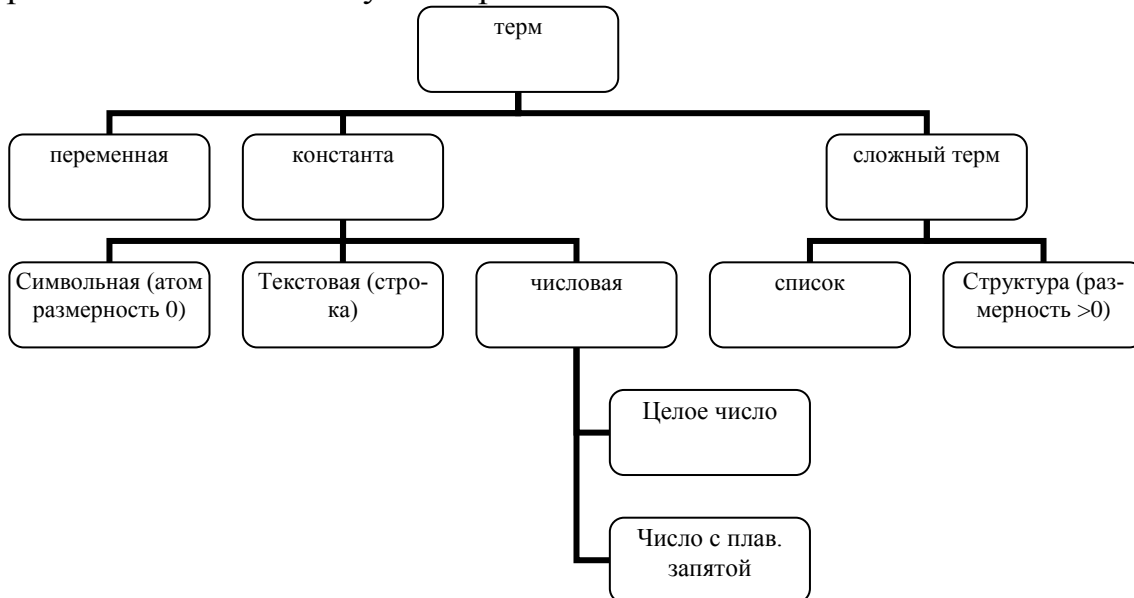
Кто живет и на земле, и в воде?

2. СТРУКТУРЫ ДАННЫХ. СЛОЖНЫЕ УТВЕРЖДЕНИЯ В ПРОЛОГЕ

ЦЕЛЬ: Знакомство со структурами данных и их унификацией. Построение сложных утверждений в Прологе с помощью связок "и", "или", "не".

2.1. СТРУКТУРА ЯЗЫКА

В основе языка Пролог лежит универсальный тип данных, называемый термом. Любой объект задачи и любое отношение между объектами представляется термом того или иного вида. Все типы данных Пролога можно рассматривать как частный случай термина:



В традиционных языках программирования базовой операцией является присваивание значения переменной. В Прологе используется более общий механизм присваивания, известный как **унификация** (сопоставление, конкретизация).

Унификация

*заключается в посимвольном или поэлементном сравнении двух конструкций языка и связывании переменных-параметров одной конструкции (образца) с соответствующими элементами другой. Унификация может быть успешной (**true**) или неуспешной (**fail**). В случае успеха оба унифицируемых термина могут измениться, при этом они станут синтаксически равными. Кроме того, после унификации две переменные могут оказаться взаимосвязанными.*

Унификация сложных термов определяется рекурсивно.

ЗАДАНИЕ 2.1

Прочитать приложение 1.

2.2. СОСТАВНЫЕ ЦЕЛИ-ЗАПРОСЫ

Составная цель - это несколько предикатов, соединенных знаками запятой ("и"), точкой с запятой ("или"), которые реализуют соответствующие логические связки. В конце целевого утверждения указывается точка. Доказательство составных запросов в Прологе осуществляется слева направо.

В случае связки "или" переменные в подцелях являются локальными, и связывания одноименных переменных не происходит.

Например, в правиле

c(X):-a(X);b(X).

в подцелях a(X) и b(X) переменные X не будут связаны. Это правило можно заменить двумя:

c(X):-a(X).

c(X):-b(X).

ЗАДАНИЕ 2.2

- a) Набейте базу:/* авто(Марка,Цена,Цвет,Возраст) - предикат */
- авто(москвич,9500,зеленый,1).**
- авто(москвич,3000,синий,5).**
- авто(волга,15000,черный,1).**
- авто(волга,8000,белый,6).**
- авто('УАЗ',9000,зеленый,3).**
- авто('ВАЗ',6000,белый,4).**
- авто('ВАЗ',4000,синий,10).**
- авто('ВАЗ-2108',8000,серый,2).**

- b) Задайте вопрос: "Найти автомобиль с ценой меньше 5000, любого цвета, и любым сроком эксплуатации."
- c) Добавьте к базе правило: "Следует купить автомобиль, если он зеленый или синий, не дороже некоторой суммы и его срок эксплуатации меньше 3-х лет".
- d) Задайте вопрос к полученной программе: "Какие автомобили можно купить за 10000 ?" Найдите все решения.

2.3. СОСТАВНЫЕ ОБЪЕКТЫ

Составные объекты или структуры состоят из имени структуры-функтора и нескольких компонент структуры, которые могут быть любыми объектами, в том числе структурами.

ЗАДАНИЕ 2.3

Нарисуйте деревья для структур:

авто(марка('ВАЗ',2108),цвет(синий),цена(\$ (7500))).

имеет(студент('Олег'),книга('Грибоедов','Горе от ума')).

ЗАДАНИЕ 2.4

a) Введите базу:

имеет('Олег', книга('Пушкин','Капитанская дочка')).

имеет('Лена', книга('Монтень','Опыты')).

имеет('Ира',платок(синий)).

имеет('Лена', платок(красный)).

ЗАМЕЧАНИЕ. Для создания операторной формы предиката с функтором **Name** можно воспользоваться встроенным предикатом **op(Precedence, Type, Name)**, где **Precedence** – приоритет вводимого оператора (целое неотрицатель-

ное число), **Type** – тип вводимого оператора (инфиксный, постфиксный и т.п.). Операторная форма позволяет записывать факты, запросы и правила в операторном виде. Разъясним это на примере.

?-op(550,xfx,имеет).

Здесь тип **xfx** значит, что оператор **имеет** не обладает ни правой, ни левой ассоциативностью (в отличие от случаев **xfy** и **yfx** соответственно). После выполнения такого запроса можно ввести вышеуказанный факт в виде:

'Олег' имеет книга('Пушкин','Капитанская дочка').

Так же можно написать запросы в функциональной и операторной формах:

?- имеет(Б,Ъ).

Б = 'Олег',

Ъ = книга('Пушкин', 'Капитанская дочка').

?- Б имеет Ъ.

Б = 'Олег',

Ъ = книга('Пушкин', 'Капитанская дочка').

Как видно из выполнения запросов операторная и функциональная формы записи предикатов дают один и тот же результат. Подробнее о предикате **op/3** можно прочитать во встроенной справке (для запуска справки можно использовать запрос **?-help(op).** или **?-help(op/3).**).

b) Задайте вопросы к этой базе данных:

Кто имеет какую-нибудь книгу Монтеня?

Кто какую книгу имеет?

Кто что имеет?

Верно ли, что Лена имеет синий платок?

2.4. ОТРИЦАНИЕ В ПРОЛОГЕ

Логическая операция "не" - отрицание в SWI/PROLOG - обозначается через **not** и применяется к предикатам. Отрицание некоторого предиката является истинным, если невыполнимым является исходный предикат, т.е. не существует возможности доказательства его истины.

База знаний Пролога описывает замкнутый мир объектов и отношений, поэтому отрицательный ответ на вопрос может обозначать как логическую ложь, так и ответ "не знаю".

Чаще всего операция **not** используется в определении цели "проверка" в схеме программирования "метод генерации и проверки", который состоит в порождении множества кандидатов в решение и дальнейшей их фильтрации:

решение (X) : - кандидат (X) , проверка (X) .

ЗАДАНИЕ 2.5

Задайте к базе из задания 1.5 запросы:

Кто живет ровно в одной среде обитания?

Кто живет хотя бы в двух (ровно в двух) средах обитания?

ЗАДАНИЕ 2.6

a) Опишите базу знаний на Прологе со следующими фактами:

Иван имеет 10000 руб. Иван имеет телевизор. Иван имеет машину-вишневую "Волгу". Иван имеет магнитофон. Петр имеет 5000 руб. Петр имеет телевизор. Петр имеет холодильник. Коля имеет 20000 руб. Коля имеет телевизор.	имеет('Иван',rub(10000)). имеет('Иван','телевизор'). ...
---	--

b) Задайте к ней вопросы:

Что имеет Петр?

Кто имеет 10000 рублей?

Кто что имеет?

Какие вещи имеет Иван, которых нет у Николая?

(Деньги - не вещь)

c) Добавьте к этой базе данных следующие факты:

Цена машины "Волга"-32000.

Цена телевизора- 8400.

Цена холодильника - 4200.

Цена магнитофона - 3500.

Цена видео - 12000.

Цена приемника - 1300.

Цена часов - 500.

d) Задайте к расширенной базе вопросы:

Может ли Петр купить видео?

У кого больше всего денег?

e) Введите правило, определяющее, что некто может купить вещь, если у него хватит денег и этой вещи у него нет.

f) Задайте вопросы:

Что может купить Коля?

Что может купить Коля из того, что имеет Иван?

g) Добавьте к базе следующие факты:

Анна является женой Коли.

Мария является женой Ивана.

Ольга является женой Петра.

h) Введите правило, что "у жены есть все, что есть у мужа".

i) Задайте вопросы:

Имеет ли Мария машину?

Что может купить Анна, чего не имеет Мария?

3. РЕКУРСИЯ В ПРОЛОГЕ

ЦЕЛЬ: Знакомство с основным методом программирования в Прологе - рекурсией. Замена рекурсии итерацией (программирование с накопителями). Арифметические предикаты.

3.1. РЕКУРСИЯ

Основным методом программирования в Прологе является рекурсия. Рекурсивным называется определение функции (предиката) через эту же функцию (предикат).

Рекурсивное правило в общем случае имеет следующий вид:

R: - A, U, B, R, C.

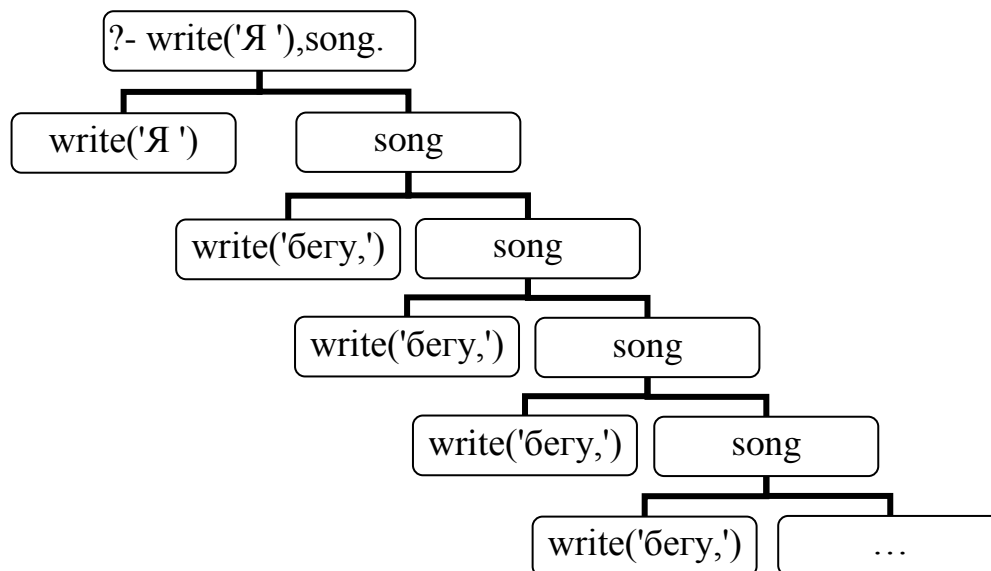
где **R** - предикат, определяющий рекурсивный вызов, **U** - предикат, определяющий условие выхода из рекурсии по неумеху, **A, B, C** - группы предикатов, не влияющие на рекурсивный вызов. В процессе рекурсии предикаты группы **C** запоминаются и выполняются лишь по завершении рекурсии (обратный ход).

ПРОГРАММА 1.

song:- write('бегу, '),song.

Рассмотрим дерево вывода ответа на запрос:

?- write('Я '),song.



Как видно, в процессе выполнения программы возникает бесконечно длинная строка. Это пример так называемой бесконечной рекурсии ("зацикливание" программы).

Запустите программу, когда надоест любоваться красотами написания этой песни, нажмите на клавишу **b** для приостановки бесконечного вывода или **e** - для выхода из интерпретатора.

Подобные программы не представляют интереса, поскольку в них отсутствует условие выхода. Условием выхода из рекурсии обычно является некий

факт или правило, при успешном выполнении которого программа заканчивает свою работу.

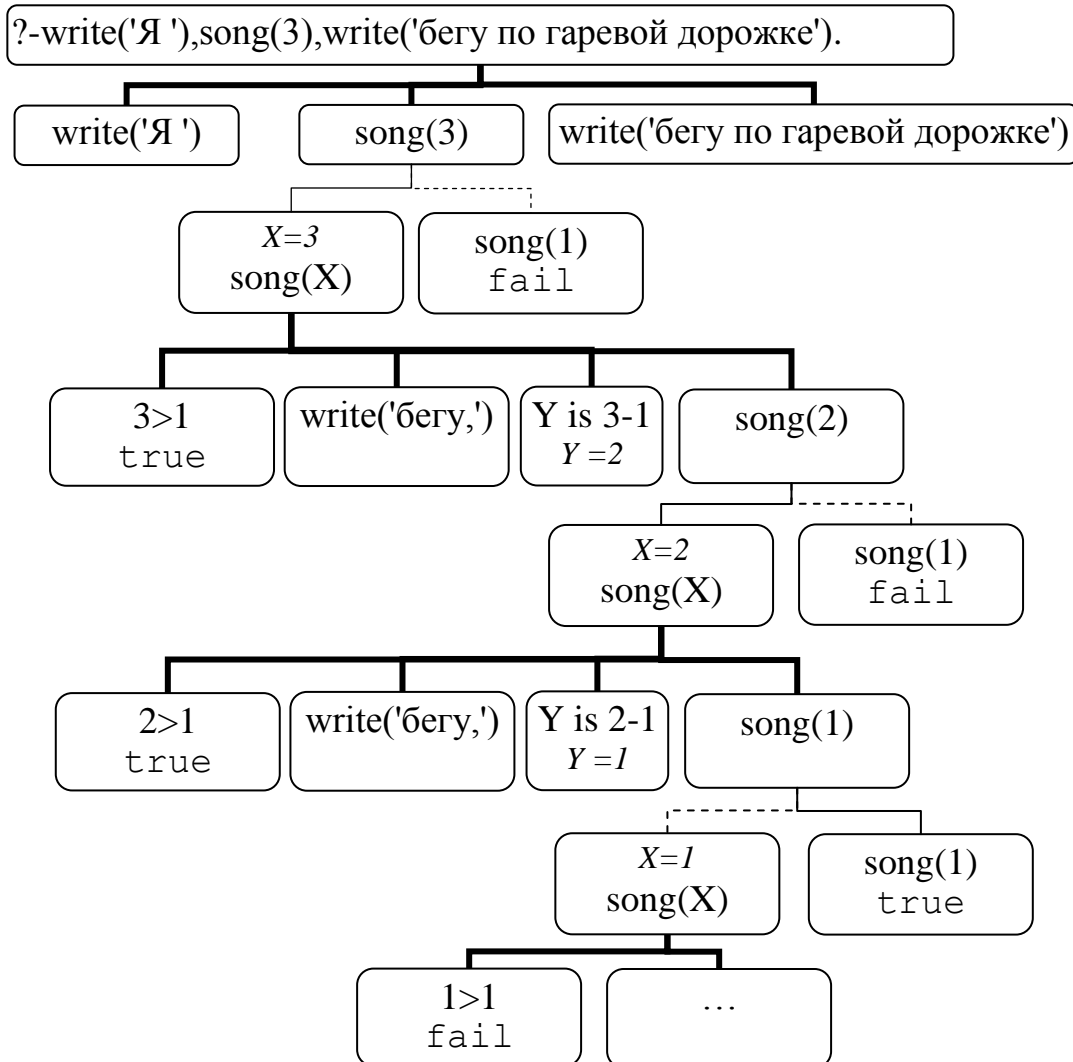
Рассмотрим следующий вариант программы.

ПРОГРАММА 2.

song(X):- (X>1), write('бегу, '), (Y is X - 1), song(Y).

song(1).

?-write('Я '), song(3), write('бегу по гаревой дорожке').



В данной программе используются встроенные арифметические предикаты отношения ($X > Y$) и присваивания ($X \text{ is } A$, где X – свободная переменная, а A - арифметическое выражение, конкретизированное числовым значением).

ЗАДАНИЕ 3.1

Запустите программы 1 и 2.

Рассмотрим задачу нахождения факториала некоторого целого неотрицательного числа.

ПРОГРАММА 3.

Рекурсивное вычисление факториала (вариант 0).

factorial(1,1) ./* Условие выхода из рекурсии $1!=1$ */

factorial(N,F) :-

N>1,

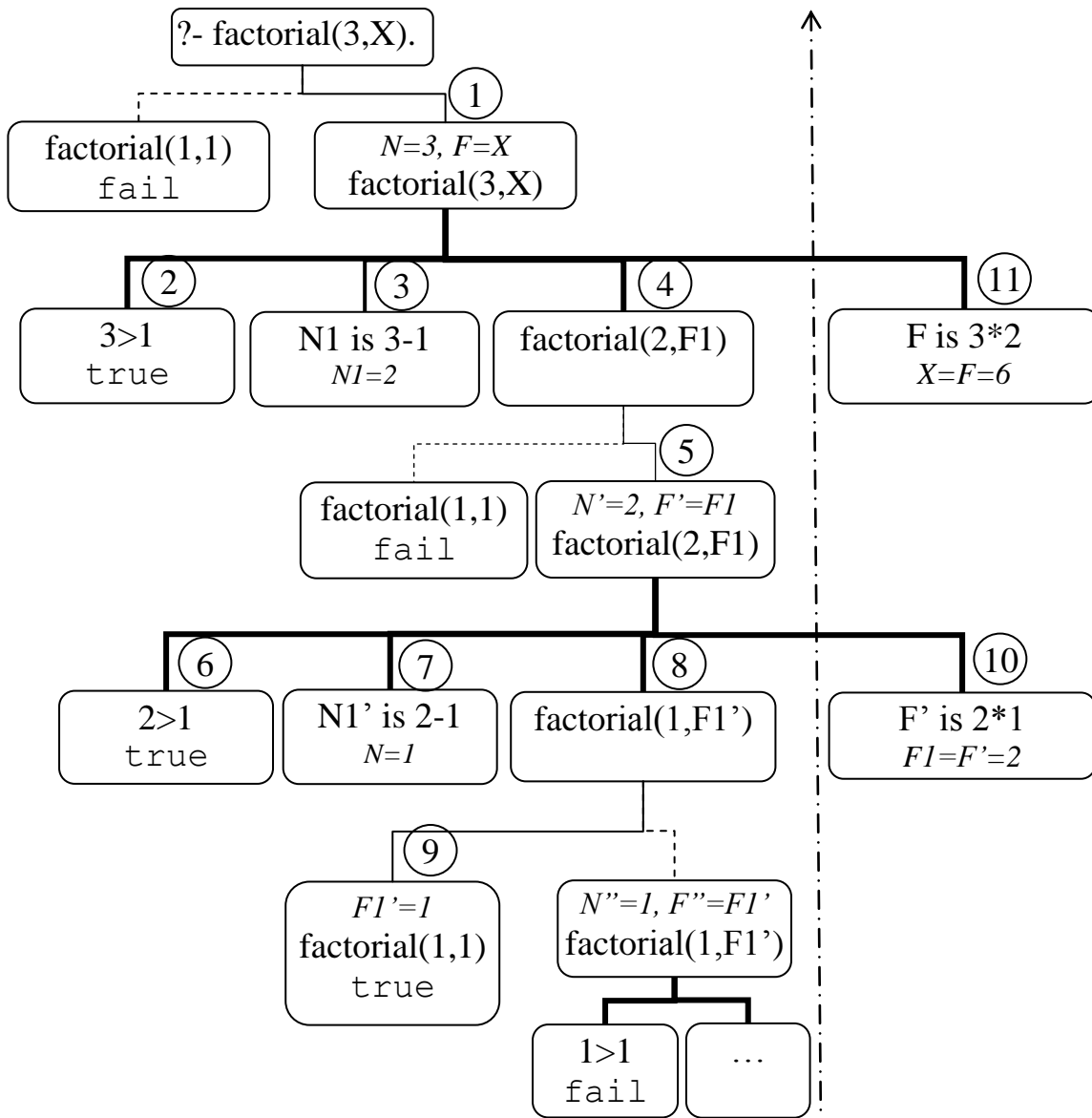
N1 is N-1,

factorial(N1, F1),

F is N*F1.

Рассмотрим дерево вывода ответа на вопрос:

?-factorial(3,X).



Слева на рисунке прямой ход до факта, останавливающего рекурсию (шаги с 1 по 9). Обратный ход с подсчетом результата – это шаги 10 и 11. В программе несколько точек возврата, но все альтернативные ветви ложны.

ЗАДАНИЕ 3.2

Запустите программу с отладчиком (переход в режим отладки производится с помощью предиката **debug**). Не забудьте установить контрольные точки или точки трассировки. Введите запросы:

?-factorial(3,X).

?-factorial(1,X).

?-factorial(0,X).

3.2. ПРОГРАММИРОВАНИЕ С НАКОПИТЕЛЯМИ

При реализации рекурсии данные помещаются в стек всякий раз, когда выполняется рекурсивный вызов. Чем больше глубина рекурсии, тем больше требуется стековой памяти. Итеративные программы работают в фиксированном объеме памяти, не зависящем от числа итераций. Итеративные вычисления можно смоделировать, используя в рекурсивных определениях с одним рекурсивным вызовом в правой части дополнительные аргументы-переменные для передачи промежуточных значений. Эти переменные называются **накопителями (аккумуляторами)**.

ПРОГРАММА 4.

Итеративное определение факториала (вариант 1).

factorial(N,FactN):- fact(N,FactN,1,1).

fact(N,FactN,I,P):- /* накопитель I - аналог счетчика */

I<N /* накопитель P – промежуточное значение факториала*/

I1 is I+1, /* FactN - значение факториала */

P1 is P*I1,

fact(N,FactN,I1,P1).

fact(N,FactN,N,FactN).

ЗАДАНИЕ 3.3

Выполните программу 4 в режиме трассировки. Введите запрос:

?-factorial(3,F).

ПРОГРАММА 5.

Итеративное определение факториала (вариант 2, более эффективный).

factorial(N,FactN):- fact(N,FactN,1).

fact(N,FactN,P):-

N>0,

P1 is P*N,

N1 is N-1,

fact(N1,FactN,P1).

fact(0,FactN,FactN).

ЗАДАНИЕ 3.4

Выполните программу 5 в режиме трассировки. Введите запрос и нарисуйте для него дерево вывода:

?-factorial(4,F).

3.3. РЕКУРСИВНЫЕ ОБЪЕКТЫ

Другим типом рекурсии в Прологе является рекурсия по данным.

ПРОГРАММА 6.

Рекурсивное определение списка студентов.

```
/*группа номер 1, состоящая из студентов 'Шекспир', 'Мольер', 'Чехов'*/
kurs(1, группа('Шекспир', группа('Мольер', группа('Чехов', empty)))).
kurs(2, группа('Гильберт', группа('Эйлер', группа('Лейбниц',
    группа('Кантор', empty)))).
```

ЗАДАНИЕ 3.5

Запустите программу 6, введите запросы:

?-kurs(1,X).

?-kurs(N, группа(X,Y)).

?-kurs(N1, группа(X, группа(Y,Z))).

ЗАДАНИЕ 3.6

Напишите программу для определения n -го числа Фибоначчи без накопителей и с накопителями. Числа Фибоначчи определяются следующим законом: первые два числа равны единице, а каждое последующее равно сумме двух предыдущих. То есть получим ряд 1,1,2,3,5,8,13,21,34,...

ЗАДАНИЕ 3.7

Напишите программу для определения суммы нечетных (четных) чисел из n первых чисел Фибоначчи.

ЗАДАНИЕ 3.8

Дано число. Проверить, является ли оно суммой нечетных (четных) чисел из n первых чисел Фибоначчи. Найти число n .

ЗАДАНИЕ 3.9

Напишите программу для вычисления функции Аккермана, определенной на множестве пар неотрицательных чисел.

$$A(X, Y) = \begin{cases} Y + 1, & \text{при } X = 0 \\ A(X-1, 1), & \text{при } X > 0, Y = 0 \\ A(X-1, A(X, Y-1)), & \text{при } X > 0, Y > 0 \end{cases}$$

Можно ли написать эту программу с накопителями?

4. УПРАВЛЕНИЕ ЛОГИЧЕСКИМ ВЫВОДОМ

ЦЕЛЬ: Знакомство с механизмами управления логическим выводом в Прологе. Отсечение. Моделирование циклов в Прологе.

4.1. ОТСЕЧЕНИЕ(CUT)

Для управления механизмом перебора используются встроенные предикаты: отсечение («!») и неудача (*fail*). Предикат отсечения применяется, когда надо изменить процесс возврата.

ПРОГРАММА 1.

```

a(1,1).           %(1)
b(2).             %(2)
b(3).             %(3)
c(2).             %(4)
c(3).             %(5)
d(4).             %(6)
a(X,Y):-b(X),!,c(Y).  %(7)
a(X,X):-d(X).       %(8)
?- a(N,M).

```

Если бы не было предиката отсечения «!», то мы получили бы шесть ответов: $N=1, M=1$; $N=2, M=2$; $N=2, M=3$; $N=3, M=2$; $N=3, M=3$; $N=4, M=4$. При выполнении предиката отсечения («проходе» «!» слева направо), **предикаты**, стоящие в правиле (7) **левее** «!», «замораживаются», т.е. устраняются все их точки ветвления и прекращается поиск альтернативных решений для $b(X)$, а для $a(X,Y)$ - использование альтернативных утверждений (8), лежащих ниже правила (7). Для предиката $c(Y)$ продолжается поиск альтернативных решений, но невозможен возврат левее «!». Получим три ответа: $N=1, M=1$; $N=2, M=2$; $N=2, M=3$.

A :- B , C , D , ! , E , F .

Если подцель « E,F » - неуспешна, бектрекинг попадает на «!», и вся цель A - неуспешна.

Можно выделить три случая использования отсечения.

- I.** Если при некоторых условиях какая-либо цель никогда не должна быть успешной, комбинация **cut-fail** исключит выполнение остальных правил, согласующихся с этой целью.

Например, предикат **not(P)** можно определить с помощью отсечения следующим образом:

```

not(P) :- P,!,fail.
not(_ ) :- true.

```

- II.** Для устранения бесконечных циклов.

В программе 2, приведенной ниже, с помощью отсечения обеспечивается выход из рекурсии. При выполнении правила 1 по предикату отсечения происходит замораживание всех альтернативных утверждений для ***factorial***, стоящих ниже правила 1, (т.е. прекращается выполнение рекурсивного правила 2).

ПРОГРАММА 2.

```

factorial(0, 1):- !. /* Условие выхода из рекурсии 0!=1 */
factorial(N,F) :- N1 is N-1, factorial(N1, F1), F is N*F1.

```

- III.** При программировании взаимоисключающих утверждений.

Например,

```

sign(X,-1):- X < 0,!.
sign(X,0):- X = 0,!.

```

$\text{sign}(X,1).$ ⁵ $\% X > 0.$

ЗАДАНИЕ 4.1

Нарисуйте дерево вывода ответа на запрос

?- factorial(3,F).

ЗАДАНИЕ 4.2

Напишите программу для определения размера одежды, используя предикат `размер(Номер, Рост)` и следующие критерии определения номера:

Номер	1	2	3	4	5
Интервал	[158,164)	[164,170)	[170,176)	[176,182)	[182,188]

Например, второй рост можно определить правилом

размер(2,R):- R >= 164, R < 170.

Добейтесь наименьшего числа сравнений в программе, используя отсечение. Рассмотрите случай неуспешного определения роста.

4.2. ОРГАНИЗАЦИЯ ЦИКЛА ВAF-МЕТОД

Первый метод организации повторений получил название **BAF-метода** (Backtrack After Fail - возврат после отказа).

Предикат отказа *fail* используется для получения гарантированного неуспеха при доказательстве некоторой цели. Например, правило

A:- B,fail.

будет выполняться столько раз, сколько имеется альтернатив для **B** в этом правиле.

ПРОГРАММА 3.

a:- write(1).

a:- write(2).

b(X):- a,X='еще'.

c:- a.

d:- a,fail.

?-b(X).

?-c.

?-d.

ЗАДАНИЕ 4.3

Выполните программу 3 с данными запросами. Объясните результаты и нарисуйте деревья вывода.

ЗАДАНИЕ 4.4

Используя предикат **fail**, напишите правило, которое позволило бы распечатать столицы всех стран из базы.

country('England','London').

country('Russia','Moscow').

country('France','Paris').

⁵ До последнего правила вывод дойдет только, если $X > 0$, во всех остальных случаях это правило не будет даже рассматриваться как альтернатива в точке возврата (точка будет заморожена после выполнения операции отсечения).


```
country('China','Pekin').
country('Japan','Tokyo').
country('Italy','Rome').
```

4.3. ОРГАНИЗАЦИЯ ЦИКЛА UDR-МЕТОД

Второй способ организации повторений получил название **UDR-метода** (User Defined Repeat - повторение, управляемое пользователем).

Встроенный предикат **repeat** позволяет генерировать альтернативные решения с помощью механизма бэктрекинга, причем это возможно для целей, которые не всегда успешно согласовываются при первом обращении, либо которые могут иметь много решений. Всякий раз, когда при бэктрекинге происходит возврат к **repeat**, этот предикат успешно согласовывается, и при последующем согласовании предикатов, стоящих правее **repeat**, переменные могут конкретизироваться различными значениями.

Предикат **repeat** определяется следующим образом:

```
repeat.
```

```
repeat:- repeat.
```

ЗАМЕЧАНИЕ. Предикат **repeat /0** является встроенным в SWI-Prolog. При попытке его переопределения интерпретатор скорее всего выдаст сообщение об ошибке.

ЗАДАНИЕ 4.5

Выполните программу 3 с запросом

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

Постройте дерево вывода и объясните результат.

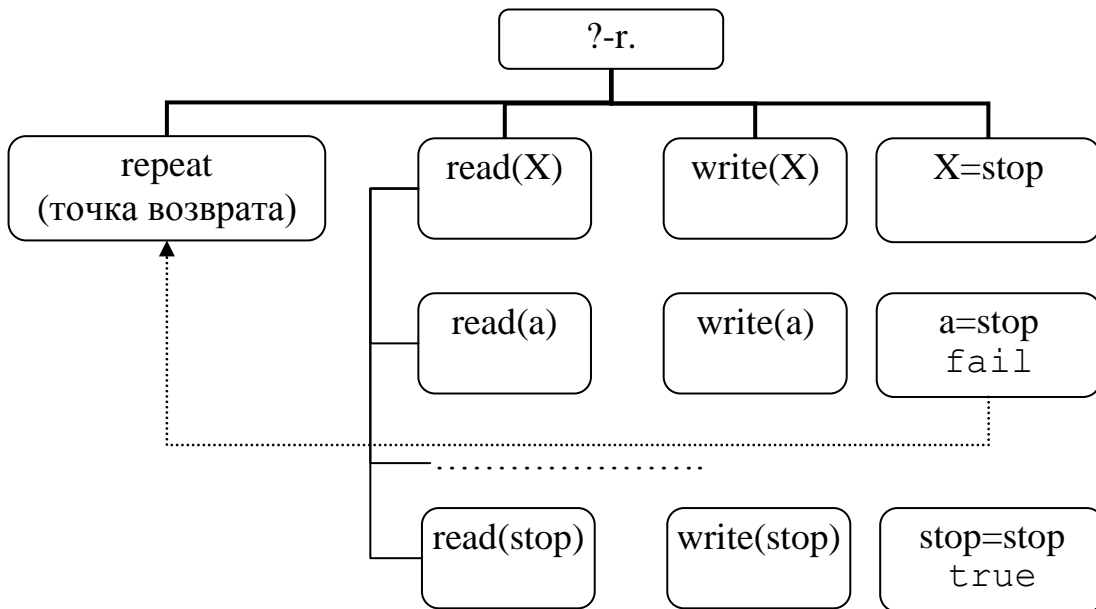
ПРОГРАММА 4.

/* ввод с клавиатуры слов и вывод их на экран до тех пор, пока не будет введено слово **stop** (в конце терма, вводимого **read**, необходимо поставить точку и нажать клавишу **Enter**) */

```
r:- repeat, read(X), write(X), X=stop.
```

```
?-r.
```

Вывод схематично можно представить так:



ЗАДАНИЕ 4.6

Выполните программу 4 в режиме трассировки⁶.

Можно предложить более общую схему организации цикла с помощью предиката *repeat* при выполнении некоторого условия.

4.4. ИСПОЛЬЗОВАНИЕ НАДРЕЗОВ (SNIP)

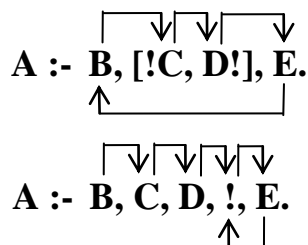
Надрез обозначается открывающей скобкой *[!* и закрывающей скобкой *!]*. Например, для правила

цель:- подцель1,[! подцель2,подцель3 !],подцель4.

действие надреза распространяется на подцели *подцель2* и *подцель3*, расположенные между этими двумя скобками.

Snip аналогичен *cut*, однако отличается от него тем, что если после бектрекинга управление передается *snip*, весь предикат неуспешным не будет. Бектрекинг лишь "пропустит" те подцели, которые находятся внутри *snip* и продолжится для подцелей, которые расположены раньше *snip*⁷.

Отличие надреза от отсечения:



Для *cut* если E неуспешна, бектрекинг попадает на *cut* и весь предикат A неуспешен, в отличие от ситуации со *snip*. Для *snip*, если подцель E неуспешна, то бектрекинг попадает на подцель B.

⁶ Не забывайте ставить точки в конце вводимых слов.

⁷ Т.е. «замораживаются» точки возврата только для предикатов внутри надреза.

Надрезы целесообразно использовать в случаях, когда нужно ограничить бектрекинг для исключения ненужного поиска, но отсечение не требуется.

УПРАЖНЕНИЕ. Вообще говоря, надрезы не реализованы в SWI-Prolog. Используя отсечение, реализуйте надрез (т.е. «заморозьте» точки возврата только для предикатов, которые должны быть внутри надреза). Одна из реализаций может быть представлена в виде схемы (по результату равносильна вышеприведённой схеме надреза) :

A:-B, T, E.

T:-C, D, !.

b(2). % (1)

b(3). % (2)

c(2). % (3)

c(3). % (4)

d(4). % (5)

d(5). % (6)

e(10). % (7)

e(11). % (8)

e(12). % (9)

a(X,Y,Z,W):-b(X),[! c(Y), d (Z) !],e(W). % (10)

a(X,X,X,X):-d(X). % (11)

/* В SWI-Prolog надрез можно реализовать через отсечение следующим образом: */

a(X,Y,Z,W):-b(X),f(Y,Z),e(W).

a(X,X,X,X):-d(X).

f(Y,Z):-C(Y),d(Z),!.

2 ?- a(X,Y,Z,W).

X = 2,Y = 2,Z = 4,W = 10 ; X = 2,Y = 2,Z = 4,W = 11 ;

X = 2,Y = 2,Z = 4,W = 12 ; X = 3,Y = 2,Z = 4,W = 10 ;

X = 3,Y = 2,Z = 4,W = 11 ; X = 3,Y = 2,Z = 4,W = 12 ;

X = 4,Y = 4,Z = 4,W = 4 ; X = 5,Y = 5,Z = 5,W = 5.

Как видно из примера надрез «замораживает» предикаты $c(Y)$ и $d(Z)$ в строке (10). Поэтому при получении вывода с использованием этой строки значения переменных Y , Z при поиске всех альтернативных решений не меняются – для $c(Y)$ и $d(Z)$ не ищется других альтернатив, кроме как $c(2)$ и $d(4)$ соответственно.

ЗАДАНИЕ 4.7

Используйте базу данных из задания 1.6 лабораторной работы 1. Добавьте факты для каждого животного X

животное(X).

Для исключения повторения названия животных на запросы
"Кто живет хотя бы (ровно) в двух средах обитания?"

можно использовать надрез

цель(X):- животное(X),[!живет(X,Y),живет(X,Z),Y\=Z!].

5. СПИСКИ

ЦЕЛЬ: Знакомство с понятием списка и операциями над списками.

Список

- это упорядоченная последовательность элементов. Элементами списка могут быть любые термы Пролога. Удобной формой записи списков является так называемое списочное обозначение. В данном обозначении каждый элемент списка отделяется от соседнего запятой, а весь набор элементов заключается в квадратные скобки, например, **[a,b,c,d]**.

Фактически список - это структура с функтором '•'/2 (точка с арностью 2). Согласно этому определению, список состоит из первого элемента и хвоста, который представляет собой список из остальных элементов.

Пустой список - это список, не содержащий ни одного элемента, он обозначается [].

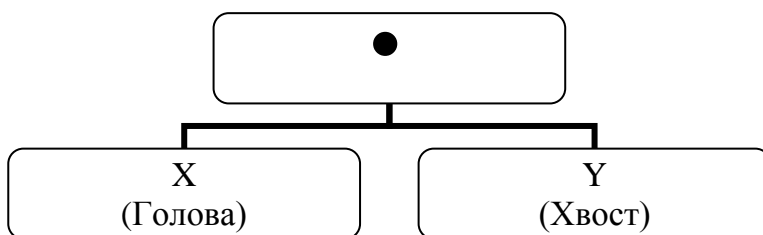
Примеры.

N	элементы списка	запись со скобками	запись с функтором '•'
1	a	[a]	'•'(a, [])
2	a b c	[a,b,c]	'•'(a, '•'(b, '•'(c,[])))
3	[a]	[[a]]	'•'('•'(a,[],[]))
4	[]	[[]]	'•'([],[])
5	[a] [b,c]	[[a],[b,c]]	'•'('•'(a,[]), '•'(b, '•'(c,[])))

Список унифицируется с другим списком, если попарно унифицируются их элементы.

Список может делиться на "голову" и "хвост" с помощью операции отделения головы, которая обозначается вертикальной чертой (|), т.е. [Голова|Хвост]. Голова - фиксированное количество элементов. Хвост - список из оставшихся элементов списка. Чаще всего используется голова, состоящая из одного элемента.

В виде дерева список [X|Y] изображается следующим образом:



Примеры сопоставления списков со списком [Голова|Хвост]

N	Список	Голова	Хвост
1	[a]	a	[]
2	[a,b,c]	a	[b,c]
3	[[a]]	[a]	[]
4	[]	не сопоставляется	не сопоставляется
5	[a [c,d]]	a	[c,d]

Список **[1,2|[3,4]]** равен списку **[1,2,3,4]**. При сопоставлении со списком **[X,Y|Z]** получим **X=1, Y = 2** и **Z = [3,4]**.

В SWI/PROLOG имеется особый вид списков - **символьные списки**. Символьный список - это фактически последовательность целых чисел, соответствующих ASCII-коду символов. Символьные списки заключаются в двойные кавычки. Следующие три списка являются сопоставимыми:

"abc" **[97, 98, 99]** **'!(97,'!(98,'!(99),[]))'**

ПРОГРАММА 1 Разделение списка на голову и хвост.

write_list([]).

write_list([H|T]):- /* разделение списка на голову и хвост, */

write(H), nl, /* печать головы, пропуск строки */

write_list(T). /* рекурсивный вызов предиката от оставшегося списка */

ЗАДАНИЕ 5.1

Запустите программу 1, посмотрите результат для списка **[1,2,a,3]**.

ПРОГРАММА 2. Конкатенация и разбиение списков.

append([],List,List).

append([X|L1],List2,[X|L3]):-append(L1,List2,L3).

ЗАМЕЧАНИЕ. Предикат **append /3** является встроенным в SWI-Prolog.

Используя предикат **append(X,Y,Z)**, осуществите склейку списков **[1,2,3]** и **[4,5]**, вычитание списка **[4,5]** из **[1,2,3,4,5]** и "разбиение" списка на две части в режиме трассировки.

ЗАДАНИЕ 5.2

Используя понятие списка, напишите программы для решения следующих задач:

- Определение длины списка.
(Длина пустого списка равна 0.
Длина непустого списка равна длине его хвоста плюс 1.)
- Определение принадлежности элемента к списку.
(Элемент принадлежит списку, если он - его голова. Элемент принадлежит списку, если он принадлежит хвосту списка.)
- Поиск суммы элементов числового списка.
- Поиск максимального и минимального элементов числового списка с использованием одной рекурсии.

- Инверсия списка.

ЗАДАНИЕ 5.3

Напишите программу сортировки числового списка методом "пузырька".

6. МОДИФИКАЦИЯ УТВЕРЖДЕНИЙ ПРОГРАММЫ. РАБОТА С БАЗОЙ ДАННЫХ.

ЦЕЛЬ. Научиться работать с динамическими базами знаний.

Встроенные предикаты `asserta/1` и `assert/1` позволяют добавлять новые утверждения в базу данных Пролога (в начало и в конец соответственно), а `retract/1` - удалять утверждение, заголовок которого унифицируется с аргументом. Например, можно составить запрос, который опрашивает пользователя о знании языков:

```
язык(итальянский).
язык(немецкий).
язык(японский).
язык(французский).
язык(английский).
```

```
?- write('Введите Ваше имя '),
   read(Имя),
   язык(Яз),
   write('Знаете ли Вы '), write(Яз),
   write(' язык'),nl,
   read(да),
   assert(владеет(Имя,Яз)),fail.
```

Часто используется предикат `retractall/1`, удаляющий из базы данных все предложения, заголовки которых унифицируются с аргументом.

Рассмотрим пример базы данных, содержащей информацию, относящуюся к членам клуба: фамилию и возраст, размер членского взноса, а также данные о том, уплачен ли взнос. Такие данные будут представлены в базе фактами, после добавления их командой `пополнить_состав(член(...))` (см. ниже в заданиях):

```
%член(_Фамилия,_Возраст,_Данные_об_уплате)
:-dynamic(член/3).
член('Иванов',15,уплачено).
член('Иванов',33,не_уплачено).
член('Хромов',40,не_уплачено).
```

Конструкция `:-dynamic(член/3)` необходима для того, чтобы возможно было изменять базу фактов, связанную с предикатом `член/3` с помощью предиката `assert` во время выполнения. Для безвозвратной отмены действия предиката `dynamic` используют предикат `compile_predicates(List_of_nameArity)`. По-

сле использования этого предиката факты указанные в аргументе не могут быть изменены динамически до конца программы (пример использования: **compile_predicates(член/3)**).

Размер взноса не указывается, он определяется по возрасту:

взнос(Возраст,рублей(1)):-Возраст<18.

взнос(Возраст,рублей(2)):-Возраст>=18.

Рассмотрим некоторые операции над базой данных.Внести сведения о новом члене организации:

пополнить_состав(Член):-assert(Член).

Выдать на терминал сведения о членстве в организации:

выдать_сведения(член(Фамилия,Возраст,Данные_об_уплате)):-
член(Фамилия,Возраст,Данные_об_уплате),
взнос(Возраст,Сумма),
write(член(Фамилия,Возраст,Сумма,Данные_об_уплате)),nl,fail.
выдать_сведения(_).

Удалить сведения о членах организации:

сократить_состав(Член):-retractall(Член).

Внести данные о том, что член организации уплатил членский взнос:

запись_об_уплате(член(Фамилия,Возраст)):-
retract(член(Фамилия,Возраст,не_уплачено)),
assert(член(Фамилия,Возраст,уплачено)).

ЗАДАНИЕ 6.1

Выполните следующие запросы к программе

?-пополнить_состав(член('Пронин',45,уплачено)).

?-пополнить_состав(член('Сидоров',27,не_уплачено)).

?-пополнить_состав(член('Данилов',12,не_уплачено)).

?-пополнить_состав(член('Иванов',15,уплачено)).

?-пополнить_состав(член('Иванов',33,не_уплачено)).

?-пополнить_состав(член('Хромов',40,не_уплачено)).

?-выдать_сведения(_).

ЗАДАНИЕ 6.2

Внесем данные о том, что Хромов оплатил взнос и просмотрим

содержание всей базы и отдельно список должников, а затем исключим всех, кто не уплатил взнос, выполнив запросы:

```
?-запись_об_уплате(член('Хромов',40)),выдать_сведения(_).
?-выдать_сведения(член(_,_,не_уплачено)).
?-сократить_состав(член(_,_,не_уплачено)),выдать_сведения(_).
```

Предикат `выдать_сведения/1` пример применения вынуждаемого возврата для получения всех возможных вариантов согласования цели. При этом результат, выводимый на экран, уничтожается механизмом возвратов из памяти программы, но его можно накапливать, используя побочный эффект действия предикатов `assert/1` и `retract/1`. Это накопление реализуется процедурами действия над счетчиком:

```
установить_счетчик(Имя,Начало):-
retractall(счетчик(Имя,_)),
assert(счетчик(Имя,Начало)).
```

```
увеличить_счетчик(Имя,Прирост):-
retract(счетчик(Имя,Значение)),
Новое_значение is Значение+Прирост,
assert(счетчик(Имя,Новое_значение)).
```

```
сбросить_счетчик(Имя,Значение):-
retract(счетчик(Имя,Значение)).
```

Для иллюстрации рассмотрим процедуру, подсчитывающую число членов клуба:

```
подсчет_членов(член(Фамилия,Возраст,Данные_об_уплате),_-):
установить_счетчик(число_членов,0),
член(Фамилия,Возраст,Данные_об_уплате),
увеличить_счетчик(число_членов,1),
fail.
подсчет_членов(_,Счетчик):-
сбросить_счетчик(число_членов,Счетчик).
```

Если программа предназначена для получения ряда значений при помощи одного из встроенных предикатов чтения входных данных, то такой подход использовать уже нельзя, так как `read/1` и другие читают новое значение только при первом вызове и не делают этого при возврате. Эту трудность можно пре-

одолеть, добавив в программу определение предиката repeat/0 - согласуемого всегда, в том числе и при возвратах:

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

Используем этот прием для определения процедуры меню:

```
меню:-repeat,  
nl,nl,write(' МЕНЮ'),nl,  
write('1. Сведения о членах клуба. '),nl,  
write('2. Посчитать количество членов. '),nl,  
write('3. Добавить запись о члене клуба. '),nl,  
write('0. ВЫХОД'),nl,nl,  
write('Введите номер пункта меню '),  
read(X),  
обработать(X).
```

Предикат обработать/1 предназначен для организации остановки или повторения соответственно сделанному пользователем выбору:

```
обработать(0):-!.   
обработать(X):-пункт(X),fail.
```

пункт/1 выполняет соответствующую операцию

```
пункт(1):-nl,write('Состав клуба:'),  
nl,выдать_сведения(_),!.   
пункт(2):-nl,подсчет_членов(член(_,_),Число),  
write('Количество членов клуба = '),  
write(Число),nl,!.   
пункт(3):-nl,write('Введите данные нового члена клуба '),nl,  
write('Фамилия '),read(Фамилия),  
write('Возраст '),read(Возраст),  
write('Отметка об уплате взноса '),read(Данные_об_уплате),  
пополнить_состав(член(Фамилия,Возраст,Данные_об_уплате)),!.   
пункт(_):-write('Такой пункт неопределен!'),nl,!. 
```

ЗАДАНИЕ 6.3

Запустите программу, выполнив запрос:

?-меню.

ЗАМЕЧАНИЕ. В случае ввода нескольких слов для одного запроса от предиката **read/1** или в случае если ввод начинается с заглавной буквы стоит заключать вводимую последовательность символов в апострофы. Причина в том, что предикат **read(X)** считывает терм из текущего потока ввода и унифицирует его с термом **X**.

ЗАДАНИЕ 6.4

Сохраните текст программы под новым именем. Добавьте пункты удаления члена клуба и внесения отметки об уплате взноса (обратите внимание, что для этого необходимо ввести фамилию и возраст члена клуба). А также удаления неплательщиков.

ЗАДАНИЕ 6.5

Самостоятельно напишите базу данных "Классный журнал", содержащую информацию о сроках и результатах сдачи экзаменов школьниками по трем предметам: математике, физике и информатике. Напишите меню для работы с ней. Ваша программа должна предоставлять возможность добавлять записи о сдаче, пересдаче экзаменов; получения ведомости; получения списка неуспевающих; операции отчисления и т.д.

7. ПРИЛОЖЕНИЕ 1. ТИПЫ ДАННЫХ SWI/PROLOG

7.1. ПЕРЕМЕННЫЕ

Именем переменной может быть любая последовательность латинских и русских букв и цифр, начинающаяся с прописной буквы или символа подчеркивания "_"; это может быть также одиночный символ подчеркивания "_", представляющий анонимную (безымянную) переменную.

ПРИМЕРЫ переменных:

Balloon **X** **_0039** **_** **A1**

Свободная (не имеющая значения) переменная может унифицироваться с любым термом, при этом она может конкретизироваться значением (в случае константы или сложного терма), либо связываться с другой переменной. Для связанной переменной в сопоставлении используется ее значение. Анонимная переменная значения не получает, с ней всегда возможно сопоставление.

7.2. СИМВОЛЬНЫЕ КОНСТАНТЫ ИЛИ АТОМЫ

Символьные константы могут содержать латинские буквы, цифры и специальные (прочие) символы (в том числе кириллицу). Если они начинаются с прописной буквы или цифры или содержат специальные символы, то они должны быть заключены в апострофы. Если апостроф используется внутри атома, он удваивается.

ПРИМЕРЫ символьных констант:

tennis **'23'** **'Paris'** юла **'----->'** **'Mary's'**

Символьная константа всегда унифицируется с такой же символьной константой. Длина символьной константы не должна превышать 255 символов⁸.

7.3. ЦЕЛЫЕ ЧИСЛА.

Целые числа могут быть положительные и отрицательные. Целое число всегда унифицируется с равным ему целым числом. Размер целых чисел ограничен только размером предоставляемого стека.

ЗАМЕЧАНИЕ. Данное утверждение верно для версий собранных с использованием библиотеки GMP (GNU multiple precision arithmetic library).

ASCII-коды символов также являются целыми числами.

7.4. РАЦИОНАЛЬНЫЕ ЧИСЛА

Рациональное число в SWI-prolog определено (в сборке с использованием GMP) аналогично рациональному числу в математическом смысле. Рациональное число представляется в виде пары двух целых чисел (деление на ноль запрещено) и обозначается функтором **rdiv**, который может выступать в роли функции двух аргументов (любые числа - не только целые), возвращающей рациональное число, либо в роли инфиксного оператора. При оперировании справа от оператора **is** только рациональными числами результат так же будет являться рациональным числом. Целое число является рациональным.

Для приведения числа с плавающей точкой к типу рационального числа используют функции **rational(X)** и **rationalize(X)**, где **X** - число с плавающей точкой. **rational(X)** вычисляет рациональное число соответствующее машинному приближению числа **X**. **rationalize(X)** – «учитывает» ошибку округления и вычисляет рациональное число, соответствующее **X** (а не его машинному представлению).

ПРИМЕР.

?- X is rational(0.12).

X = 8646911284551353 rdiv 72057594037927936.

?- X is rationalize(0.12).

X = 3 rdiv 25.

Однако, можно убедиться, что для чисел, у которых машинное представление совпадает с самим числом, функции работают одинаково.

?- X is 1/256.

X = 0.00390625.

?- X is rational(0.00390625).

X = 1 rdiv 256.

?- X is rationalize(0.00390625).

X = 1 rdiv 256.

⁸ Не путать символ апострофа (') с двойной кавычкой (").

ПРИМЕРЫ использования рациональных чисел:

A is 2 rdiv 6	A = 1 rdiv 3
A is 4 rdiv 3 + 1	A = 7 rdiv 3
A is 4 rdiv 3 + 1.5	A = 2.83333
A is 4 rdiv 3 + rational(1.5)	A = 17 rdiv 6

7.5. ЧИСЛА С ПЛАВАЮЩЕЙ ТОЧКОЙ

Числа с плавающей точкой представляются в виде C-типа double.

ПРИМЕРЫ чисел с плавающей точкой:

0.5 55.3 -83.0E21 2134.2 122.34e25

Число с плавающей точкой унифицируется с равным ему числом с плавающей точкой. Однако следует помнить, что операции над числами с плавающей точкой выполняются приближенно и попытка унификации двух чисел с плавающей точкой, которые кажутся идентичными, может окончиться неудачей.

7.6. СТРОКИ

Строки - это текстовые константы, которые представляются в виде последовательности ASCII-символов, заключенной между знаками двойных кавычек. С точки зрения внутреннего представления строка – список из ASCII-кодов символов, образующих строку.

ПРИМЕР

“He climbed and he climbed”

?- X="Hello". % пример унификации

X = [72, 101, 108, 108, 111]

Пустая строка записывается “” (ей соответствует пустой список). Если необходимо использовать знак двойных кавычек внутри строки, то его удваивают (“”).

Строка унифицируется с идентичной ей строкой, но не унифицируется с одиночным символом, даже если она единичной длины. Строки и атомы никогда не унифицируются, даже если они содержат одинаковые символы.

7.7. СТРУКТУРЫ

Структура - универсальный тип данных, который используется для группировки термов или представления отношения между ними. Структура состоит из функтора и аргументов:

функтор(терм1, терм2,...,термN)

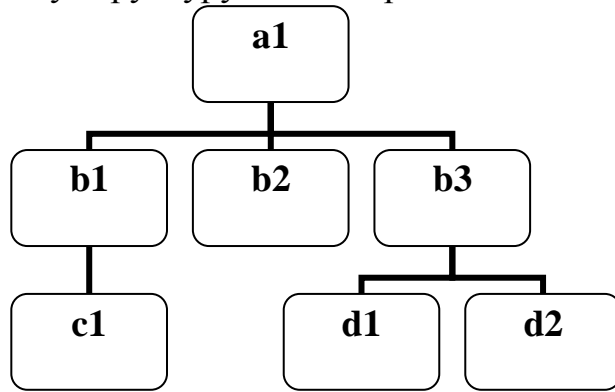
Число аргументов называется размерностью (арностью) структуры⁹.

ПРИМЕР

a1(b1(c1),b2,b3(d1,d2))

⁹ Между функтором и левой скобкой пробел не ставится.

Эту структуру можно представить в виде дерева:

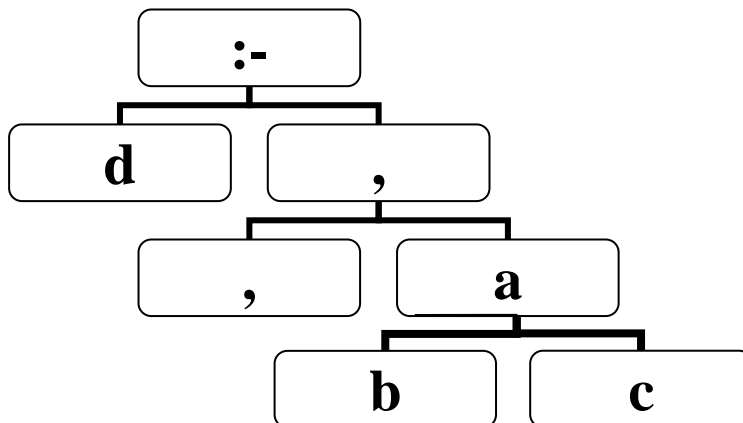


Структура унифицируется с другой структурой, если их функторы совпадают и аргументы попарно унифицируются.

Пример унифицируемых структур: `book(Author,Title)` и `book(james,"The lonely tree")`. Пример неунифицируемых структур: `name(jones,charles)` и `name(jones,chuck)`. В виде терма представляется любой объект программы. Например, структура правила:

d:-a,b,c.

может быть представлена деревом:



8. ПРИЛОЖЕНИЕ 2. ВСТРОЕННЫЕ ПРЕДИКАТЫ И ОПЕРАТОРЫ.

Предикат/оператор	Описание
write(Term)	Вывести в текущий поток вывода Term.
read(Term)	Извлечь из текущего потока вывода следующий терм и унифицировать его с Term. Ввод должен заканчиваться точкой.
tell(File)	Открыть файл для записи и перевести на него текущий поток вывода.
told	Перевести текущий поток вывода на стандартный поток вывода.

see(File)	Открыть файл для чтения и перевести на него текущий поток ввода.
seen	Перевести текущий поток ввода на стандартный поток ввода.
append(List1, List2, List3)	Успешен, когда список List3 унифицируем с объединением списков List1 и List2. Все аргументы могут быть свободными переменными. Результат унификация соответствующих списков.
append(ListOfLists, List)	Успешен, когда объединение списка списков ListOfLists унифицируем со списком List. Результат: унификация соответствующих списков.
member(Elem, List)	Успешен, когда Elem унифицируем с одним из элементов списка List. Результат – унификация соответствующего элемента списка с Elem.
nextto(X, Y, List)	Успешен, когда Y следует непосредственно за X в списке List.
delete(List1, Elem, List2)	Удалить все элементы списка List1 унифицируемые с Elem. Результат помещается в List2.
select(Elem, List, Rest)	<p>Успешен, когда Rest является списком List с удалённым элементом Elem. То есть этим предикатом можно удалять и вставлять элементы списка.</p> <p>Пример:</p> <pre> ?- select(a,L,[1,2,3]). L = [a, 1, 2, 3] ; L = [1, a, 2, 3] ; L = [1, 2, a, 3] ; L = [1, 2, 3, a] ; false.</pre>
nth0(Index, List, Elem)	Успешен, когда элемент списка List с номером Index унифицируем с Elem. Отсчёт номеров элементов начинается с 0.
nth1(Index, List, Elem)	То же, что и nth0/3, но отсчёт номеров элементов начинается с 1
last(List, Elem)	Успешен, когда Elem унифицируется с последним элементом списка List. Если хвост списка List не определён, то при бектрекинге хвост будет увеличиваться.
reverse(List1, List2)	Изменяет порядок элементов List1 и унифицирует результат с List2.
permutation(List1, List2)	Успешен, когда список List1 образован из списка List2 перестановкой элементов.
sumlist(List, Sum)	Унифицирует Sum с суммой элементов списка List.

max_list(List, Max)	Унифицирует Max с максимальным элементом списка List.
min_list(List, Min)	Унифицирует Min с минимальным элементом списка List.
IntExpr1 mod IntExpr2	Остаток от деления IntExpr1 на IntExpr2.
IntExpr1 // IntExpr2	Целая часть от деления IntExpr1 на IntExpr2
sleep(Time)	Приостановление выполнения на Time секунд.
Term =.. List	<p>Выражение истинно когда терм Term инициализируется с термом соответствующим списку List, головой которого является функтор терма, а оставшийся хвост представляет собой список аргументов терма.</p> <p>Пример:</p> <pre> ?- a(b(1),2,3,X)=..L. L = [a, b(1), 2, 3, X]. ?- a(b(1),2,3,X)=..[a,b(X) _]. X = 1. ?- a(Y,2,3,X)=..[a,b(2) _]. Y = b(2). ?- Term=..[a,b,c(4),1,2,3,X]. Term = a(b, c(4), 1, 2, 3, X). </pre> <p>Таким образом с помощью данного оператора можно конструировать термы из элементов и «разбирать» их на элементы.</p>
call(Goal)	<p>«Запустить на выполнение» предикат хранящийся в Goal. Этот предикат может потребоваться, например, после конструирования предиката предыдущим оператором.</p> <p>Пример:</p> <pre> ?- assert(a(1)). ?- read(Y),X=..[a,Y],call(X) . : 1. Y = 1, X = a(1). ?- read(Y),X=..[a,Y],call(X) . : 23. false. </pre>

op(Precedence, Type, Name)	Объявление операторной формы с именем Name , приоритетом Precedence и типом Type для предиката с именем Name . Новые операторы можно вводить для увеличения удобства записи нужных предикатов (пример на с. 17)
-----------------------------------	---

Больше информации о встроенных предикатах можно найти во встроенной в SWI-Prolog англоязычной справке. Для доступа к ней из командного окна выбрать меню Help->Online Manual либо набрать запрос **?-help**. Поиск в справке (по функтору) можно осуществлять в нижнем поле поиска.

ЛИТЕРАТУРА

1. Хоггер К. Введение в логическое программирование. - М.: Мир, 1988. - 349 с.
2. Клоксин У., Меллиш К. Программирование на языке Пролог. - М.: Мир, 1987. - 336 с.
3. Стерлинг Л., Шапиро Э. Искусство программирования на языке Пролог. - М.: Мир, 1989, - 235 с.
4. Братко И. Программирование на языке Пролог для искусственного интеллекта. - М.: Мир, 1990. - 560 с.
5. Ковальски Р. Логика в решении проблем. - М.: Наука, 1990. - 280 с.
6. Малпас Дж. Реляционный язык Пролог и его применение. - М.: Наука, 1990.- 464 с.
7. Стобо Д.Ж. Язык программирования Пролог. - М.: Радио и связь, 1993. - 368 с.

1. ЗНАКОМСТВО С SWI/PROLOG. ЗАПУСК ПРОСТОЙ ПРОГРАММЫ	3
1.1. ГЛАВНОЕ МЕНЮ	3
1.2. ПЕРВАЯ ПРОГРАММА	3
1.3. ИСПОЛЬЗОВАНИЕ ВСТРОЕННЫХ ПРЕДИКАТОВ.....	8
1.4. ВЫПОЛНЕНИЕ И ТРАССИРОВКА ПРОГРАММЫ.....	9
2. СТРУКТУРЫ ДАННЫХ. СЛОЖНЫЕ УТВЕРЖДЕНИЯ В ПРОЛОГЕ	16
2.1. СТРУКТУРА ЯЗЫКА	16
2.2. СОСТАВНЫЕ ЦЕЛИ-ЗАПРОСЫ	16
2.3. СОСТАВНЫЕ ОБЪЕКТЫ	17
2.4. ОТРИЦАНИЕ В ПРОЛОГЕ	18
3. РЕКУРСИЯ В ПРОЛОГЕ	20
3.1. РЕКУРСИЯ	20
3.2. ПРОГРАММИРОВАНИЕ С НАКОПИТЕЛЯМИ	23
3.3. РЕКУРСИВНЫЕ ОБЪЕКТЫ.....	24
4. УПРАВЛЕНИЕ ЛОГИЧЕСКИМ ВЫВОДОМ.....	24
4.1. ОТСЕЧЕНИЕ(CUT).....	24
4.2. ОРГАНИЗАЦИЯ ЦИКЛА VAF-МЕТОД.....	26

4.3.	ОРГАНИЗАЦИЯ ЦИКЛА UDR-МЕТОД	27
4.4.	ИСПОЛЬЗОВАНИЕ НАДРЕЗОВ (SNIP).....	28
5.	СПИСКИ.....	30
6.	МОДИФИКАЦИЯ УТВЕРЖДЕНИЙ ПРОГРАММЫ. РАБОТА С БАЗОЙ ДАННЫХ.....	32
7.	ПРИЛОЖЕНИЕ 1. ТИПЫ ДАННЫХ SWI/PROLOG.....	36
7.1.	ПЕРЕМЕННЫЕ	36
7.2.	СИМВОЛЬНЫЕ КОНСТАНТЫ ИЛИ АТОМЫ	36
7.3.	ЦЕЛЫЕ ЧИСЛА.	37
7.4.	РАЦИОНАЛЬНЫЕ ЧИСЛА	37
7.5.	ЧИСЛА С ПЛАВАЮЩЕЙ ТОЧКОЙ	38
7.6.	СТРОКИ	38
7.7.	СТРУКТУРЫ	38
8.	ПРИЛОЖЕНИЕ 2. ВСТРОЕННЫЕ ПРЕДИКАТЫ И ОПЕРАТОРЫ.	39
	ЛИТЕРАТУРА	42