Функциональное и логическое программирование

Лекция 4

1.11 Функционалы

В Лиспе функции могут выступать в качестве аргументов (аргументом функции может быть определяющее функцию лямбда-выражение или имя другой функции). Такой аргумент называется *функциональным*, а функция, имеющая функциональный аргумент, называется *функционалом*.

1.11.1 Аппликативные (применяющие) функционалы

Применяющим функционалом называется функционал, который применяет функциональный аргумент к остальным параметрам.

(**APPLY** fn sp)

Вычисляет значение функционального аргумента (функции от n переменных) для фактических параметров, которые являются элементами списка.

Пример 1:

Написать функциональный предикат **ALL**, который возвращает t в том и только в том случае, если функциональный аргумент истинен для каждого элемента списка.

```
(defun all(p L)
  (cond
    ((null L) t)
    ((apply p (list(car L)))(all p (cdr L)))
    (t nil)
))
```

(all (lambda(x)(<= 0 x)) '(1 0 2 3 4 -1 0)) → nil

(**FUNCALL** fn v_1 v_2 … v_n)

Работает аналогично **APPLY**, но аргументы функционального аргумента (функции от n переменных) задаются не списком, а как аргументы **FUNCALL**, начиная со второго.

Пример 2***:***

Написать функцию сортировки списка методом вставки в виде функционала **SORT1**, у которого функциональный аргумент будет задавать порядок сортировки.

```
(defun sort1(p L)
  (cond
    ((null L) L)
    (t (add (car L) (sort1 p (cdr L)) p))
))
(defun add(x L p)
 (cond
  ((null L)(list x))
  ((funcall p x (car L))(cons x L))
  (t (cons (car L)(add x (cdr L) p)))
))
```

Можно использовать для сортировок в различных порядках:

(sort1 '< '(4 3 1 0 5 9 1)) → (0 1 1 3 4 5 9)

(sort1 '> '(4 3 1 0 5 9 1)) → (9 5 4 3 1 1 0)

(sort1 'string< '(h a f w r)) → (a f h r w)

1.11.2 Отображающие функционалы или MAP-функции

Отображающие функционалы с помощью функционального аргумента преобразуют список в новый список или порождают побочный эффект, связанный с этим списком. Такие функционалы начинаются на MAP.

(**MAPCAR** fn sp_1 sp_2 … sp_n)

Возвращает список, состоящий из результатов последовательного применения функционального аргумента (функции n переменных) к соответствующим элементам n списков. Число аргументов-списков должно быть равно числу аргументов функционального аргумента.

Пример 3:

Заменить в списке все элементы на пару (<элемент> *).

(mapcar (lambda(x) (list x '*)) '(1 s d 2)) → ((1 *) (S *) (D *) (2 *))

Пример 4:

Функция **SUM3** вычисляет сумму кубов элементов числового списка.

```
(defun sum3(L)
 (eval (cons '+ (mapcar '* L L L)))
)
```

или можно так:

```
(defun sum3(L)
 (apply '+(mapcar '* L L L))
)
```

(sum3 '(1 2 3)) → 36

(**MAPLIST** fn sp_1 sp_2 … sp_n)

Отображающий функционал **MAPLIST** действует подобно **MAPCAR**, но действия осуществляются не над элементами списков, а над последовательными хвостами этих списков, начиная с самих списков.

Пример 5:

(maplist 'reverse '(a s d f g)) → ((g f d s a) (g f d s) (g f d) (g f) (g))

(maplist (lambda(x)(eval(cons '+ x))) '(1 2 3 4 5)) → (15 14 12 9 5)

Объединяющие функционалы MAPCAN и MAPCON

Работа их аналогична соответственно **MAPCAR** и **MAPLIST**. Различие заключается в способе построения результирующего списка. Если функционалы **MAPCAR** и **MAPLIST** строят новый список из результатов применения функционального аргумента с помощью функции **LIST**, то функционалы **MAPCAN** и **MAPCON** для построения нового списка используют структуроразрушающую псевдофункцию **NCONC**, которая делает на внешнем уровне то же самое, что и функция **APPEND.** Функционалы **MAPCAN** и **MAPCON** удобно использовать в качестве фильтров для удаления нежелательных элементов из списка.

(MAPCAN fn sp_1 sp_2 … sp_n)

Пример 6:

Удалить из числового списка все элементы, кроме отрицательных.

```
(mapcan (lambda(x)
        (cond
          ((minusp x)(list x))
          (t nil)
       )) '(-3 4 5 0 -1)) → (-3 -1)
```

Сравним с работой mapcar:

```
mapcar (lambda(x)
        (cond
          ((minusp x)(list x))
          (t nil)
       )) '(-3 4 5 0 -1)) → ((-3) nil nil nil (-1))
```

(MAPCON fn sp_1 sp_2 … sp_n)

Пример 7:

Преобразовать одноуровневый список во множество.

```
(mapcon (lambda(L)
          (cond
            ((member (car L)(cdr L))nil)
             (t (list(car L)))
       )) '(1 2 3 1 4 1 2 3)) → (4 1 2 3)
```

Глава 2 Логическое программирование. Основы языка Пролог

Логическое программирование базируется на убеждении, что не человека следует обучать мышлению в терминах операций компьютера, а компьютер должен выполнять инструкции, свойственные человеку. В чистом виде логическое программирование предполагает, что инструкции не задаются, а сведения о задаче формулируются в виде логических аксиом. Такое множество аксиом является альтернативой обычной программе. Подобная программа может выполняться при постановке задачи, формализованной в виде логического утверждения, подлежащего доказательству (*целевого утверждения*).

Идея использования логики исчисления предикатов I порядка в качестве основы языка программирования возникла в 60-е годы, когда создавались многочисленные системы автоматического доказательства теорем и вопросно-ответные системы. В 1965 г. Робинсон предложил принцип резолюции, который в настоящее время лежит в основе большинства систем поиска логического вывода. В нашей стране была разработана система ПРИЗ, которая может доказать любую теорему из школьного учебника геометрии.

PROLOG (programming in logic) - 1972 г., Колмероэ, Марсельский университет.

Группа занималась проблемой автоматического перевода с одного языка на другой.

Основа PROLOG - исчисления предикатов I порядка и метод резолюций.

PROLOG - язык для описания данных и логики их обработки. Программа на Прологе не является таковой в классическом понимании, поскольку не содержит явных управляющих конструкций типа условных операторов, операторов цикла и т.д. Она представляет собой модель фрагмента предметной области, о котором идет речь в задаче.

Использование PROLOG:

- область автоматического доказательства теорем;
- построение экспертных систем;
- машинные игры с эвристиками (например, шахматы);
- автоматический перевод с одного языка на другой.

Реализации языка Пролог:

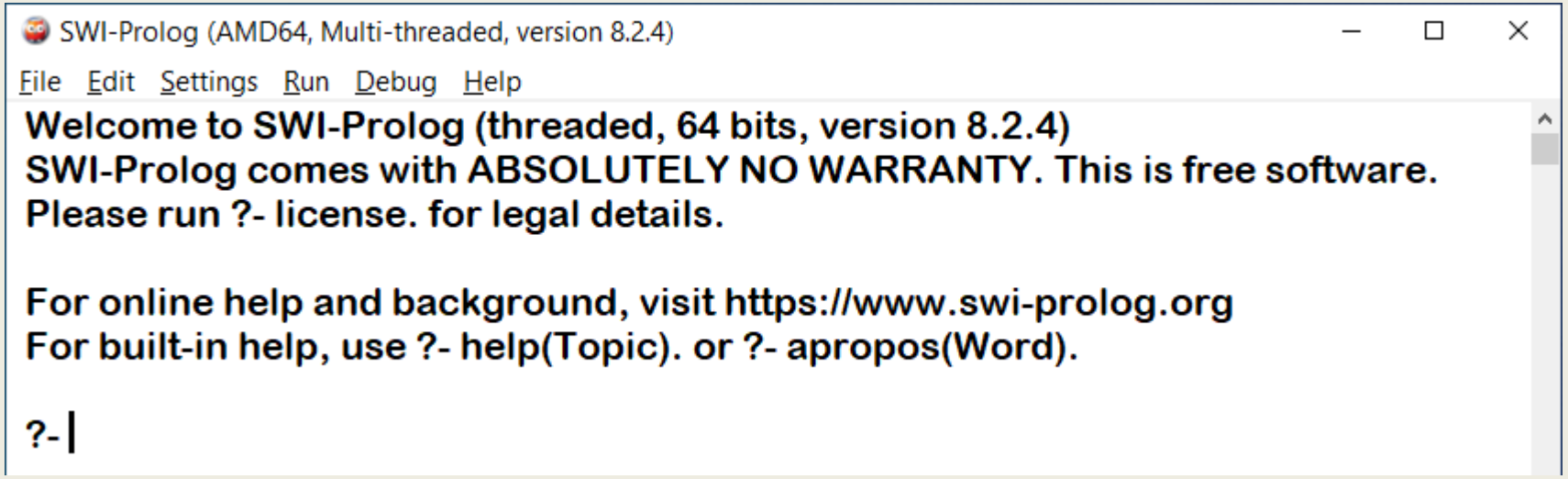
Wisdom Prolog, SWI Prolog, Turbo Prolog, Visual Prolog, Arity Prolog и т.д.

SWI-Prolog (SWI перевод с гол. социально-научная информатика) - 1987 г., Ян Вьелемакер, Амстердамский университет.

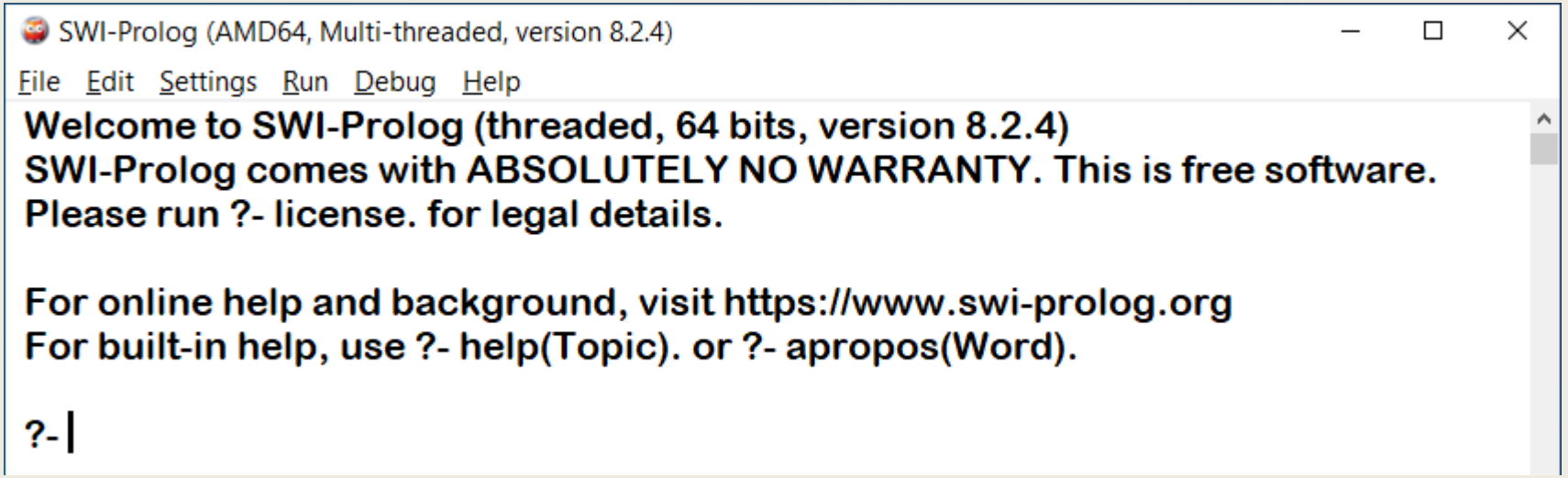
SWI-Prolog позволяет разрабатывать приложения любой направленности, включая Web-приложения и параллельные вычисления, но основным направлением использования является разработка экспертных систем, программ обработки естественного языка, обучающих программ, интеллектуальных игр и т.п. Это интерпретатор. Файлы, содержащие программы, написанные на языке SWI Prolog, имеют расширение pl.

Ссылка для скачивания.

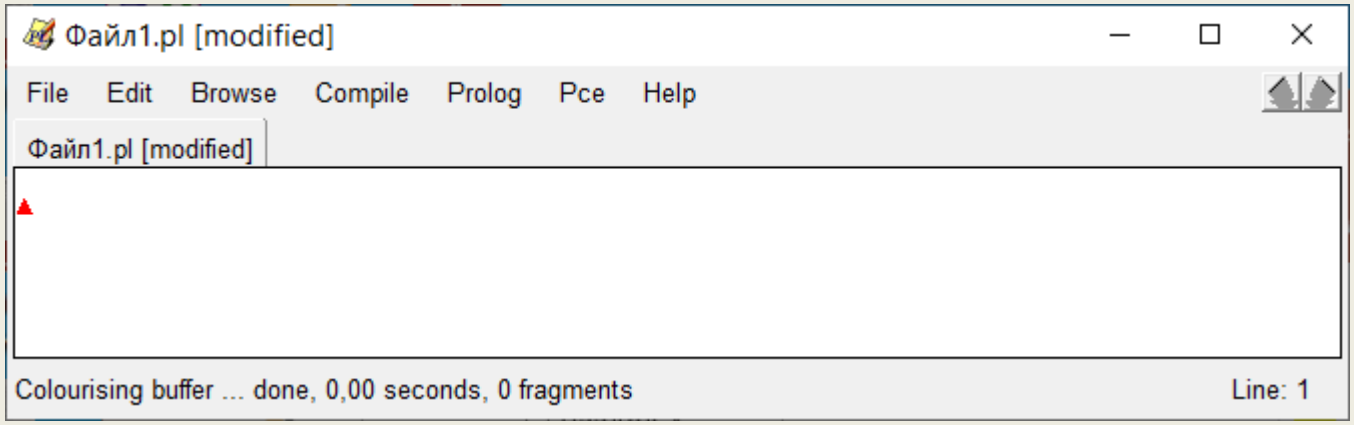
https://www.swi-prolog.org/Download.html



Загрузка существующего файла интерпретатору File-Consult. После загрузки можно задавать вопросы или утверждение для доказательства после знака вопроса.



Если создается новый файл, то можно открыть окно редактора File-New (откроется новое окно).

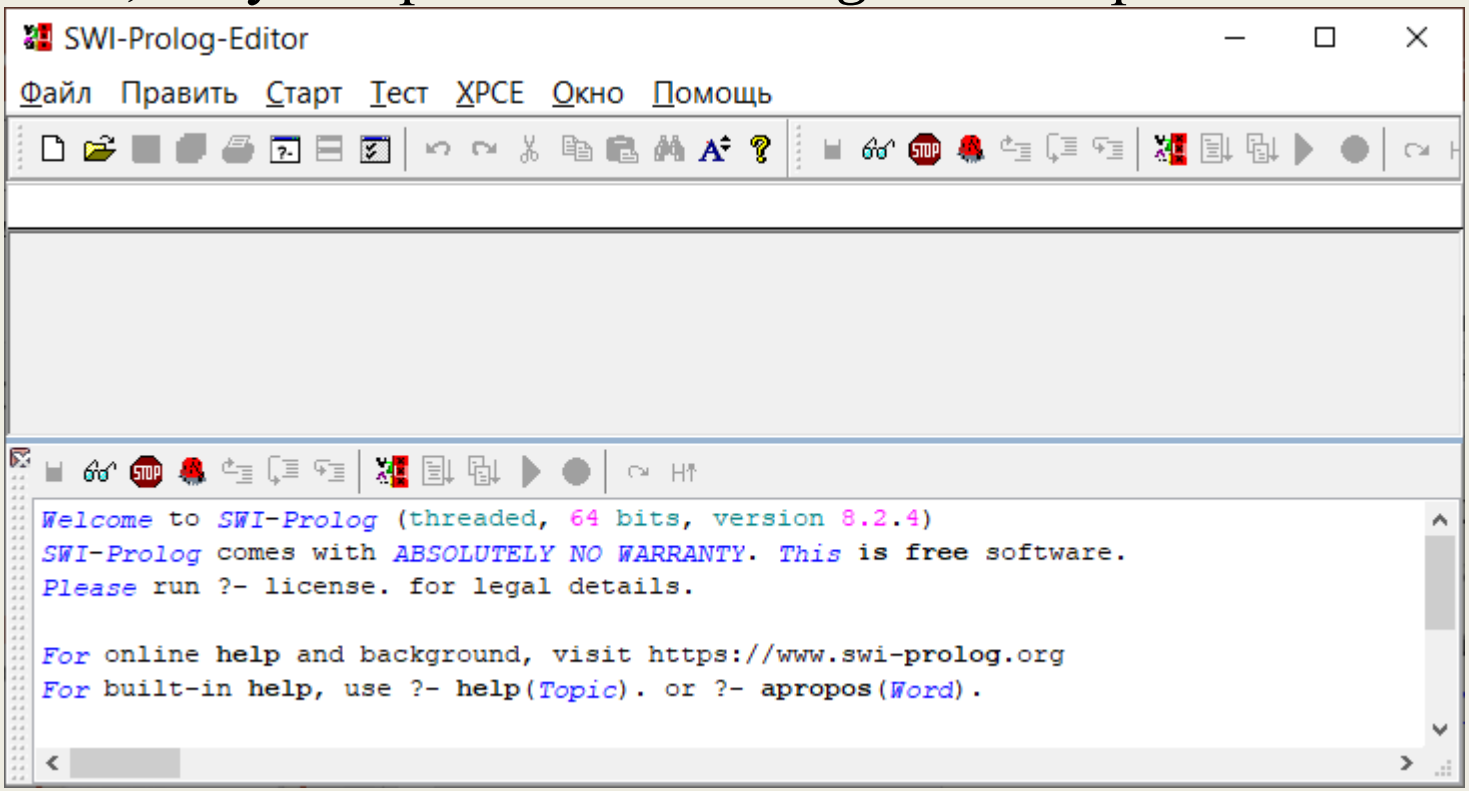


Загрузить файл интерпретатору Compile-Compile buffer.

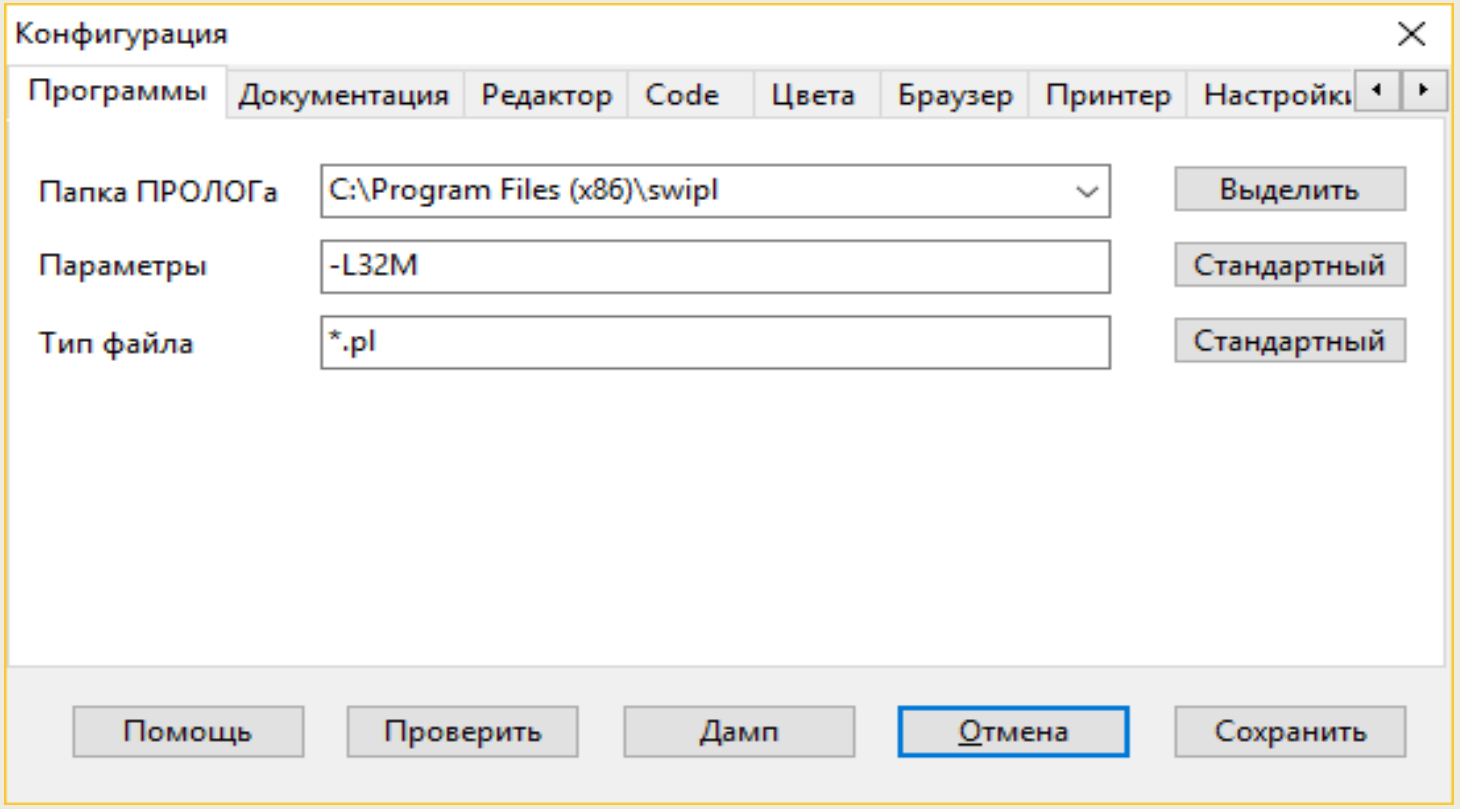
Вместо встроенного редактора удобно использовать SWI-Prolog Editor – среда программирования.

Ссылка для скачивания https://arbeitsplattform.bildung.hessen.de/fach/informatik/swiprolog/indexe.html.

Выбираем 32-битную или 64-битную версию в зависимости от того, какую версию SWI-Prolog скачали ранее.

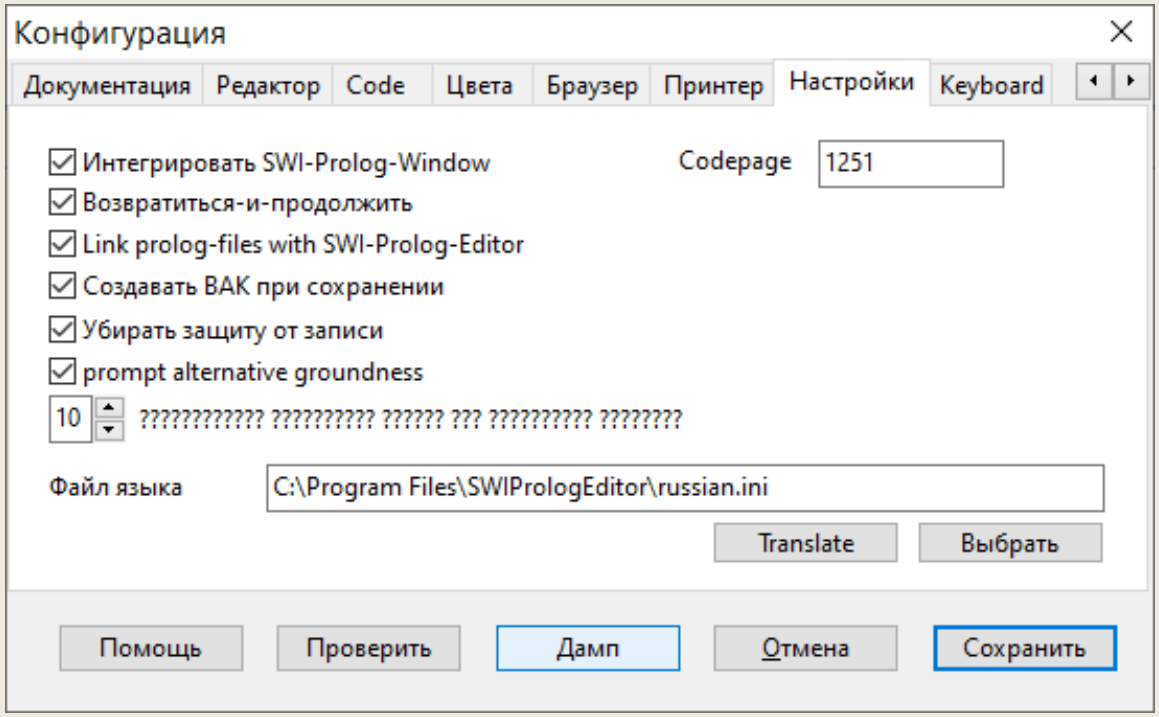


Если связывание SWI-Prolog Editor с SWI-Prolog не произошло автоматически, то необходимо указать путь к исполняемому файлу SWI-Prolog. Для этого выберите пункт главного меню Окно – Конфигурация и в открывшемся окне на вкладке Программы пропишите путь к папке bin, в которой находится исполняемый файл swipl-win.exe.

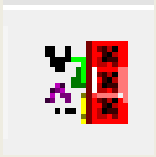


Необходимо выполнить настройку кодовой страницы для правильного сопоставления строковых констант, набранных русским алфавитом, между текстом программы в среде SWI-Prolog-Editor и языком SWI-Prolog.

Для этого выберите пункт главного меню Окно – Конфигурация и в открывшемся окне на вкладке Настройки установите номер кодовой страницы 1251.

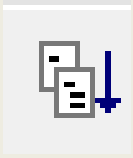


Значки на панели инструментов:



Перезапустить SWI-Prolog

Подать на вход интерпретатору содержимое текущего окна

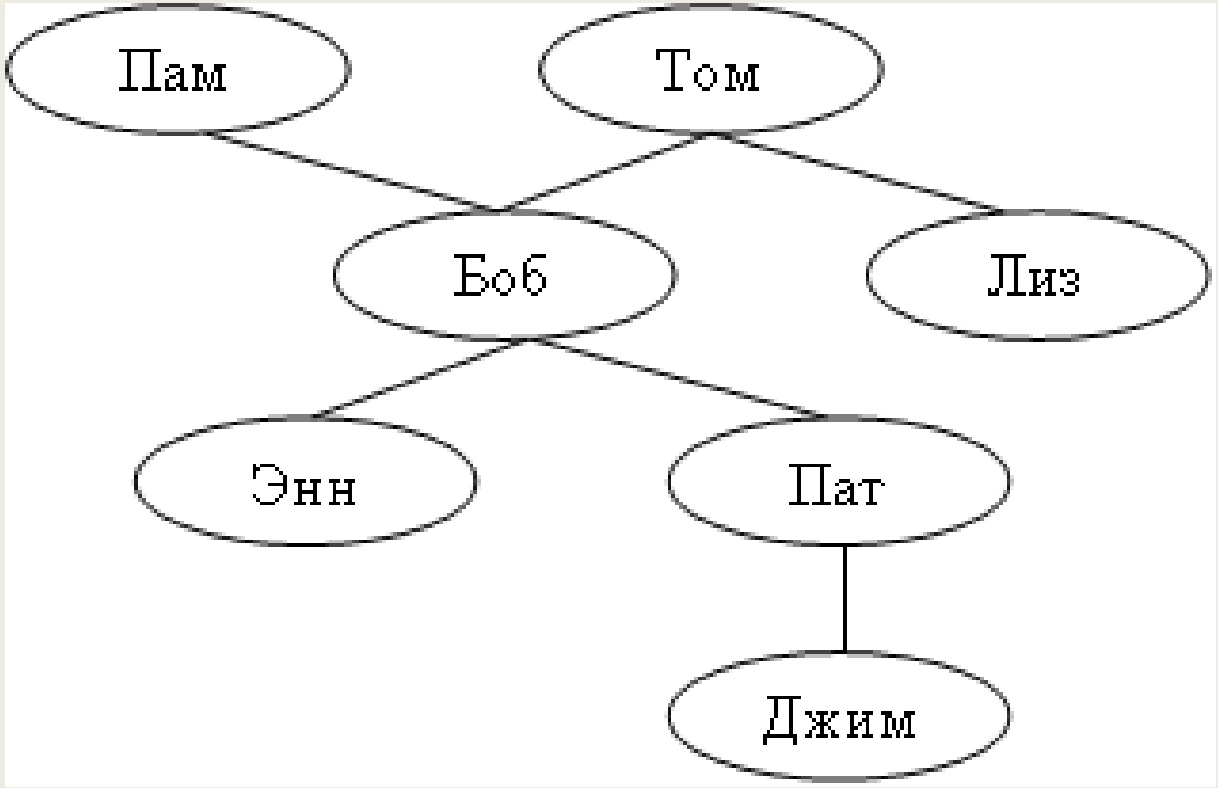


Подать на вход интерпретатору содержимое всех окон

Остановить работу программы

2.1 Факты и правила

Пример 1: Написать программу, описывающую следующее дерево семейных отношений:



Программа:

```
родитель(пам,боб).
родитель(том,боб).
родитель(том,лиз).
родитель(боб,энн).
родитель(боб,пат).
родитель(пат,джим).
```

Теперь можно задавать вопросы.

Вопросы к написанной программе:

1. Боб является родителем Пат?

?- родитель(боб,пат).

true.

2. Пат – ребенок Лиз?

?- родитель(пат,лиз).

false.

3. Кто родители Лиз?

?- родитель(X,лиз).

X = том.

4. Кто дети Энн?

?- родитель(энн,Х).

false.

5. Кто дети Боба? (заметим, что их детей - двое)

?- родитель(боб,Ч).

Ч = энн ;

Ч = пат.

После найденного первого решения Пролог ждет дальнейших указаний: продолжить поиск решений (тогда нажимаем ; или Enter в SWI-Prolog-Editor) или прекратить (тогда нажимаем . или a).

6. Есть ли дети у Пам?

?- родитель(пам,_).

true.

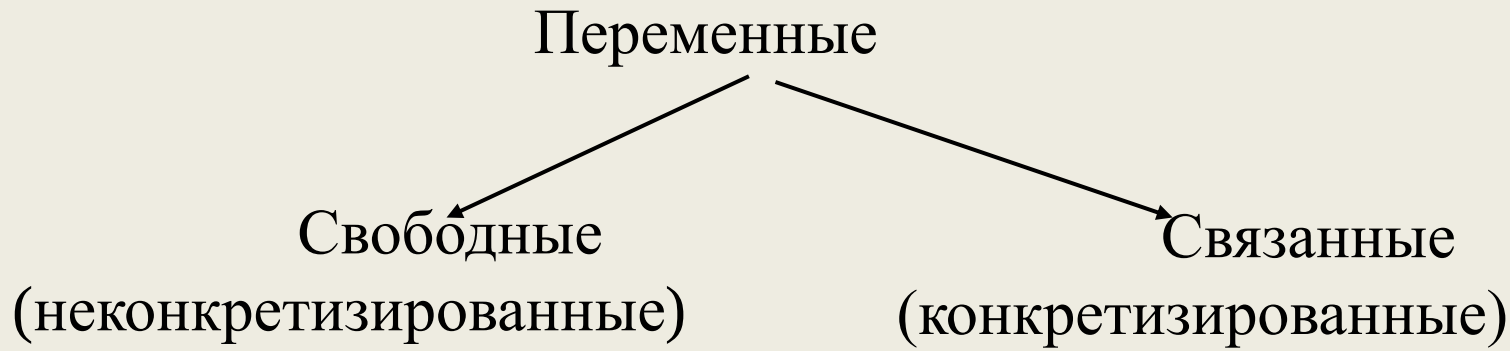
Вопросы могут быть простые и сложные (в качестве связки «и» при составлении сложного вопроса используется запятая).

Варианты ответов Пролог-системы на заданные вопросы:

- true
- false
- перечисление возможных значений переменных в ответе, при которых утверждение истинно. Если решение не единственно, то Пролог ожидает дальнейших указаний по продолжению поиска решений.

Программа на Прологе состоит из фактов и правил.

Факт – безусловное истинное утверждение
<имя предиката>(O_1,O_2,…,O_n).
O_i – конкретный объект (константа) или абстрактный объект (переменная).
В конце факта ставится точка!
Константы начинаются со строчной буквы.
Переменные начинаются с прописной буквы или подчерка.
Переменная обозначает объект, а не область памяти! Поэтому не можем менять ее значение (типа X=X+1).



Свободная переменная – переменная, которая еще не получила значения. Она не равняется ни нулю, ни пробелу; у нее вообще нет никакого значения. Такие переменные еще называют неконкретизированными.

Переменная, которая получила какое-то значение и оказалась связанной с определенным объектом, называется связанной. Если переменная была конкретизирована каким-то значением и ей сопоставлен некоторый объект, то эта переменная уже не может быть изменена в текущем предложении.

Область действия переменной – одно предложение! Связанная переменная не может изменяться внутри предложения.

Анонимная переменная начинается с символа подчеркивания и предписывает интерпретатору проигнорировать значение этой переменной.

Если в предложении несколько анонимных переменных, то все они отличаются друг от друга, несмотря на то, что записаны с использованием одного и того же символа.

Правило – утверждение, которое истинно при выполнении некоторых условий, оно позволяет описывать новые отношения. Правило имеет вид:

<голова правила > :– <тело правила>.

Головой правила является предикат, истинность которого следует установить.

Тело правила состоит из одного или нескольких предикатов, связанных логическими связками: конъюнкция (обозначается запятой), дизъюнкция (обозначается точкой с запятой) и отрицание (означается not или \+). Так же как в логических выражениях, порядок выполнения логических операций можно менять расстановкой скобок.

Можно в теле правила использовать разветвление вида:

(<условие>-><действие 1>;<действие 2>)

Пример 2: Добавим одноместное отношение мужчина (факты). Опишем новое двуместное отношение дед в виде правила.

мужчина(том).

мужчина(боб).

мужчина(джим).

дед(X,Y):-мужчина(X),родитель(X,Z),родитель(Z,Y).

Вопрос: Кто дед Джима?

?- дед(Ч,джим).

Ч = боб ;

false.

Вопрос: Кто внуки Тома?

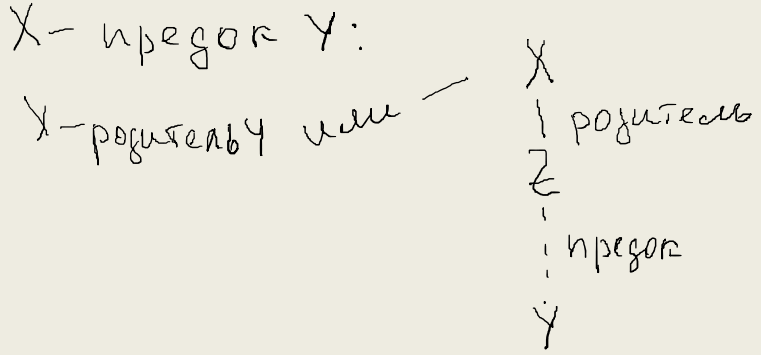
?- дед(том,X).

X = энн ;

X = пат ;

false.

Пример 3: Опишем новое двуместное отношение предок.



предок(X,Y):-родитель(X,Y).

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

или

предок(X,Y):-родитель(X,Y);

родитель(X,Z),предок(Z,Y).

Вопрос: Кто предок Джима?

?- предок(X,джим).

X = пат ;

X = пам ;

X = том ;

X = боб ;

false.

2.2 Поиск решений Пролог-системой

Вопрос к системе – это последовательность, состоящая из одной или нескольких целей.

Для ответа на поставленный вопрос Пролог-система должна *достичь всех целей*, т.е. показать, что утверждения вопроса истинны в предположении, что все отношения программы истинны.

Если в вопросе имеются переменные, то система должна найти конкретные объекты, которые, будучи подставлены вместо переменных, обеспечат достижение цели. Если система не в состоянии вывести цель из имеющихся фактов и правил, то ответ должен быть отрицательный.

Факты и правила в программе соответствуют аксиомам, вопрос – теореме.

При поиске ответа на поставленный вопрос находится факт или правило для содержащегося в вопросе предиката и выполняется операция сопоставления (унификации) объектов предиката.

Операция унификации объектов успешна:

- сопоставляются две одинаковые константы;
- сопоставляется свободная переменная с константой (при этом свободная переменная становится означенной);
- сопоставляется связанная переменная с константой, равной значению переменной;
- сопоставляется свободная переменная с другой свободной переменной (переменные не получают значений, но становятся сцепленными, т.е. когда одна из них получит значение, то и вторая получит это же значение).

После успешного сопоставления все переменные получают значения и становятся связанными, а предикат считается успешно выполненным (если сопоставление выполнялось с фактом) или заменяется на тело правила (если сопоставление выполнялось с головой правила). Связанные переменные освобождаются, если цель достигнута или сопоставление неуспешно.

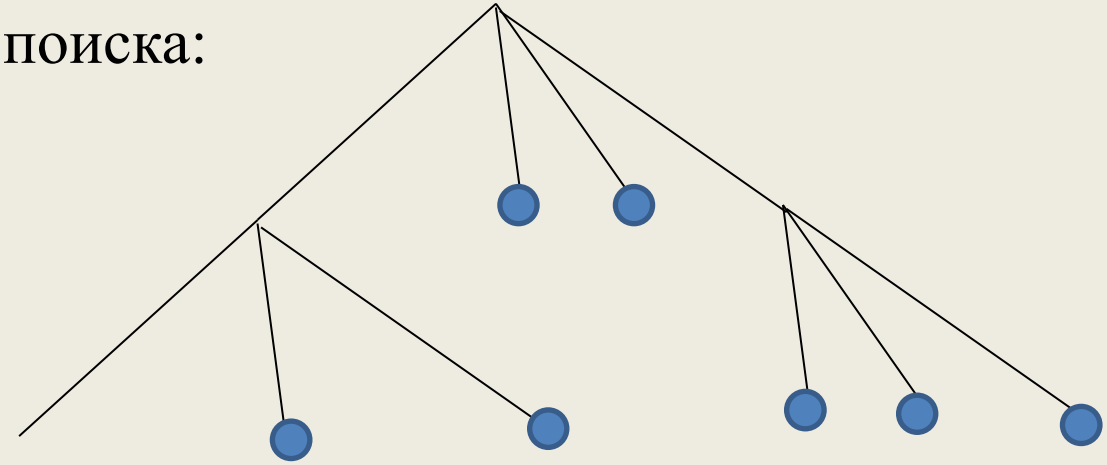
Процесс унификации похож на использование оператора =.

A=B может интерпретироваться как присваивание слева направо, справа налево, как сравнение.

Для достижения цели используется механизм отката.

При вычислении цели выполняется сопоставление с фактами и головами правил. Сопоставления выполняются слева направо.

Дерево поиска:



Выделены точки отката, которые Пролог запоминает для поиска альтернативных путей решения.

Если цель была неуспешна, то происходит откат к ближайшему указателю отката.

Если цель достигнута, но использовались не все указатели отката, то будет продолжен поиск решений.

Трассировка

Включение трассировки: trace.

Отказ от трассировки: notrace.

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

Слова, появляющиеся в окне трассировки:

Call Далее указывается текущая цель

Exit Указывается цель, которая успешна.

Redo Возврат в отмеченную точку возврата для поиска альтернативного решения.

Fail Указанная цель не была достигнута.

В круглых скобках указывается глубина в дереве поиска, нумерация начинается с 10.

?- предок(том,энн).

Пример:
Трассировка вопроса к Пролог-системе: предок(том,энн).
?- trace.

[trace] ?- предок(том,энн).
   Call: (10) предок(том, энн) ? creep
   Call: (11) родитель(том, энн) ? creep
   Fail: (11) родитель(том, энн) ? creep
   Redo: (10) предок(том, энн) ? creep
   Call: (11) родитель(том, _29946) ? creep
   Exit: (11) родитель(том, боб) ? creep
   Call: (11) предок(боб, энн) ? creep
   Call: (12) родитель(боб, энн) ? creep
   Exit: (12) родитель(боб, энн) ? creep
   Exit: (11) предок(боб, энн) ? creep
   Exit: (10) предок(том, энн) ? creep