Севастопольский государственный университет
кафедра Информационные системы

Курс лекций по дисциплине "МЕТОДЫ И СИСТЕМЫ ИСКУССТВЕННОГО ИНТЕЛЛЕКТА" (МиСИИ)

Лектор: Бондарев Владимир Николаевич

Лекция 15

Общая характеристика языка Пролог. Основные понятия. Арифметические выражения. Списки и рекурсия.

В.Бондарев

Общая характеристика языка

Язык Пролог был разработан в начале семидесятых годов в университете Марселя (Франция) под руководством А. Колмероэ. Само название языка определяет его суть – **язык логического программирования.**

Существуют различные версии языка Пролог: Quintus-Prolog, Micro-Prolog, Arity Prolog, Visual Prolog, **SWI-Prolog** и др. Первая эффективная реализация Пролога была создана в Эдинбургском университете в 1977. Синтаксис данной версии стал фактическим стандартом.

Программа на Прологе представляет собой **декларативное описание отношений** некоторой предметной области. Выполнение программы заключается в постановке вопросов, относящихся к предметной области, и автоматическом поиске ответов на вопросы с помощью встроенных механизмов логического вывода.

Основные понятия языка

Программа на Прологе представляет совокупность ***утверждений*** предметной области, записанных с использованием предикатов.

В математике ***предикатом*** называется неоднородная двузначная ***логическая*** функция от любого числа аргументов. Ее записывают в виде ***P(x1,x2,..,xn)*** и называют ***n-местным предикатом***. Здесь аргументы ***x1,x2,..,xn*** — принадлежат одному и тому же или различным множествам, представляющим объекты предметной области. Функциональную букву ***P*** называют ***предикатной буквой***.

В общем случае *n*-местный предикат ***P(x1,x2,..,xn)*** задает отношение между аргументами ***x1,x2,..,xn*** , которое означает, что “***x1,x2,..,xn*** находятся между собой в отношении ***P***”. Например, полагая, что предикатная буква ***P*** представляет отношение **«произведение»** , можно трехместному предикату ***P(x1,x2,x3)*** дать следующую интерпретацию: **“*x1* есть произведение *x2* на *x3*”.**

Основные понятия языка

В языке Пролог при записи утверждений вместо предикатных букв используют имена, которые записывают строчными буквами. ***В конце утверждения ставится точка.***

Существуют два типа утверждений Пролога: **факты и правила.**

Факт представляет собой истинное утверждение, которое представляется в виде предиката. Например, **“Игорь — отец Святослава”** записывается на Прологе в следующей форме

отец(´Игорь´,´Святослав´).

Здесь **отец** — это имя предиката, а аргументы **'Игорь'** и **'Святослав'** представляют собой символьные константы.

Символьные константы должны начинаться со **строчной буквы** или заключаться в одинарные кавычки.

Переменные начинаются с **прописной буквы** или символа подчеркивания. Так, **X,Y, T34, _a, _132** — это имена переменных.

Основные понятия языка

Правило — представляет собой утверждение, которое истинно при определенных условиях. Например, утверждение **“X — дед Y”** верно при условии, что **X** — это отец **Z**, где **Z** — это один их родителей **Y**. На Прологе это правило запишется в виде

дед(X,Y): – отец (X,Z), родитель(Z,Y).

Правило состоит из ***заголовка (головы)*** и ***тела.*** Заголовок и тело соединяются знаком “:– ”, соответствующим импликации “ ←”. **Запятая,** записанная в теле правила, соответствует логической связке “и”. **Точка с запятой** – связке “или”.

Приведенное выше правило интерпретируется следующим образом: “отношение **дед(X,Y)** верно, если верны отношения **отец(X,Z)** и **родитель(Z,Y)**”, которые называют ***условиями.***

В теле правила можно использовать логическую связку “или”:

родитель(X,Y): – отец(X,Y); мама(X,Y).

Иными словами, **X** является родителем **Y**, если **X** — это отец или мама **Y**.

Основные понятия языка

В Прологе используют **однострочные и многострочные комментарии**. Однострочный комментарий начинается со знака “**%**”, а многострочный ограничивается символами: “**/*…*/**”.

Совокупность фактов и правил в Прологе образуют ***базу данных***.
Рассмотрим пример базы данных, содержащей отношения родства:

```
% Факты
отец('Иван','Сергей' ).          % Иван — отец  Сергея
отец( 'Иван','Анна' ).           % Иван — отец   Анны
мама('Мария','Сергей').          % Мария — мать Сергея
% Правило
родители(O,M,P ): – отец(O,P), мама(M,P).
```

Когда база данных создана и загружена в память Пролог-системы, к ней можно обращаться с запросами. Для этого необходимо в окне консоли Пролог-системы ввести ***целевое утверждение***:

```
?– отец('Иван','Сергей').
```

Основные понятия языка

Это обращение инициирует процесс поиска (доказательства) соответствующих утверждений в базе данных. Система начинает искать факты, ***сопоставимые*** с фактом целевого утверждения. При этом выполняется ***процедура унификации***. **Два факта сопоставимы**, если совпадают имена предикатов, и соответствующие аргументы попарно сопоставимы. В рассматриваемом случае Пролог находит в базе данных факт **отец('Иван','Сергей'),** сопоставимый с целевым утверждением. Обнаружив этот факт, Пролог-система ответит **Yes** (да).

На вопрос

? – отец('Иван',P). % Кто дети Ивана?

Пролог-система выдаст ответ **P='Сергей'**. Если ввести с клавиатуры “**;**”, то Пролог-система продолжит поиск, и получит второе значение **P='Анна'**. Если еще раз ввести точку с запятой, то система вернет значение **No**.

Основные понятия языка

Можно задать и более сложный вопрос: “Кто родители Сергея?”:

? – родители(О,М,'Сергей').

В результате получим

О='Иван';

М='Мария';

No.

В данном случае Пролог-система воспользовалась правилом и попытку доказательства целевого утверждения

родители(О,М,'Сергей')

заменила на доказательство двух новых подцелей **отец(О,'Сергей')** и **мама(М,'Сергей').** В результате было установлено, что первая подцель достижима, если **О='Иван'**, вторая подцель достижима, если **М='Мария'**.

Основные понятия языка

В общей форме **правило** можно записать следующим образом

P: – P1, P2, …, Pn,

где **P,P1,P2, …,Pn** – **элементарные (атомарные)** формулы.

Атомарная формула имеет вид

P(t1, t2,…, tn),

где **P** – предикатный символ; **t1, t2,…, tn** – множество **термов**, являющихся аргументами атомарной формулы.

Терм может быть ***константой***, ***переменной*** или ***составным*** термом (структурой). ***Константы*** языка Пролог подразделяются на ***числа и атомы***. Примеры числовых констант: **-1, 176.5, 0.29e–3**. ***Атомы*** языка Пролог представляют **символьные константы**, которые или начинаются строчной буквой, или заключаются в одинарные кавычки, или состоят из специальных символов. Примеры атомов: **сергей ; 'Сергей‘ ; x400 ; <--> ; ==>**

Основные понятия языка

Областью действия переменной является утверждение. В пределах утверждения одно и то же имя принадлежит одной переменной. Два утверждения могут использовать одно имя переменной различным образом.

Переменные, которым присвоены значения, называются ***конкретизированными.***

Существуют также ***анонимные*** переменные, то есть переменные без имени. Анонимные переменные обозначаются символом подчеркивания. Например, **отец('Иван', _).** Это означает, что нас не интересует, какие подстановки будут выполняться в анонимную переменную. Каждая анонимная переменная уникальна, т.е. отличается от всех других анонимных переменных в утверждении.

Основные понятия языка

Структура (или составной терм) является объектом, состоящим из нескольких компонент. Структура состоит из атома, который называется ***функтором***, и последовательности термов, которые рассматриваются в качестве компонент структуры. Компоненты структуры разделяются запятыми и заключаются в круглые скобки. Функтор записывается перед скобками. Например, следующая структура имеет функтор **f** и три компоненты:

f(T1,T2,T3).

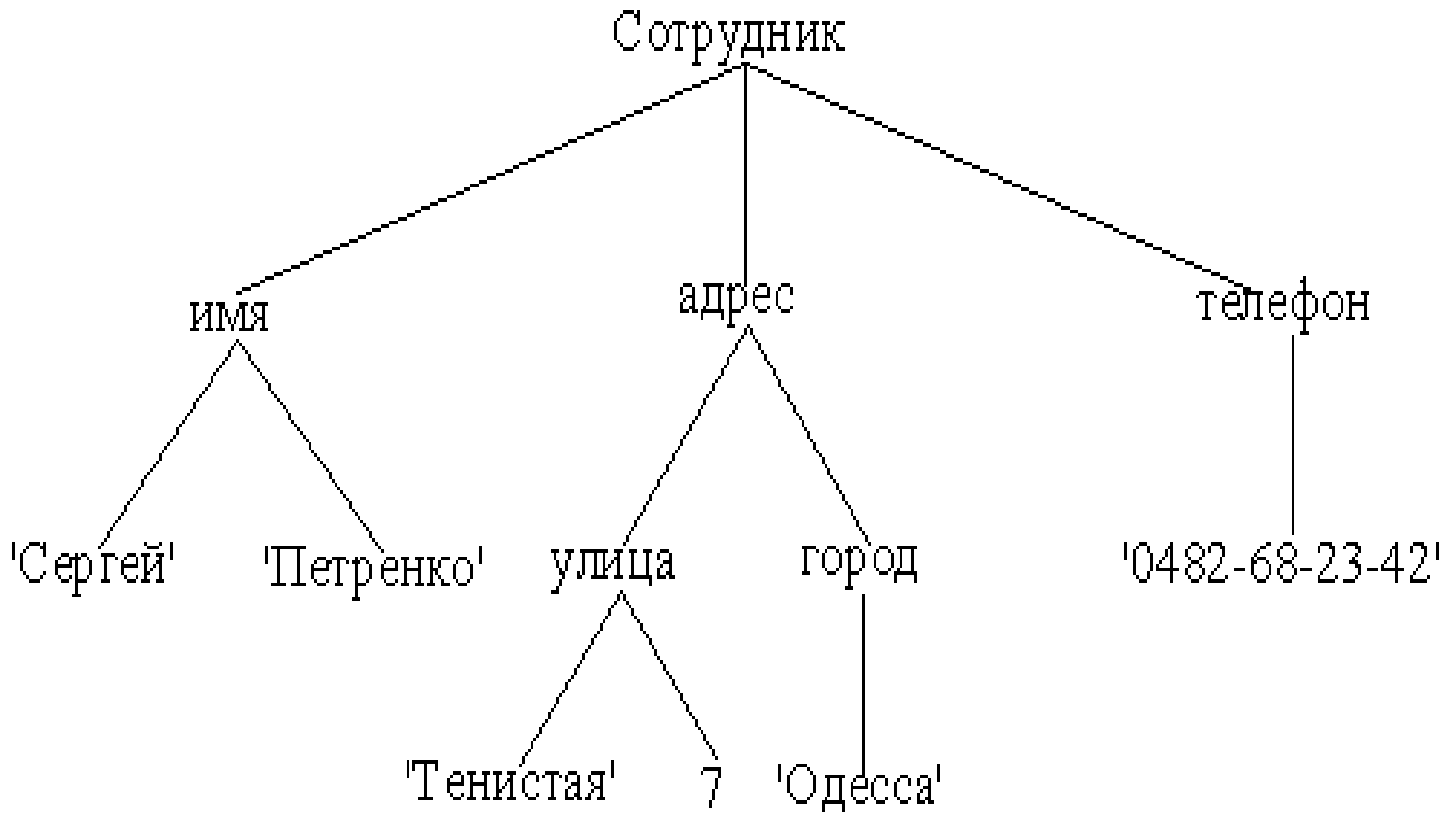
Число компонент структуры называется ***арностью.*** Так, структура **f** имеет арность 3.

Введем **иерархическую структуру,** представляющую информацию о некотором сотруднике:

сотрудник(имя('Сергей','Петренко'),
адрес(улица('Тенистая',7),город('Одесса')),
телефон('0482-68-23-42')).

Основные понятия языка

Данную структуру можно представить в виде дерева. Корень дерева называется ***главным функтором*** структуры.



Основные понятия языка

Одной из основных операций, выполняемых над термами в ходе доказательства целевого утверждения, является **сопоставление.** Два терма сопоставляются по следующим правилам:

а) если термы **X** и **Y** – константы, то они сопоставимы, только когда одинаковы;

б) если терм **X** представлен константой или структурой, а терм **Y** – не конкретизированной переменной, то термы **X** и **Y** сопоставимы, и **Y** принимает значение **X**;

в) если термы **X** и **Y** – структуры, то они сопоставимы, когда у них совпадают главные функторы и арность, а также сопоставимы соответствующие компоненты структуры.

Возможность сопоставления двух термов проверяется с помощью оператора “=”. Так, сопоставление

? – отец('Иван',P) = отец(O,'Сергей').

закончится успехом при **O='Иван'** и **P='Сергей'**.

Основные понятия языка

При рассмотрении пролог-программы полезно выделять два уровня ее смысла: **декларативный и процедурный**. ***Декларативный смысл*** пролог-программы связан с отношениями, объявленными (декларируемыми) в программе, он определяет, достижимо ли целевое утверждение, и при каких значениях переменных оно будет верным. ***Процедурный смысл*** определяет, как пролог-система обрабатывает отношения пролог-программы, каким образом пролог-система отвечает на вопросы

Поиск решения в пролог-системах протекает в автоматическом режиме, использующем принцип резолюции и принцип возврата к альтернативным вариантам возможных решений.

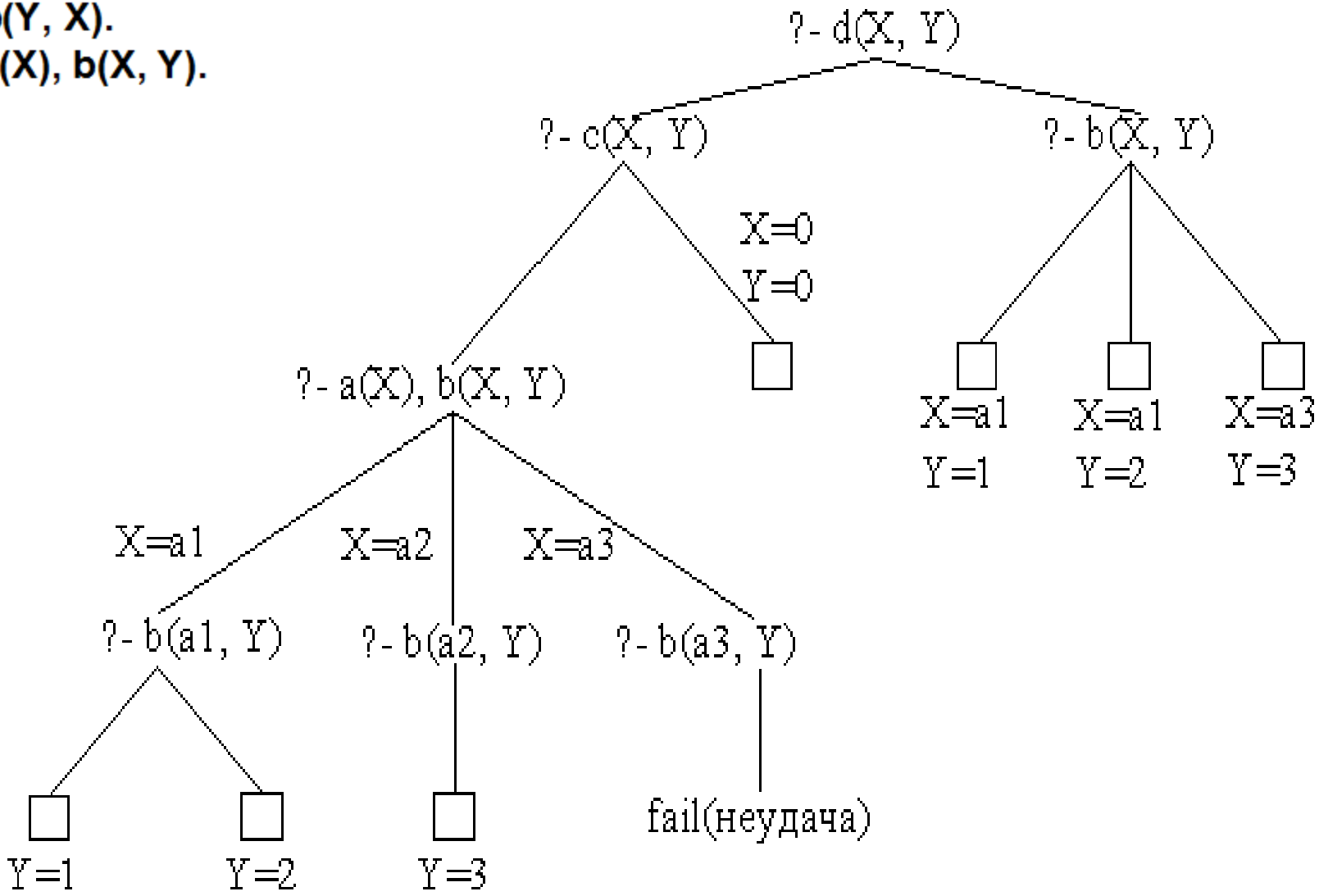
Для более точного понимания процедурного смысла пролог-программы строят **деревья вывода.**

Основные понятия языка

```
d(X,Y): – c(X, Y).          /*1*/
d(X,Y): – b(Y, X).          /*2*/
c(X,Y): – a(X), b(X, Y).    /*3*/
c(0,0).                     /*4*/
a(a1).                      /*5*/
a(a2).                      /*6*/
a(a3).                      /*7*/
b(a1,1).                    /*8*/
b(a1,2).                    /*9*/
b(a2,3).                    /*10*/
? – d(X,Y).                 /*11*/
```

Дерево вывода

```
d(X,Y): – c(X, Y).
d(X,Y): – b(Y, X).
c(X,Y): – a(X), b(X, Y).
c(0,0).
a(a1).
a(a2).
a(a3).
b(a1,1).
b(a1,2).
b(a2,3).
? – d(X,Y).
```



Основные понятия языка

Каждая ветвь **дерева вывода** описывает один шаг программы, а узел, находящийся в конце ветви, определяет то целевое утверждение, которое должно быть доказано. Корень дерева содержит исходное целевое утверждение. В конце пути, представляющем успешно завершенную последовательность выполнения, фигурирует **пустое условие**, обозначаемое на рисунке квадратом. Существуют такие последовательности шагов выполнения, которые заканчиваются неудачей (**fai**l).

Ввод команды “**;**” заставит пролог-систему выполнить ***возврат*** к точке выбора, в которой остались нереализованные варианты выполнения. При возврате происходит **автоматическое стирание подстановок**, которые были сделаны в переменные после соответствующей точки выбора.

Неудача (fail) автоматически заставляет пролог-систему осуществить возврат к возможным точкам выбора вариантов выполнения тех или иных условий.

Арифметические выражения

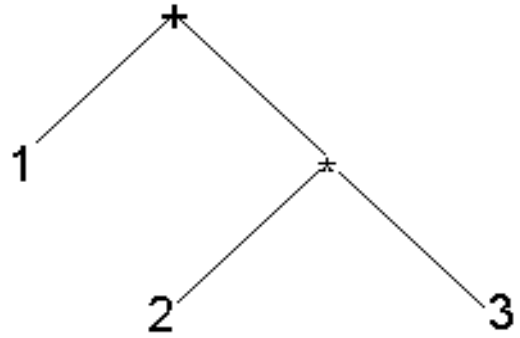
Язык Пролог предназначен, главным образом, для программирования задач символьной обработки информации. Тем не менее, в любую пролог систему включаются все обычные **арифметические операторы**: + (сложение), - (вычитание), * (умножение), / (деление), **mod** (остаток от деления целых чисел), **div** (целочисленное деление).

В Пролог входят также все необходимые **операторы сравнения чисел**: < (меньше), > (больше), >= (больше или равно), =< (меньше или равно), =\= (не равно), =:= (равенство целочисленных выражений).

Арифметические выражения в Прологе строят из арифметических операторов, числовых переменных, констант и скобок. При этом арифметические выражения представляют собой структуры.

Арифметические выражения

Например, выражение **1+2*3** представляется структурой



Если пролог-системе задать вопрос
? – X=1+2*3.
то в результате получим **X=1+2*3**, а не **X=7**.

Объясняется это тем, что оператор “=” выполняет сопоставление и переменной X ставит в соответствие структуру, изображенную на рисунке. При этом арифметическое выражение не вычисляется. Для того чтобы заставить пролог-систему выполнить необходимые вычисления, следует применить специальный оператор **is**. Например,

? – X is 1+2*3.

В этом случае ответ будет **X=7**.

Арифметические выражения

Оператор is определен как инфиксный оператор, который в общем виде записывается так:

X is Y.

Здесь **X** – или число, или не конкретизированная переменная, а **Y** – арифметическое выражение. Если **X** – число, то вычисляется значение выражения **Y**, которое сравнивается с **X**. При равенстве значений **X** и **Y** попытка доказательства **X is Y** заканчивается успехом. Если **X** – не конкретизированная переменная, то вычисленное значение выражения **Y** приписывается переменной **X**.

```
? – 3 is 1+2.
Yes
? – X is 7-2.
X=5
? – X=1, X is X+1.
No
```

Арифметические выражения

Определим функцию двух переменных $f(x, y)=x+y+1$:

f(X,Y,Z):- Z is X+Y+1.

? – f(1, 2, Z).

Z=4.

? – f(X,2,4).

завершится неудачей, так как оператор **is** “работает” только в направлении **справа налево**.

Точно так же, как и оператор **is**, **операторы сравнения** вычисляют свои аргументы перед выполнением:

? – 5+3-1>2+1.

Yes

В большинстве современных реализаций языка Пролог имеются встроенные предикаты для **вычисления элементарных функций**: возведение в квадрат (**sqr**), логарифм (**ln**), синус (**sin**), косинус (**cos**), абсолютное значение (**abs**) и др.

Списки и рекурсия

Список – это структура данных, составленная их произвольного числа элементов. Элементы списка отделяются друг от друга запятыми и заключаются в квадратные скобки:

[a, b, c, d].

Для представления списка в виде структуры данных, состоящей из **головы и хвоста**, в Прологе широко используется еще одно обозначение, в котором голова и хвост списка отделяются вертикальной чертой. Например, в записи **[H|T]** переменная **H** – представляет голову списка, а переменная **T** – хвост. Применив символ "|", список **[a, b, c, d]** можно представить следующими различными способами:

[a,b,c,d]=[a| [b,c,d]]=[a, b| [c,d]]==[a, b, c| [d]]=[a, b, c, d| [ ]].

Здесь пара квадратных скобок **[ ]** обозначает пустой список.

? – [x1, x2, x3 | x4]=[1, 2, 3, 4].

x1=1, x2=2, x3=3, x4=[4].

Списки и рекурсия

Над списками часто выполняют следующие **операции**: добавление элемента в список, удаление элемента из списка, объединение списков, поиск элемента в списке.

Добавление элемента в список:

добавить (X, L, [X|L]).

Здесь **X** – добавляемый элемент; **L** – список, в который добавляется элемент; **[X|L]** – результирующий список. Таким образом, элемент **X** добавляется в начало списка **L**.

Представление списков в виде головы и хвоста, где хвост, в свою очередь, тоже список, является рекурсивным. Поэтому обработка списков часто выполняется с помощью **рекурсивных предикатов**.

Рекурсивными называют предикаты, в определениях которых содержатся ссылки на самих себя.

Списки и рекурсия

Предикат, проверяющей **вхождение элемента H** в список:

member(H,[H|T]).

member(H,[X|T]):-member(H,T).

С помощью факта задается истинное утверждение о том, что элемент **H** и список **[H|T],** головной элемент которого есть **H**, находятся в отношении **member.** Правило означает, что элемент **H** и список **[X|T]** будут находиться в отношении **member** , если указанное отношение имеет место между элементом **H** и хвостом списка **T**. Иными словами, элемент **H** содержится в списке **[X|T]**, если он входит в хвост **T** этого списка.

Пусть требуется написать программу, выполняющую **соединение двух списков** и возвращающую в качестве результата третий список. Определим для этого предикат (отношение) **append(X,Y,Z),** где **X** и **Y** – исходные списки, а **Z**– результирующий список.

Списки и рекурсия

При описании отношения **append** необходимо учесть два случая:

1) если **X** представляет пустой список, то второй и третий аргумент (т. е. **Y** и **Z**) отношения представляют собой один и тот же список, что выражается в виде факта **append([ ],L,L);**
2) если **X** не пустой список, то он имеет голову и хвост и может быть записан в виде **[H|T];** в результате соединения такого списка со списком **Y**, получим новый список **[H|W],** где между хвостом **T** первого списка, списком **Y** и хвостом **W** результирующего списка должно существовать отношение **append(T,Y,W).** Это записывается в виде правила:

append([H|T],Y,[H|W]):-append(T,Y,W).

Иными словами, списки **[H|T], Y**, и **[H|W]** находятся в отношении **append,** если в этом же отношении находятся списки **T,Y и W**.

Пролог-программа, решающая поставленную задачу:

```
append([ ],L,L).
append([H|T],Y,[H|W]):-append(T,Y,W).
```

Списки и рекурсия

В общем случае все такие определения строятся по следующей схеме:

предикат(…[ ]…).
предикат(…[Голова|Хвост]…): –
обработка(Голова),
предикат(…[Хвост]…).

Рекурсивные вызовы прекратятся, когда хвост списка окажется пустым списком. В приведенном определении первое утверждение определяет условие выхода из рекурсии, а второе утверждение – правило, предусматривающее при каждом вызове обработку очередного элемента списка и рекурсивный вызов определяемого предиката, аргументом которого является хвост списка.

Списки и рекурсия

Определим, например, предикат **реверс(X,Y)**, где список **Y** представляет **инверсную копию списка X**:

```
реверс([ ], [ ]).                          %1
реверс([H|T],Y): – реверс(T,Ys),           %2
        append(Ys,[H],Y).
```

Для того чтобы выполнить инверсию списка **[H|T]** , необходимо выполнить инверсию хвоста **T** и получить список **Ys**, а затем добавить голову **H** в конец списка **Ys** и получить результирующий список **Y**.

Удаление элемента **X** из списка **L** можно представить в виде предиката **удалить(X, L, L1),** где **L1** – результирующий список. Если **X** является головой списка **L**, то результирующий список **L1** – это хвост списка **L**. Если **X** находится в хвосте списка **L**, то для удаления **X** необходимо рекурсивно вызвать предикат **удалить**, подставив в качестве второго аргумента хвост списка **L**.

Списки и рекурсия

удалить(X, [X|T], T).
удалить(X, [H|T], [H|T1]): –
удалить(X, T, T1).

Предикат **удалить** можно использовать и в **обратном порядке**.
? – удалить (1, L, ['a', 'b']).
L=[1, 'a', 'b'];
L=['a', 1, 'b'];
L=['a', 'b', 1].

В данном случае **выполняется вставка** элемента **1** в произвольные позиции списка. В итоге получаются различные списки **L**, исключив из которых элемент **1**, получим список **['a', 'b'].**