Логическое программирование на Visual Prolog
Списки

Оглавление

Введение

Список – это упорядоченная совокупность произвольного числа элементов. Порядок расположения элементов в последовательности является существенным. Элементами списка могут быть любые термы – константы, переменные, списки, структуры, которые включают другие списки, и т.п. Списки позволяют представить практически любой тип структуры данных, который может потребоваться при обработке символьной информации.

Одна из форм представления списков на Прологе – это скобочная форма записи списка. Она представляет собой заключенную в квадратные скобки последовательность элементов списка, разделенных запятыми. Например:

- [ ] – пустой список
- [B] – список, состоящий из одного элемента B;
- [A, B, C] – список, состоящий из трех элементов A, B, C;
- [x, C, d1, “hello”, (d1, m2)] – элементами списка могут быть любые термы – переменные, константы, структуры и пр.

Преимущество использования списков – в более естественном и компактном представлении информации. Например, факт того, что студент Иванов изучает определенные дисциплины, можно записать следующим образом:

```
clauses
изучает("Иванов", ["математика", "физика", "информатика"]).
```

Поведение переменных, входящих в списки, ничем не отличается от поведения переменных как аргументов предиката. Первоначально они имеют неконкретизированное значение, но в любой момент могут быть конкретизированы.

Описание список на Visual Prolog

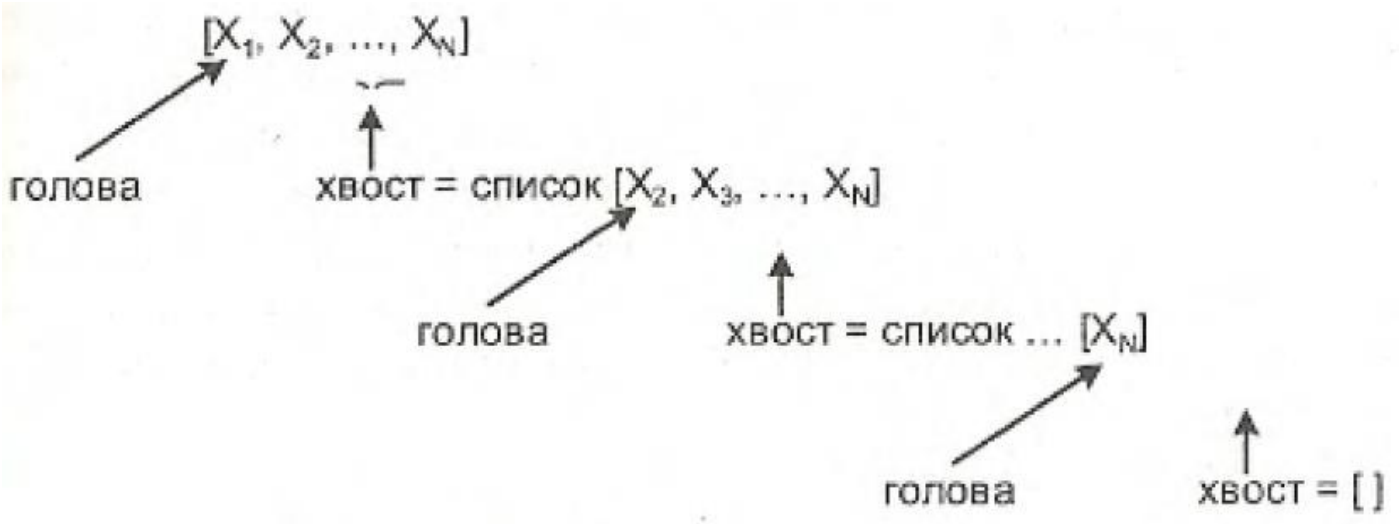
В Visual Prolog все данные и предикаты должны быть объявлены и описаны перед их использованием. Это накладывает определенные ограничения на использование списков с элементами разной природы, разного типа. Если список включает элементы разного типа, в том числе подсписки, то они рассматриваются как структуры, что значительно усложняет описание операций и представление элементов списка. Поэтому **на Visual Prolog списки чаще всего рассматриваются как совокупности однотипных элементов данных**. Вначале такие списки мы и будем рассматривать.

Списки описываются в разделе программы ***Domains***:

```
domains
   ilist = integer*. % список int
   slist = string*. % список string
domains
   name = string.
   group = integer.
   student = student(name, group).
   students = student*.
```

Голова и хвост списка

Работа со списками основана на расщеплении их на голову и хвост списка. Головой списка является его первый элемент. Хвост списка – это **список** из всех элементов исходного списка, за исключением его первого элемента (головы):



Список имеет рекурсивную структуру, то есть, определяется через самого себя. Список – это любой пустой список [], не содержащий ни одного элемента, или структура, имеющая два компонента [H|T] – голову H и хвост T списка. Голова H – это первый элемент списка, а хвост T – список из оставшихся элементов. Конец списка обычно представляют, как хвост, который является пустым списком. Пустой список не имеет ни головы, ни хвоста.

Список	*Голова*	*Хвост*
[a, b, c]	a	[b, c]
[a]	a	[]
[]	нет головы	нет хвоста

Так как операция расщепления списка на голову и хвост очень широко используется, то в Прологе введена специальная форма для представления непустого списка с головой H и хвостом T. Это записывается как [H|T], где для разделения H и T используется вертикальная черта. При конкретизации такой структуры H сопоставляется с головой списка, а T – с хвостом списка.

Унификация списков как аргументов предикатов

Одной из фаз процесса доказательства некоторого целевого предиката является унификация его аргументов с аргументами утверждений базы знаний. Как проходит унификация, если аргументами являются списки? Введем понятие шаблона (образца) списка. **Шаблон списка** – это форма описания множества (семейства) списков, обладающих вполне определенными свойствами. Так, шаблон [X|Y] описывает любой произвольный список, состоящий не менее чем из одного элемента. Шаблон [X1, X2|Y] описывает любой произвольный список, состоящий не менее чем из двух элементов. Шаблон Z – описывает любой список, в том числе и пустой. Шаблон может содержать как переменные, так и

константы. Например, шаблон [b|Z] задает любой список, первым элементов которого является элемент b.

При унификации происходит сопоставление шаблонов. Если шаблоны целевого утверждения и утверждения базы знаний представляют списки с несовместимыми различными свойствами (разные классы списков), то унификация заканчивается неудачей. Так, нельзя сопоставить списки:

[X1, X2|T] и [a];

[b, X|Y] и [a, b].

Если шаблоны не противоречат друг другу, то осуществляется конкретизация отдельных переменных шаблона, то есть, присвоение им значений соответствующих констант, или сцепление с соответствующими переменными другого шаблона. В результаты оба шаблона должны стать идентичными и породить общее решение – новый шаблон. Например: при сопоставлении шаблона [X, Y|Z] и списка [a, b, c] унификация пройдет успешно, и переменные принимают следующие значения: X=a, Y=b, Z=[c] (Z является хвостом).

Если в процессе сопоставления и присвоения значений шаблоны не могут стать идентичными, то унификация заканчивается неудачей. Например:

[«привет», Y, Y] и [X, X, «Пролог»],

где сопоставление показывает, что элементы обоих список должны быть одинаковы, однако «привет» не равно «Пролог».

Принадлежность элементов списку

Предположим, что имеется некоторый список студентов:

```
clauses
    ["Иванов", "Петров", "Сидоров", "Смирнов"].
```

Необходимо определить, имеется ли некоторый студент, например, *Сидоров*, в этом списке. В Прологе это можно сделать, определив отношение принадлежности объекта некоторому списку с помощью предиката *принадлежит (contains)*. Целевое утверждение *contains(X, L)* является истинным, если терм, связанный с X, является элементом списка L. Чтобы описать этот предикат, рассмотрим понятие «является элементом списка». Его суть раскрывается так: «некоторый объект является элементов списка, если он:

- Либо совпадает с головой списка;
- Либо является элементом хвоста списка».

Из определения следует, что необходимы два правила для описания предиката *contains*. Первое говорит о том, что объект X будет элементом списка, если он совпадает с головой списка L. На Прологе это можно записать так: *contains(X, [X|_])*. Используем анонимную переменную «_» для обозначения хвоста, так как хвост в этом факте не используется.

Второе правило говорит о том, что X принадлежит списку также при условии, что он является элементом хвоста списка L. Эта информация может быть записана следующим образом:

```
contains(X, [_|T]) :- contains(X, T).
```

Анонимная переменная «_», обозначающая голову списка, говорит о том, что информация о голове списка не имеет никакого значения для выбранного пути решения. Два этих правила в совокупности определяют предикат для отношения принадлежности и указывают, каким образом **просматривать список от начала до конца** при поиске некоторого элемента в списке.

Наиболее важный момент при работе с рекурсивными предикатами состоит в том, что прежде всего надо найти граничные условия и способ использования рекурсии. Для предиката *contains* имеются два граничных условия: либо искомый объект содержится в голове списка, либо не содержится вовсе. Первое распознается первым утверждением, которое приводит к прекращению поиска в списке. Второе условие встречается, когда второй аргумент предиката *contains* является пустым списком.

Каждый раз, когда при поиске соответствия для целевого предиката *contains* происходит рекурсивное обращение к тому же предикату, новая цель формируется **для более короткого списка**. Очевидно, рано или поздно произойдет одно из двух: либо произойдет сопоставление с первым правилом для *contains*, либо в качестве второго аргумента *contains* будет задан пустой список. Как только возникает одна из этих ситуаций, прекратится рекуррентное порождение новых подцелей. Второе граничное условие не распознается ни одним из утверждений для *contains*, так что процесс поиска сопоставимого элемента списка для целевого утверждения *contains* закончится неудачей.

Итоговая реализация на языке Visual Prolog для целочисленного списка выглядит следующим образом:

```
domains
    ilist = integer*.
class predicates
    contains : (integer, ilist) determ (i,i).
clauses
    contains(X, [X|_]) :- !.
    contains(X, [_|T]) :- contains(X, T).
```

Достоинством этого предиката является то, что он показывает, как с помощью рекурсивного определения получить доступ к каждому элементу списка.

Простейшие предикаты работы со списками (с заданиями)

Рассмотрим некоторые простейшие предикаты для работы со списками.

1. Добавление элемента в начало списка (перед текущей головой). Для целочисленного списка.
   ```
   class predicates
       addHead : (integer, ilist, ilist) procedure (i,i,o).
   clauses
       addHead(H, T, [H|T]).
   ```
2. *Задание 1*. Добавление элемента в конец списка. Ответы в конце материала.
3. *Задание 2*. Определение индекса первого совпадения в списке. Если элемента в списке нет, вернуть -1. Для реализации предиката потребуется вспомогательный предикат.
   ```
   class predicates
       indexof : (integer, ilist, integer) procedure (i,i,o).
       indexof1 : (integer, ilist, integer, integer) procedure (i,i,i, o).
   ```

4. Количество элементов в списке. Используется анонимная типизация. **A*** - обозначает список из любых однотипных элементов. Использование переменной (например, **A**) вместо типа данных позволяет создавать предикаты, работающие с любым типом данных.

```
class predicates
    length_of : (A*, integer) procedure (i,o).
clauses
    length_of([], 0).
    length_of([_|T], Length + 1) :- length_of(T, Length).
```

5. *Задание 3*. Реализовать предикаты, вычисляющие сумму, количество и среднее значение элементов целочисленных списков.
6. Вывод элементов списка. Для вывода можно использовать 2 способа. Первый состоит в выводе списка как терма и сводится к простому вызову предиката *write(L)*, где L является списком. Второй способ – поэлементный вывод списка, организованный рекурсивными предикатами:

```
class predicates
    write_list : (A*) procedure (i).
clauses
    write_list([]).
    write_list([H|[]]) :- write(H, "."), !.
    write_list([H|T]) :- write(H, ", "), write_list(T).
```

Предикат *Присоединить (append)*

Рассмотрим простой, но очень полезный предикат *присоединить*. Его основное назначение – соединение двух списков в один. Согласование с базой знаний целевого утверждения *append(X, Y, Z)* должно заканчиваться удачей в том случае, если Z представляет собой соединение двух списков X и Y. Например, целевое утверждение

append([1, 2], [3, 4], [1, 2, 3, 4]).

является истинным, а утверждение

append([1, 2, 3], [3, 3], Z), write(Z).

позволяет получить новый список Z = [1, 2, 3, 3, 3] в результаты объединения двух исходных списков.

Кроме основного назначения, существует еще множество других применение, например, таких, как разделение списка всеми возможными способами, удаление начального или остаточного сегмента списка, разделения списка по заданному элементу.

Чтобы разработать предикат для некоторого отношения, нужно составить утверждения базы знаний, совокупность которых полностью описывает это отношение. На практике при определении отношения над списками нужно поступать следующим образом: выбрав один из аргументов этого отношения, составить высказывания относительно различных вариаций этого аргумента. Эти варианты должны покрывать все различные виды список, которые могут появляться в качестве данного аргумента этого отношения.

Для отношения *append(X, Y, Z)* выберем первый аргумент X. Полностью определим это отношение, задав одно утверждение для случая, когда X пустой список [], и второе – для всех случаев, когда X непустой список, то есть, представим шаблоном [X1|L1].

Когда X=[], Y и Z всегда совпадают. Любой список, присоединенный к пустому списку, дает тот же самый список. Это можно представить фактом

append([], Y, Y).

Смысл второго утверждения

append([X1|T1], Y, [X1|T2]) :- append(T1, Y, T2).

можно описать словами. Первый элемент X1 первого списка X будет и первым элементом третьего списка, то есть, если X = [X1|T1], то Z = [X1|T2].

Хвост T2 третьего аргумента Z всегда будет определять результат присоединения второго аргумента Y к хвосту L1 первого списка X. Для выполнения этой операции присоединения Y к L1 необходимо использовать предикат *append*.

Так как при каждом обращении к правилу удаляется голова списка, являющегося первым аргументом, то постепенно этот список будет исчерпан и станет пустым, так что произойдет выход на первое граничное условие.

Программа выглядит следующим образом.

```
class predicates
    append : (A*, A*, A*) nondeterm (i, i, o) (o, i, i) (i, o, i) (o, o, i) (i, i, i).
clauses
    append([], Y, Y).
    append([X1|T1], Y, [X1|T2]) :- append(T1, Y, T2).
```

Использование предиката append для разделения списков

Предикат append можно использовать и для разделения списка на два подсписка, если первые два аргумента объявить как переменные, а третий – конкретизированным списком. Например, цель

append(X, Y, [1, 2, 3]).

породит следующие решения:

X = [] Y = [1, 2, 3]

X = [1] Y = [2, 3]

X = [1, 2] Y = [3]

X = [1, 2, 3] Y = []

Если требуется получить только непустые списки, то запрос следует преобразовать к такому виду:

append([X|Tx], [Y|Ty], [1, 2, 3].

Аналогично различными шаблонами можно задавать различные варианты разбиения списка.

Операции со списками (с заданиями)

Так как список – рекурсивная структура, то все операции со списками имеют рекурсивную структуру. Операции можно разделить по сложности на три группы. Самая простая группа – список задан и надо найти его характеристики (сумму, среднее). Вторая по сложности группа –

предобразование списка по заданному правилу, третья – записать в список элементы, получаемые от генератора или по определенному закону.

1. *Задание 4*. Даны баллы студента за все время обучения в виде списка, [4, 5, 3, 5, 4, 3, 4, …]. Необходимо определить средний балл, количество пятерок, процент четверок среди всех баллов. **Решение в конце не представлено**.
2. Дан список целых чисел. Необходимо исключить из него все элементы другого списка.
```
class predicates
    exclude : (ilist Исходный_список, ilist Исключаемый_список, ilist Результат)
        procedure (i,i,o).
    exclude_elem : (integer Исключаемый_элемент, ilist Список, ilist Результат)
        procedure (i,i,o).
clauses
    exclude(List, [], List).
    exclude(List, [N|T], Res) :- exclude_elem(N, List, R), exclude(R, T, Res).
    exclude_elem(_, [], []).
    exclude_elem(N, [N|T], T):- !.
    exclude_elem(N, [Y|T], [Y|Tr]) :- exclude_elem(N, T, Tr).
```
3. *Задание 5*. Дано целое число N. Необходимо сгенерировать список, содержащий все числа от 1 до N включительно. Предикат имеет следующую сигнатуру:
```
class predicates
        gen : (integer N, ilist List) determ (i, o).
```
4. Даны два вектора, представленные в виде списков. Необходимо найти произведение векторов.
```
class predicates
    vectors_mult : (real*, real*, real) determ (i,i,o).
clauses
    vectors_mult([],_,0).
    vectors_mult([H1|T1], [H2|T2], Res + H1 * H2) :-
        vectors_mult(T1, T2, Res).

    run() :-
        vectors_mult([1,2,3, 4], [-1, 0, 3, 5], X),
        write(X), nl, !.
    run() :- succeed.
```
5. *Задание 6.* Реализовать произведение матрицы на вектор. Матрица представляется как список списков. В качестве типа аргумента указывается **real****. **Решение в конце не представлено.**
6. *Задание 7.* Реализовать произведение матриц. **Решение в конце не представлено.**

Решения к заданиям

Задание 1. Решение:

```
class predicates
    addTail : (integer, ilist, ilist) procedure (i,i,o).
clauses
    addTail(X, [], [X]).
    addTail(X, [H|T], [H|Res]) :- addTail(X, T, Res).
```

Задание 2. Решение:

```
class predicates
    indexof : (integer, ilist, integer) procedure (i,i,o).
    indexof1 : (integer, ilist, integer, integer) procedure (i,i,i, o).
clauses
    indexof(X, L, N) :- indexof1(X, L, 1, N).
    indexof1(_, [], _, -1) :- !.
    indexof1(X, [X|_], N, N) :- !.
    indexof1(X, [_|T], K, N) :- indexof1(X, T, K+1, N).
```

Задание 3. Решение

```
class predicates
    sum_count : (ilist, integer, integer) procedure (i, o, o).
    rating_avg : (ilist, real) procedure (i, o).
clauses
    rating_avg(L, S / C) :- sum_count(L, S, C).
    sum_count([], 0, 0).
    sum_count([H|T], St + H, Ct + 1) :- sum_count(T, St, Ct).
```

Задание 4. Решение

Задание без решения.

Задание 5. Решение

```
domains
    ilist = integer*.
class predicates
    append : (A*, A*, A*) nondeterm (o, i, i) (i, o, i) (o, o, i) (i, i, i).
    append : (A*, A*, A*) procedure (i, i, o).
    gen : (integer N, ilist List) determ (i, o).
    gen : (integer N, ilist List, integer Iter, ilist IterList) determ (i, o, i, i).
clauses
    append([], Y, Y).
    append([X1|T1], Y, [X1|T2]) :- append(T1, Y, T2).
    gen(N, L) :- gen(N, L, 1, [1]).
    gen(N, L, N, L) :- !.
    gen(N, L1, K, L2) :- K1 = K + 1, append(L2, [K1], L3), gen(N, L1, K1, L3).

    run() :- gen(10, X), write(X), !; succeed.
```