Представление данных в языке Пролог

Способы представления данных в языке Пролог

- **База данных фактов**
- **Структуры**
- **Списки**

Базы данных в Прологе

Под *Базой Данных (БД)* понимается совокупность фактов.

Если набор фактов задается на этапе разработки программы и не изменяется во время работы, то база данных является статической.

Статические БД используются для представления устойчивых отношений, неизменных во времени, например, справочной информации.

Все предикаты, описанные в разделе predicates, относятся к статической БД.

Пролог позволяет работать с динамическими БД, в которые можно записывать новые факты и удалять устаревшие во время работы программы.

Динамические БД позволяют отражать изменения данных.

Все предикаты, описанные в разделе database, относятся к динамической БД.

Задание статической базы данных в Прологе

```
domains
    address=address(zip,country,city,street,house)
    name,zip,country,city,street,house,phone,fax=symbol

predicates
    manufacturer(name,address,phone,fax)

clauses
manufacturer("НПО Планета",
                 address("173004","Россия","Новгородская область",
                "Великий Новгород","ул.Федоровский Ручей","2/13"),
                 "(81622) 31736", "(81622) 33286")).

manufacturer("ФГУП ПО Квант",
                 address("173000","Россия","Новгородская область",
                "Великий Новгород","ул.Большая Санкт-Петербургская",
                "73/1"),
                "(81622) 27117", "(81622) 24333")).
```

Работа с динамической базой данных в Прологе

К встроенным предикатам для работы с динамической БД относятся : asserta, assertz, retract и findall.

Предикаты asserta и assertz заносят новые факты в резидентную БД. При этом при использовании asserta добавляемый факт помещается перед всеми уже внесенными утверждениями данного предиката, а при использовании assertz - после. Синтаксис : asserta(Clause) и assertz(Clause).

Предикат retract удаляет имеющееся утверждение из динамической БД. Его синтаксис : retract(Existing_clause).

Для модификации БД рекомендуется использовать комбинацию выражений с предикатами asserta, assertz и retract.

Задание динамической базы данных в Прологе

```
domains
    address=address(zip,country,city,street,house)
    name,zip,country,city,street,house,phone,fax=symbol

database
    dmanufacturer(name,address,phone,fax).

goal
asserta(dmanufacturer("НПО Планета",
          address("173004","Россия","Новгородская область",
                "Великий Новгород","ул.Федоровский Ручей","2/13"),
          "(81622) 31736", "(81622) 33286")),
asserta(dmanufacturer("ФГУП ПО Квант",
          address("173000","Россия","Новгородская область",
               "Великий Новгород","ул.Большая Санкт-Петербургская",
               "73/1"),
          "(81622) 27117", "(81622) 24333"))
```

Сбор данных БД в список

Для сбора данных из БД с целью их последующей обработки используется встроенный предикат findall. Синтаксис предиката следующий : findall(Var_name,Dbase_pred_name,List_name), где

Dbase_pred_name - имя предиката БД, Var_name соответствует имени атрибута, List_name - имя переменной выходного списка, причем элементы списка принадлежат к тому же домену, что и Var_name. При этом домен списка должен быть объявлен в разделе domains.

```
domains
  . . .
    name_list=name*
goal
    findall(Name,dmanufacturer(Name,_,_,_),Name_list).
```

Согласование данной цели дает :

Name_list=[“НПО Планета”,”ФГУП ПО Квант]

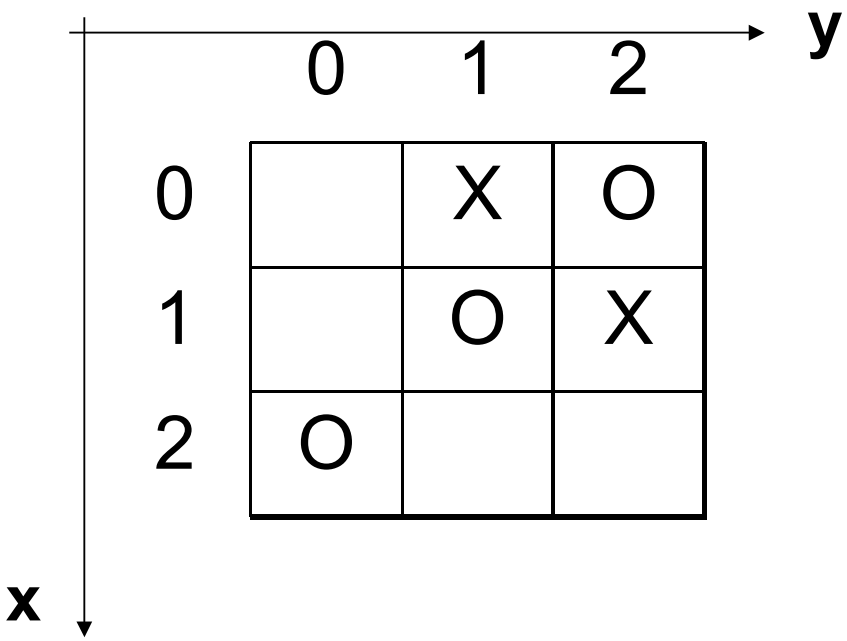
Задание

Игра «крестики-нолики». Представьте игровое поле в виде динамической базы данных фактов.

Создайте предикат init для задания произвольного состояния игрового поля.

Создайте предикат set для задания крестика или нолика в заданной позиции игрового поля.

Создайте предикат victory для определения выигрышной ситуации и победителя.



Структуры

Определение. *Структурированными объектами* (или *структурами)* называются объекты, которые имеют несколько компонентов. Сами компоненты, в свою очередь, также могут быть структурами.

Например, дата может рассматриваться как структура с тремя компонентами: число, месяц, год.

date(1, may, 2001)

Несмотря на то что структуры состоят из нескольких компонентов, они рассматриваются в программе как целостные объекты.

Структуры используются в качестве аргументов предикатов.

Структуры можно изображать в виде деревьев:

```
    data
     / | \
01 may 2001
```

Задание структуры в Прологе

Структура должна быть задана в разделе domains, если используется как параметр предиката (чаще всего), или в разделе predicates, если не используется

```
domains
    point=point(real, real)

predicates
    segm(point, point)
    triangle(point, point, point)

clauses

  segm(point(3, 3), point(3, 5)).

  triangle(point(4, 6), point(10, 5), point(7, 7)).
```

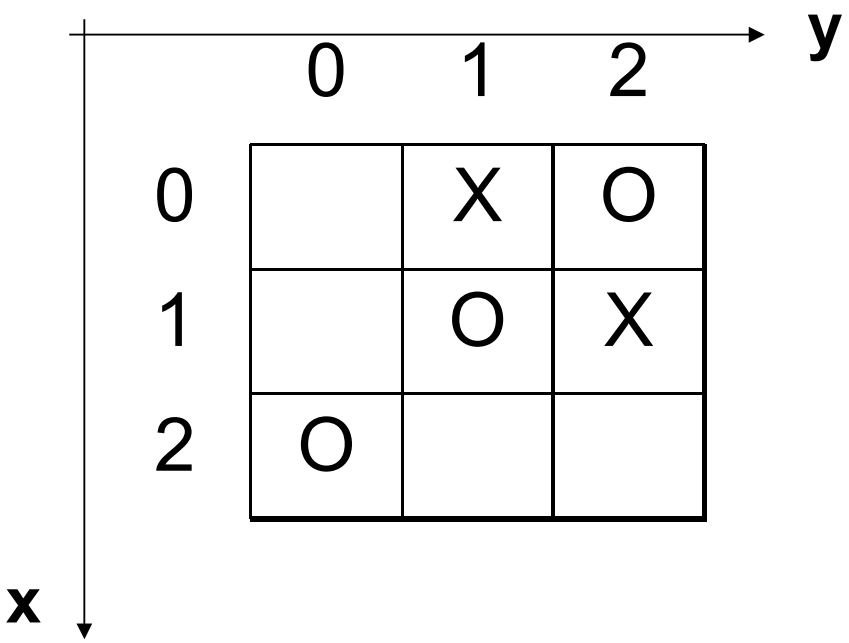
Задание

Игра «крестики-нолики». Представьте игровое поле в виде структуры.

Создайте предикат init для задания произвольного состояния игрового поля.

Создайте предикат set для задания крестика или нолика в заданной позиции игрового поля.

Создайте предикат victory для определения выигрышной ситуации и победителя.



Список как частный вид структуры

Определение. Под списком понимается упорядоченная последовательность элементов, которая может иметь произвольную длину.

Признак *упорядоченный* указывает на то, что порядок элементов в последовательности является существенным и список [1,2,3] не эквивалентен списку [3,2,1].

Элементами списка могут быть любые термы - константы, переменные, структуры, последние могут включать в себя другие списки.

Списки – это частный вид структуры, которая предопределена в языке Пролог. Список является рекурсивной структурой данных.

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

Список - это либо пустой список, не содержащий не одного элемента, либо структура, имеющая два компонента : голову и хвост. Конец списка представляется как хвост, который является пустым списком.

Способы представления списков.

Их 3 : функторный, графический и скобочный.

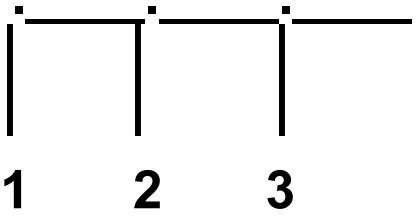
При использовании функторной формы записи голова и хвост являются компонентами функтора, обозначаемого точкой “.”.

При использовании графической формы записи список представляется как специального вида дерево, “растущее” слева направо, причем ветви направлены вниз (“виноградная гроздь”).

При использовании скобочной формы записи последовательность элементов списка, разделенных запятыми, заключается в квадратные скобки.

Пример : список из трех чисел.

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



[1,2,3]

Описание списков в Прологе

Графическая форма записи списка в виде “виноградной лозы” удобна для записи списков на бумаге, но в Пролог-программах не используется. Функторная форма записи оказывается неудобной для записи сложных списков. В Прологе для записи списков используется скобочная форма записи.

Работа со списками основана на расщеплении на *голову* и *хвост* : [Head| Tail]. Голова есть первый аргумент функтора “.”, хвост - второй : . (Head, [Tail]]). Функтор “.” используется для конструирования списка. *Хвост* списка *есть список*, состоящий из всех элементов исходного списка, за исключением первого.

Примеры : [1,2,3] - Head :1, Tail : [2,3] ; [[1], [2]] - Head : [1], Tail : [[2]].

Используя скобочную форму записи списков, можно создавать похожие на списки структуры, но не заканчивающиеся пустым списком (аналоги точечных пар Лиспа). Пример : [1 | 2]. Здесь Head : 1, Tail : 2.

Применение списков в программе.

Домен списка описывается в разделе domains, а работающий со списком предикат - в разделе predicates.

Сам список задается в программе либо в разделе clauses, либо в разделе goal. Пример :

```
domains
      os_list=symbol*
predicates
       print_list(os_list)
goal
       print_list( [«DOS»,»Windows»,»Novell»,»Unix»,»Linux»] )
clauses
       print_list([ ]).
       print_list([ Head | Tail ]) :-
                      write(Head), nl, print_list(Tail).
```

Правила сопоставления списков.

Сопоставление списков - путем конкретизацией переменных.

Список 1	Список 2	Результат
[X,Y,Z]	[cat,dog,mouse]	X=cat Y=dog Z=mouse
[dog]	[X|Y]	X=dog Y=[]
[X,Z|Y]	[cat,dog,mouse]	X=cat Z=dog Y=[mouse]
[[b,c]|Z]	[[X,Y]|[w,b]]	X=b Y=c Z=[[w,b]]
[X,Y|Z,W]	Синтаксически некорректная конструкция списка	
[white,cat]	[cat,X]	Сопоставлени е невозможно
[white|Q]	[P|cat]	P=white Q=cat

Суммирование элементов списка

Кол-во аргументов : 1, тип аргумента : список целых чисел.

Условие завершения рекурсии : пустой список.

Описание рекурсивного правила :

sum_lst ([ ] , 0).

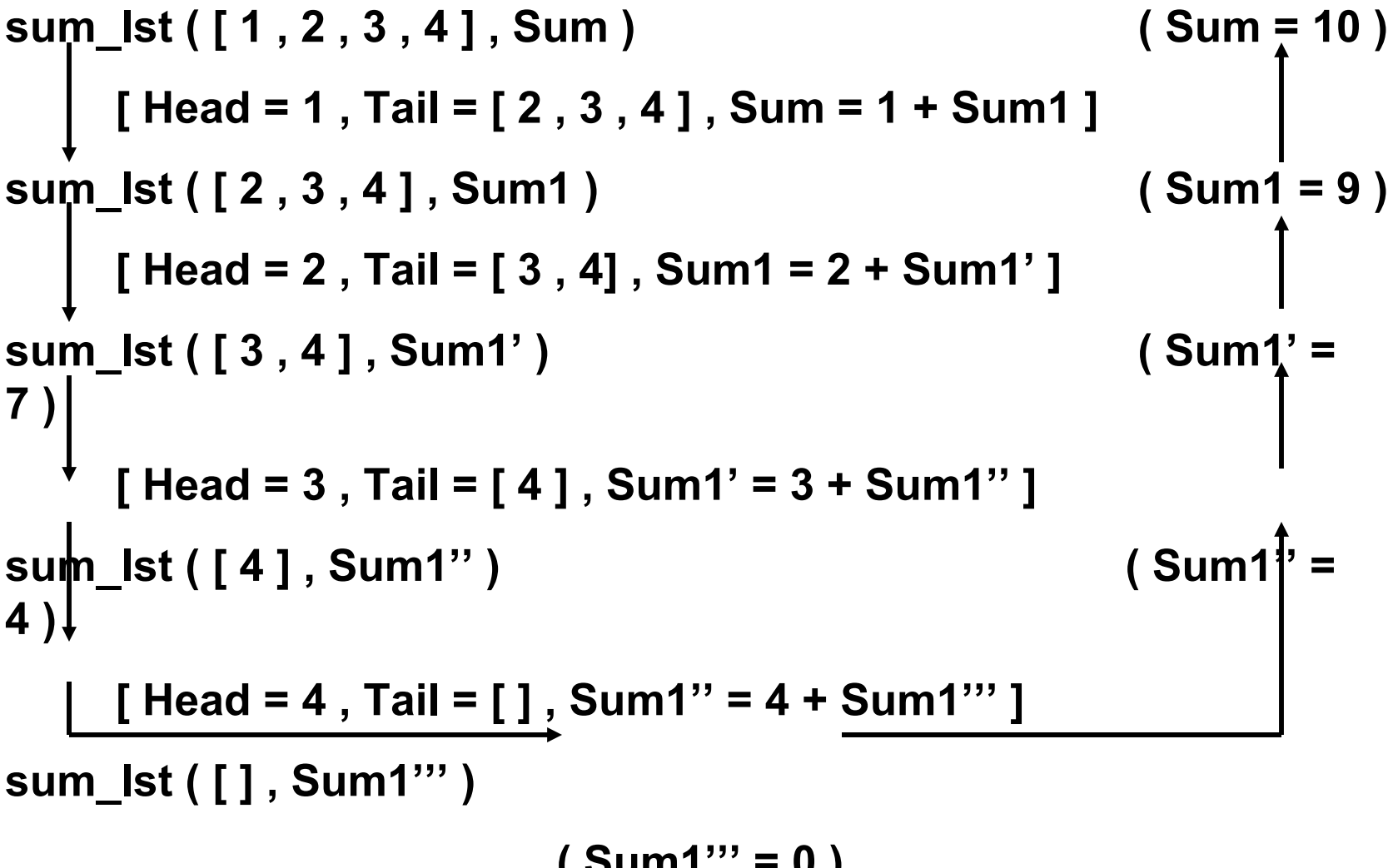
sum_lst ([Head | Tail] , Sum) : -

sum_lst (Tail , Sum1) ,

Sum = Head + Sum1 .

Рассмотрим работу правила на примере нахождения суммы элементов списка [1 , 2 , 3 , 4].

Этапы решения задачи “Нахождение суммы элементов списка”



Печать элементов списка

Постановка задачи. Есть список. Требуется напечатать все элементы списка в строчку.

Кол-во аргументов : 1.

Условие завершения рекурсии : пустой список.

Описание рекурсивного правила :

```
print_list ( [ ] ) .

print_list ( [ Head | Tail ] ) : -

                write ( Head , « «) , print_list ( Tail ).
```

Аналогично можно организовать печать элементов списка в столбик :

```
print_list_col ( [ ] ) .

print_list_col ( [ Head | Tail ] ) : - write ( Head ) , nl ,

                                      print_list_col ( Tail ) .
```

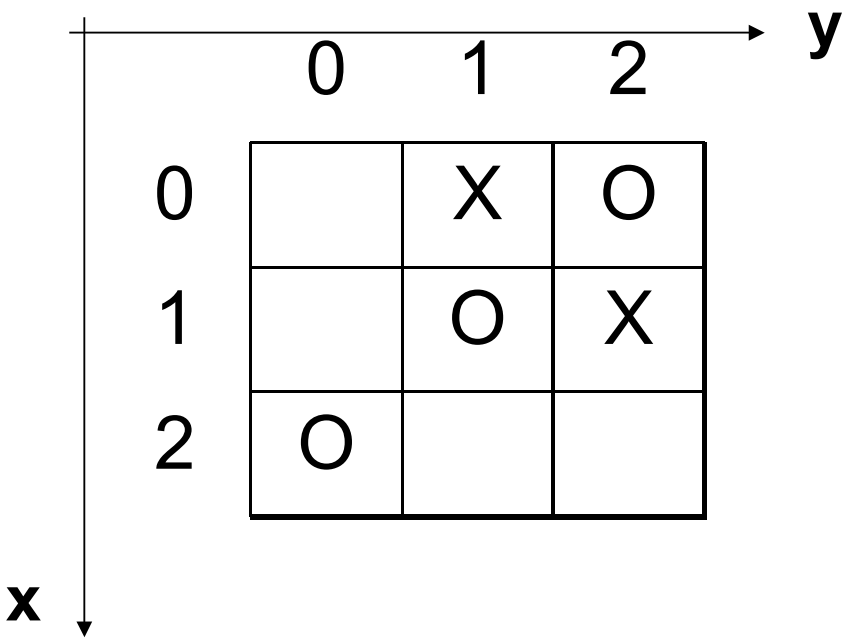
Задание

Игра «крестики-нолики». Представьте игровое поле в виде списка, где элементом списка является строка игрового поля.

Создайте предикат init для задания произвольного состояния игрового поля.

Создайте предикат set для задания крестика или нолика в заданной позиции игрового поля.

Создайте предикат victory для определения выигрышной ситуации и победителя.



Что читать

Братко, Иван.
Б87 Алгоритмы искусственного интеллекта на языке PROLOG, 3-е издание. : Пер. с англ. — М. : Издательский дом "Вильяме", 2001. — 640 с.

Глава 3. Списки, операции, арифметические выражения
3.1. Представление списков

Глава 4. Использование структур: примеры программ
4.1. Выборка структурированной информации из базы данных

Глава 7. Дополнительные встроенные предикаты
7.4. Операции с базой данных

У.Клоксин, К. Мелиш.
Программирование на языке Пролог

Глава 3. Использование структур данных