

Логическое программирование

Кевролетин В.В. группа с8403а(246)

12 June 2012

Содержание

1	Задание 20	1
1.1	Условие	1
1.2	occurrences/3	1
1.2.1	Исходный код	1
1.2.2	Тесты	2
1.3	position/3	4
1.3.1	Исходный код	4
1.3.2	Тесты	5

1 Задание 20

1.1 Условие

Написать программы для отношений occurrences/3 (9.2i) и position/3 (9.2ii).

1.2 occurrences/3

1.2.1 Исходный код

Если 2 терма равны, то число вхождений равно 1. Если терм, в котором происходит поиск составной, то следует рекурсивно применить поиск для всех его аргументов. В реализации введены 2 вспомогательных терма: compare_terms сравнивает 2 терма на равенство и возвращает 0 или 1. compare_arguments итеративно применяет проверку для каждого аргумента составного терма.

occurences(Sub, Term, N) истина, если N - число вхождений терма Sub в терм Term

Sub произвольный тип

Term произвольный тип

N число

compare_terms(Term1, Term2, X) истина, если 2 терма равны и $X = 1$, либо не равны и $X = 0$

Term1 произвольный тип

Term2 произвольный тип

X число

compare_arguments(Sub, Term, ArgNum, CurrN, N) истина, если N - сумма числа вхождений термина Sub в первые ArgNum аргументов термина Term и числа CurrN

Sub произвольный тип

Term произвольный тип

ArgNum число

CurrN число

N число

```
ocurrences(Sub, Term, N) :-
    compound(Term),
    functor(Term, _, ArgCnt),
    compare_terms(Sub, Term, X),
    compare_arguments(Sub, Term, ArgCnt, X, N).
```

```
ocurrences(Sub, Term, N) :-
    \+ compound(Term),
    compare_terms(Sub, Term, N).
```

```
compare_terms(Term, Term, 1).
```

```
compare_terms(Term1, Term2, 0) :-
    \+ compare(=, Term1, Term2).
```

```
compare_arguments(_, _, ArgNum, CurrN, CurrN) :-
    =(ArgNum, 0).
```

```
compare_arguments(Sub, Term, ArgNum, CurrN, N) :-
    >(ArgNum, 0),
    arg(ArgNum, Term, Arg),
    ocurrences(Sub, Arg, ArgN),
    (NewN is ArgN + CurrN),
    (NextArgNum is ArgNum - 1),
    compare_arguments(Sub, Term, NextArgNum, NewN, N).
```

1.2.2 Тесты

- ocurrences(+, +, +)

```
?- ocurrences(a, a, 1).
```

```
true
```

```
?- ocurrences(a, b, 0).
```

```
true.
```

?- ocurrences(a, [a, a], 2).

true

?- ocurrences(a, node(node(a, a), node(a, null)), 3).

true

?- ocurrences(a, node(null, null), 0).

true

?- ocurrences(node(a, b), node(node(a, b), node(a, node(a, b)))), 2).

true

?- ocurrences(node(a, b), node(node(a, b), node(a, node(a, b)))), 3).

false.

• ocurrences(+, +, -)

?- ocurrences(a, [b, c, d], X).

X = 0

?- ocurrences(a, [b, c, d], X).

X = 0

?- ocurrences(b, [b, c, d], X).

X = 1

?- ocurrences([d], [d, a, b, c, d], X).

X = 1

• ocurrences(-, +, +)

?- ocurrences(X, a, 10).

false.

?- ocurrences(X, a, 1).

X = a

?- ocurrences(X, node(a, a), 1).

`X = node(a, a)`

`?- ocurrences(X, node(a, a), 2).`

`X = a`

`?- ocurrences(X, node(a, a), 3).`

`false.`

- `ocurrences(-, +, -)`

`?- ocurrences(X, a, Y).`

`X = a,`

`Y = 1`

`?- ocurrences(X, node(nil, node(a, b)), Y).`

`X = node(nil, node(a, b)),`

`Y = 1`

`X = node(a, b),`

`Y = 1`

`X = b,`

`Y = 1`

`X = a,`

`Y = 1`

`X = nil,`

`Y = 1`

1.3 position/3

1.3.1 Исходный код

position(Sub, Term, Result) истина, если Result содержит список, описывающий положение терма Sub в терме Term

Sub произвольный тип

Term произвольный тип

Result список чисел

Будем восстанавливать порядок обхода снизу вверх:

`position(Term, Term, []).`

`position(Sub, Term, Result) :-
 compound(Term),`

```

functor(Term, _, N),
    position(Sub, Term, N, Result).

position(Sub, Term, N, Result) :-
    N > 1,
    N1 is N - 1,
    position(Sub, Term, N1, Result).

position(Sub, Term, N, [N | Result]) :-
    arg(N, Term, Arg),
    position(Sub, Arg, Result).

```

1.3.2 Тесты

- position(+, +, +)

```

?- position(a, a, []).
true

```

```

?- position(c, [a, b, c], [2, 2, 1]).
true

```

- position(+, +, -)

```

?- position(null, null, []).

true

```

```

?- position(a, node(a), [1]).

true

```

```

?- position(a, node(b, node(a, a)), X).

```

```

X = [2, 1]

```

```

X = [2, 2]

```

```

?- position(a, node(b, c), X).
false.

```

- position(-, +, +)

```

?- position(X, [a, a, b, c], [1]).
X = a

```

```
?- position(X, [a, a, b, c], [2, 1]).  
X = a
```

```
?- position(X, [a, a, b, c], [2, 2, 1]).  
X = b
```

- position(-, +, -)

```
?- position(X, node(nill, nill), Y).  
X = node(nill, nill),  
Y = []
```

```
X = nill ,  
Y = [1]
```

```
X = nill ,  
Y = [2]
```

```
?- position(X, [a, b, c], Y).  
X = [a, b, c],  
Y = []
```

```
X = a ,  
Y = [1]
```

```
X = [b, c] ,  
Y = [2]
```

```
X = b ,  
Y = [2, 1]
```

```
X = [c] ,  
Y = [2, 2]
```

```
X = c ,  
Y = [2, 2, 1]
```

```
X = [] ,  
Y = [2, 2, 2]
```