

Оператор отсечения и рекурсия в языке Пролог

Указания

Последовательное выполнение действий

predicates

vowel(char)

clauses

vowel('a').

vowel('e').

vowel('i').

vowel('o').

vowel('u').

vowel('y').

goal

write("Enter character: "), readchar(X), write(X), nl, vowel(X), write(" is vowel"), nl.

Проверка одновременного выполнения двух условий

predicates

```
is_digit(char)
```

clauses

```
is_digit(C):- C>='0', C<='9'.
```

goal

```
readchar(X), write(X), nl, is_digit(X), write(" is digit"), nl.
```

Проверка выполнения хотя бы одного условия из нескольких

predicates

```
nondeterm not_digit(char)
```

clauses

```
not_digit(C):- C<'0'.
```

```
not_digit(C):- C>'9'. или
```

goal

```
readchar(X), write(X), nl, not_digit(X), write(" not digit"), nl.
```

Возврат данных из предиката

predicates

increment(integer, integer)

clauses

increment(X, Y):- Y = X + 1.

goal

increment(10, Res).

1 Решение: Res = 11

Неверная реализация

predicates

increment(integer)

clauses

increment(X):- $X = X + 1$.

goal

increment(10).

Нет решений

Если ..., то ...

predicates

sgn(integer, integer)

clauses

sgn(X, Result):- X>0, Result=1.
sgn(X, Result):- X<0, Result=-1.
sgn(X, Result):- X=0, Result=0.

goal

sgn(-15, Res).

1 Решение: Res = -1

Если ..., то ..., иначе ...

predicates

min(integer, integer, integer)

clauses

min(X, Y, Z):- X<Y, Z=X, !.

min(X, Y, Z):- Z=Y.

goal

min(1, 5, Res).

1 Решение: Res = 1

Без использования отсечения
2 Решения: Res = 1 и Res = 5.

Если ..., то ..., иначе ...

predicates

min(integer, integer, integer)

clauses

если то
min(X, Y, Z):- X<Y, Z=X, !.

иначе
min(X, Y, Z):- Z=Y.

goal

min(1, 5, Res).

1 Решение: Res = 1

Без использования отсечения
2 Решения: Res = 1 и Res = 5.

Вызов предиката из другого предиката для проверки условия

predicates

is_digit(char)

nondeterm analyze(char)

clauses

is_digit(A):- A>='0', A<='9'.

analyze(B):- is_digit(B), write(" is digit"), nl, !.

analyze(B):- write(" is not digit"), nl.

goal

write("Enter character: "), readchar(X), write(X), analyze(X).

Вызов предиката из другого предиката для выполнения расчета

predicates

distance(real,real,real,real,real)

in_circle(real,real,real,real,real)

clauses

distance(X1,Y1,X2,Y2,D):- DX = X2 - X1, DY = Y2 - Y1,
D = sqrt(DX*DX + DY*DY).

in_circle(X, Y, XC, YC, R):- distance(X, Y, XC, YC, D), D<=R.

goal

in_circle(1,1,5,5,10).

ОТВЕТ: yes

Последовательные вычисления

predicates

absolute(integer, integer)

min(integer, integer, integer)

calculate(integer, integer) /* calculate(X, Y) вычисляет $Y = \min(|X|, 3)$ */

clauses


absolute(X, Y):- X < 0, Y = -X, !.

absolute(X, Y):- Y = X.

min(X, Y, Z):- X < Y, Z = X, !.

min(X, Y, Z):- Z = Y.

calculate(X, Res):- absolute(X, Xabs), min(Xabs, 3, Y), Res = Y.



goal

calculate(10, Res).

Решение Res = 3

Последовательные вычисления

predicates

absolute(integer, integer)

min(integer, integer, integer)

calculate(integer, integer) /* calculate(X, Y) вычисляет $Y = \min(|X|, 3)$ */

clauses

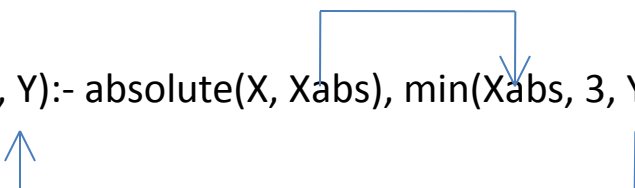
absolute(X, Y):- X < 0, Y = -X, !.

absolute(X, X).

min(X, Y, Z):- X < Y, Z = X, !.

min(X, Y, Y).

calculate(X, Y):- absolute(X, Xabs), min(Xabs, 3, Y).



goal

calculate(10, Res).

Решение Res = 3

Последовательные вычисления

predicates

absolute(integer, integer)

min(integer, integer, integer)

calculate(integer, integer) /* calculate(X, Y) вычисляет $Y = \min(|X|, 3)$ */

clauses

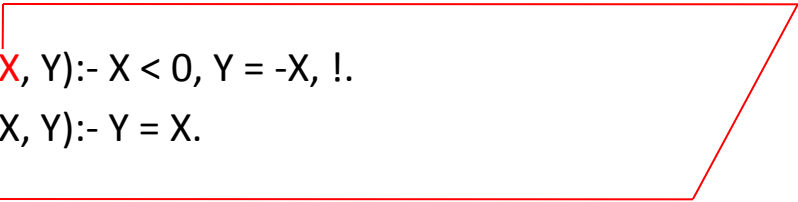
absolute(X, Y):- X < 0, Y = -X, !.

absolute(X, Y):- Y = X.

min(X, Y, Z):- X < Y, Z = X, !.

min(X, Y, Z):- Z = Y.

Это разные
переменные!



calculate(X, Res):- absolute(X, Xabs), min(Xabs, 3, Y), Res = Y.

goal

calculate(10, Res).

Реализация цикла через автоматический перебор вариантов

predicates

mass(real)

compose_mass(real, real, real)

clauses

mass(0.5).

mass(1.0).

mass(2.0).

mass(1.5).

compose_mass(MSum, M1, M2):- mass(M1), mass(M2), MSum = M1 + M2.

goal

compose_mass(3.0, M1, M2).

3 Решения:

M1 = 1.0, M2 = 2.0

M1 = 2.0, M2 = 1.0

M1 = 1.5, M2 = 1.5

Расчет длины списка (рекурсия)

domains

list = integer*

predicates

length_of(list, integer) /* 1st param – input, 2nd param – output */

clauses

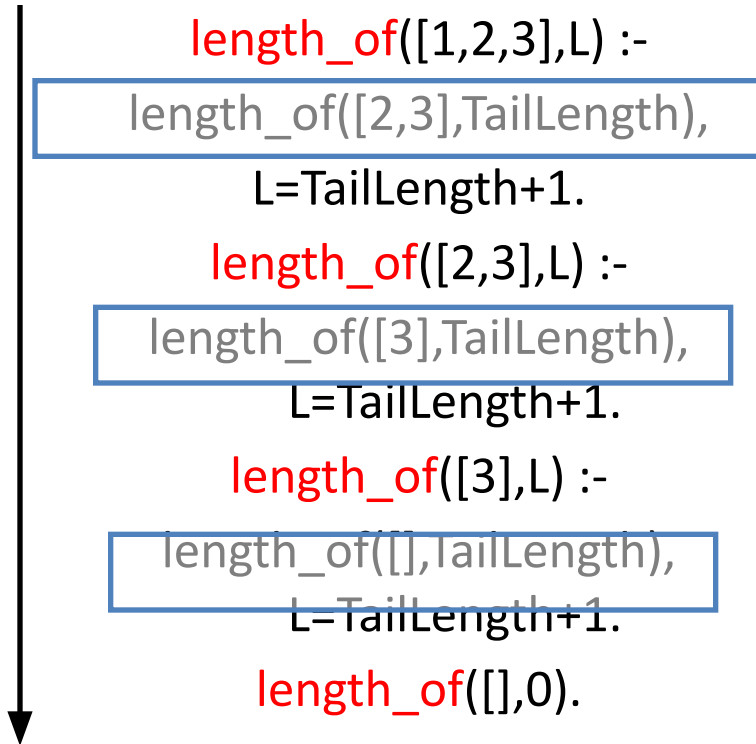
length_of([], L):- L = 0. /* условие окончания рекурсии */

length_of([H|T], L):- length_of(T, TailLength), L = TailLength + 1.

goal

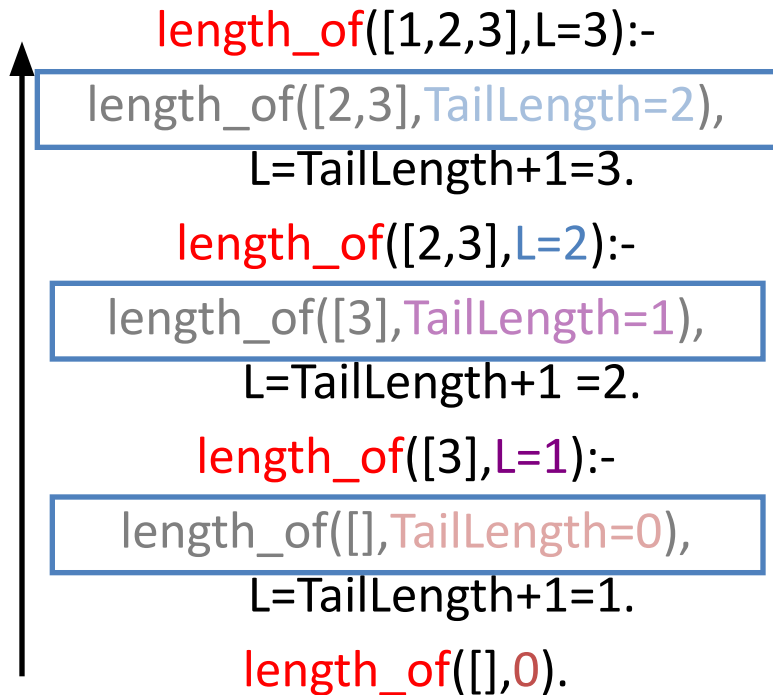
length_of([1,2,3], L).

Прямой проход рекурсии



- Здесь стрелка показывает направление рекурсивного вывода четырех подцелей, в результате чего подцель `length_of([],0)` была найдена.

Обратный проход рекурсии

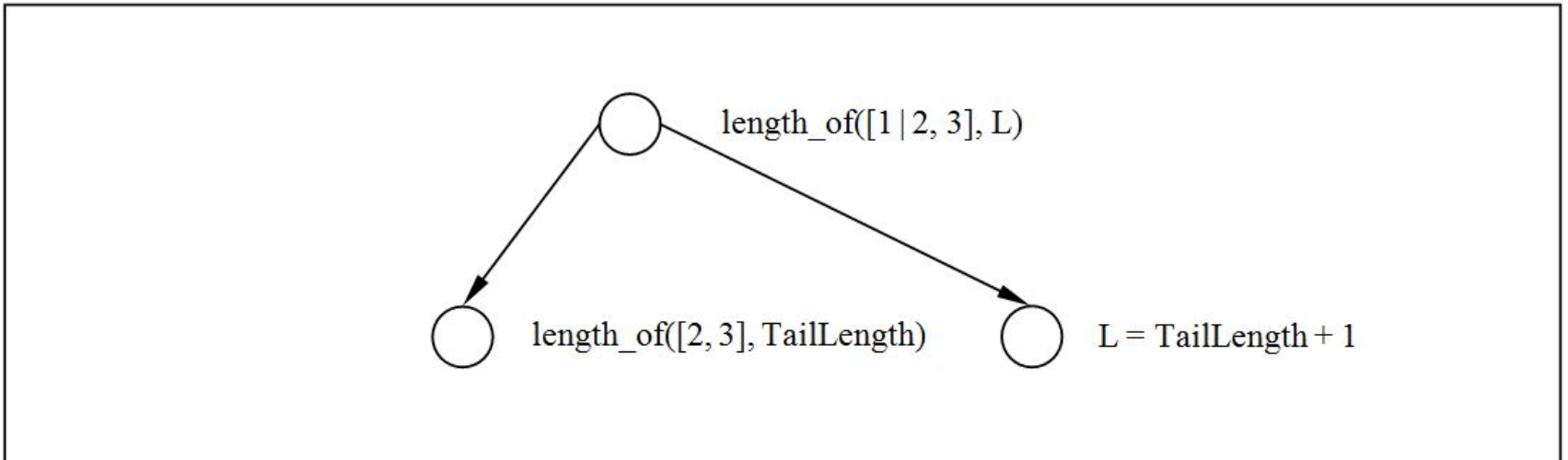


- Здесь стрелка показывает возврат назад к предыдущим подцелям от найденной подцели `length_of([],0)`.
- В процессе возврата на каждом шаге вычисляется значение `L` длины списка.
- В результате вычисляется длина `L=3`.
- Ответ: Yes, `L=3`.

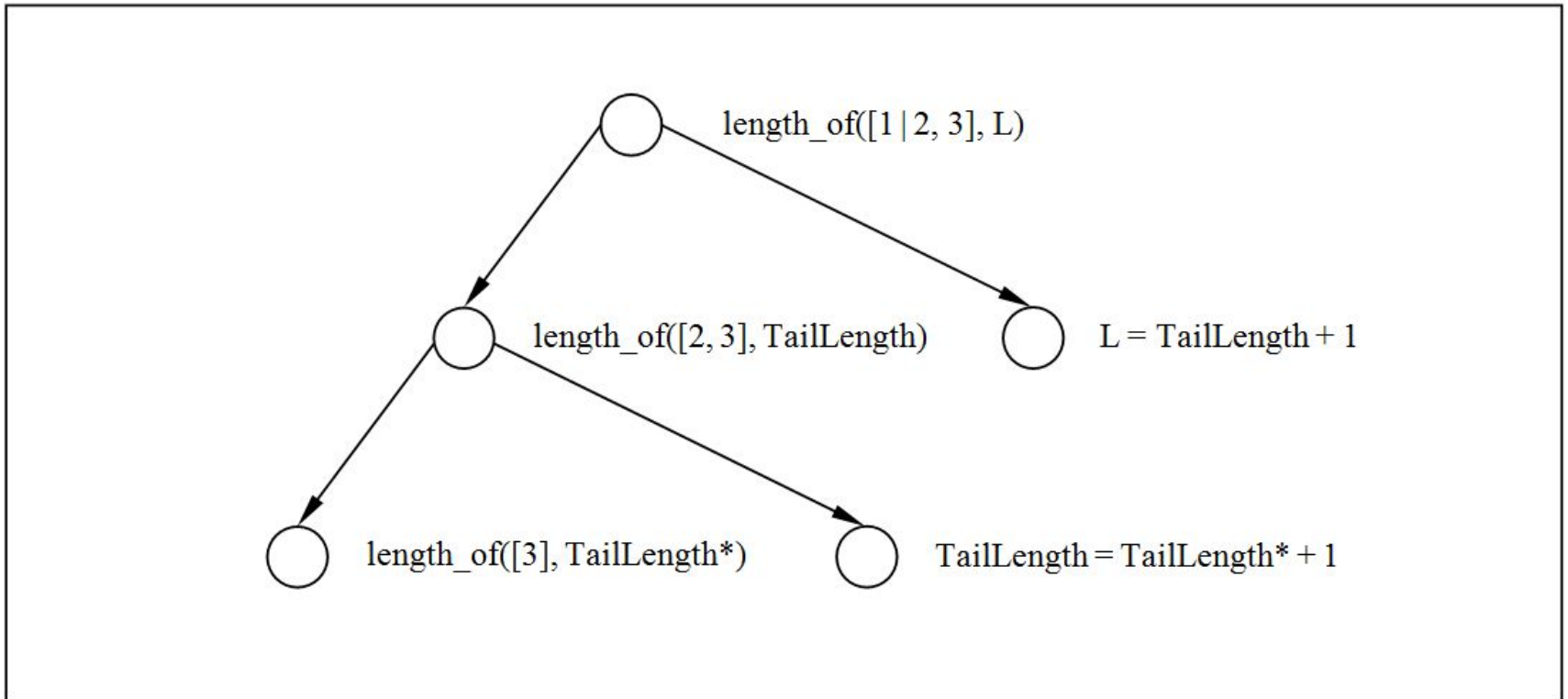
Шаг 0

length_of([1, 2, 3], L)

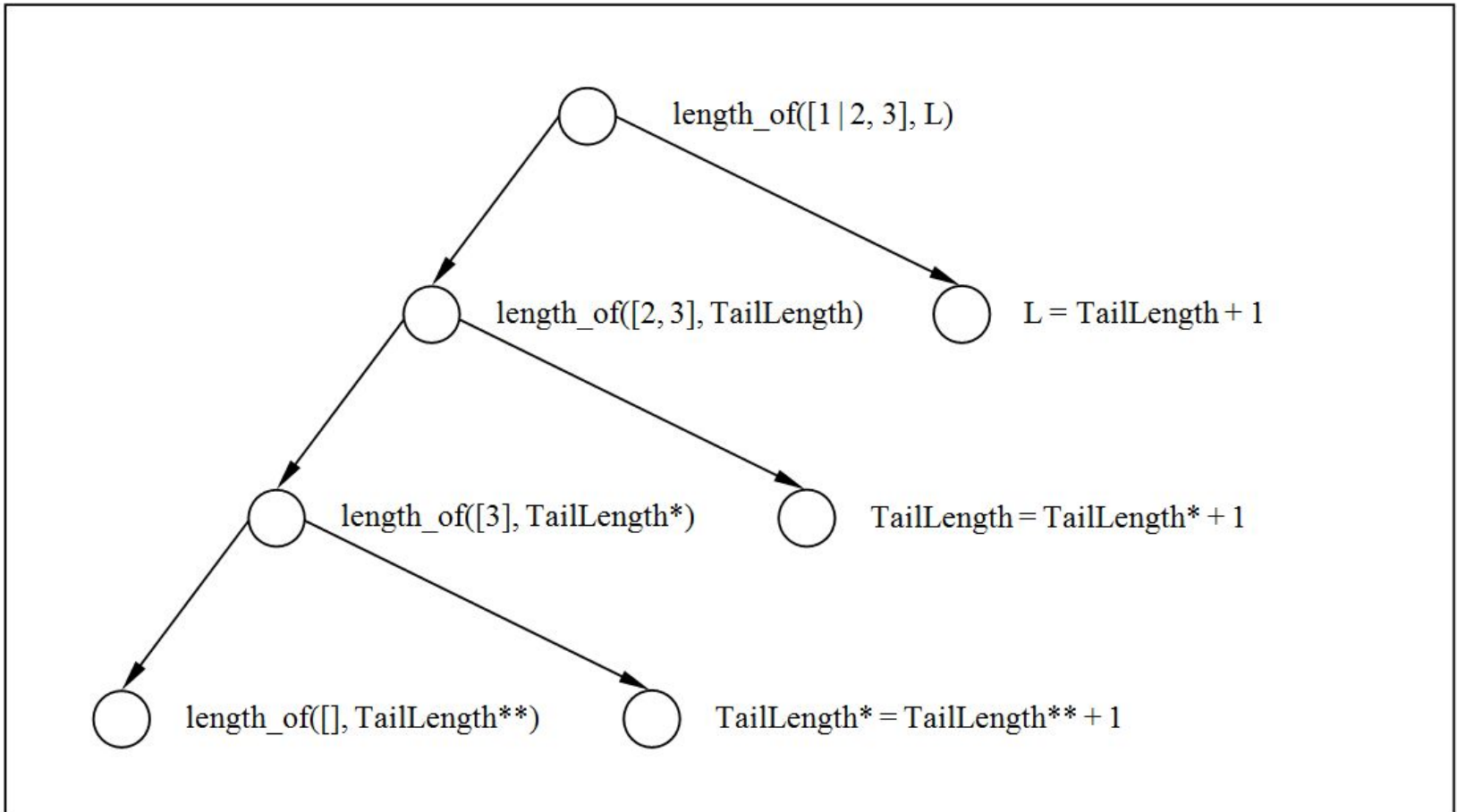
Шаг 1



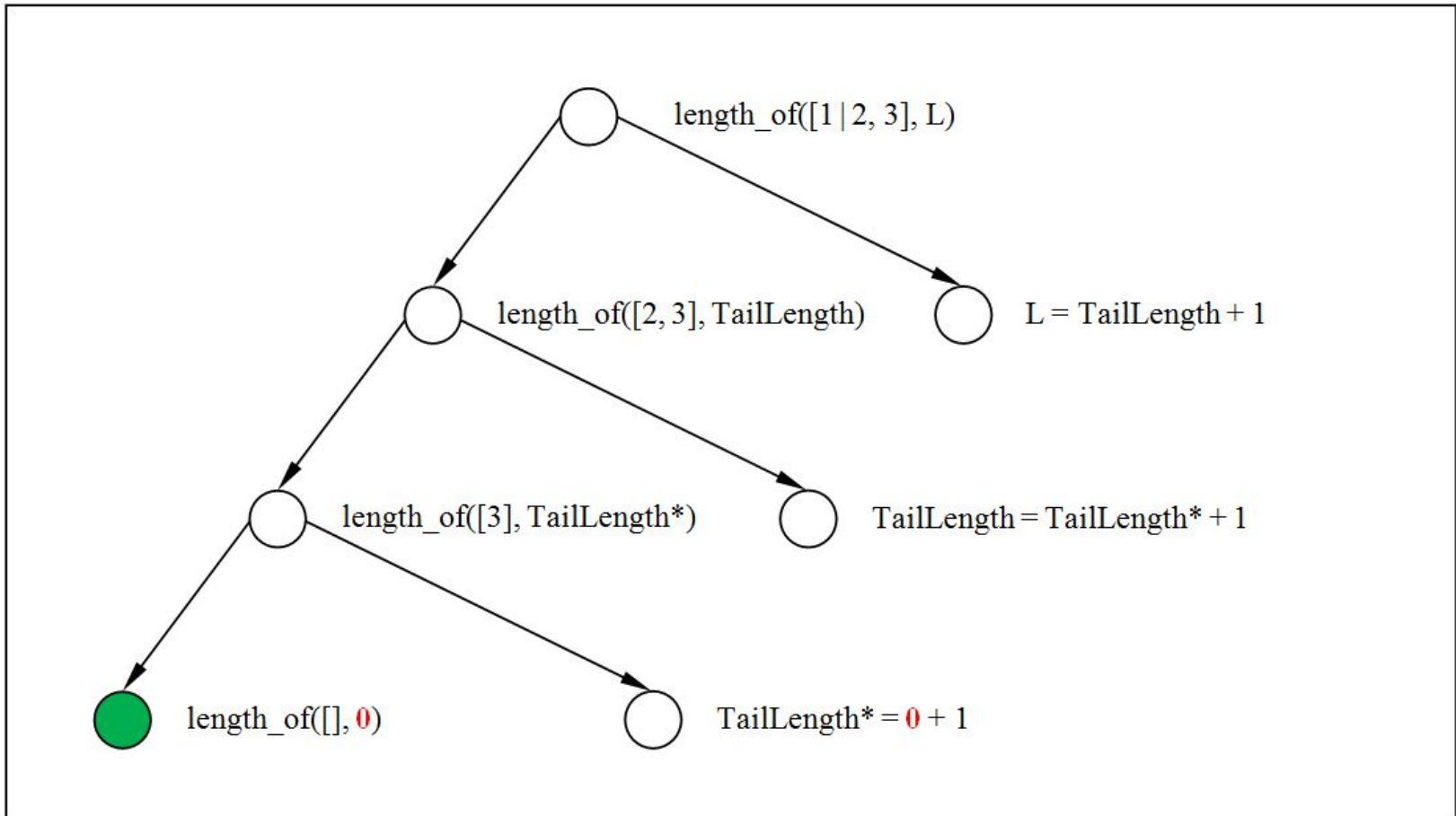
Шаг 2



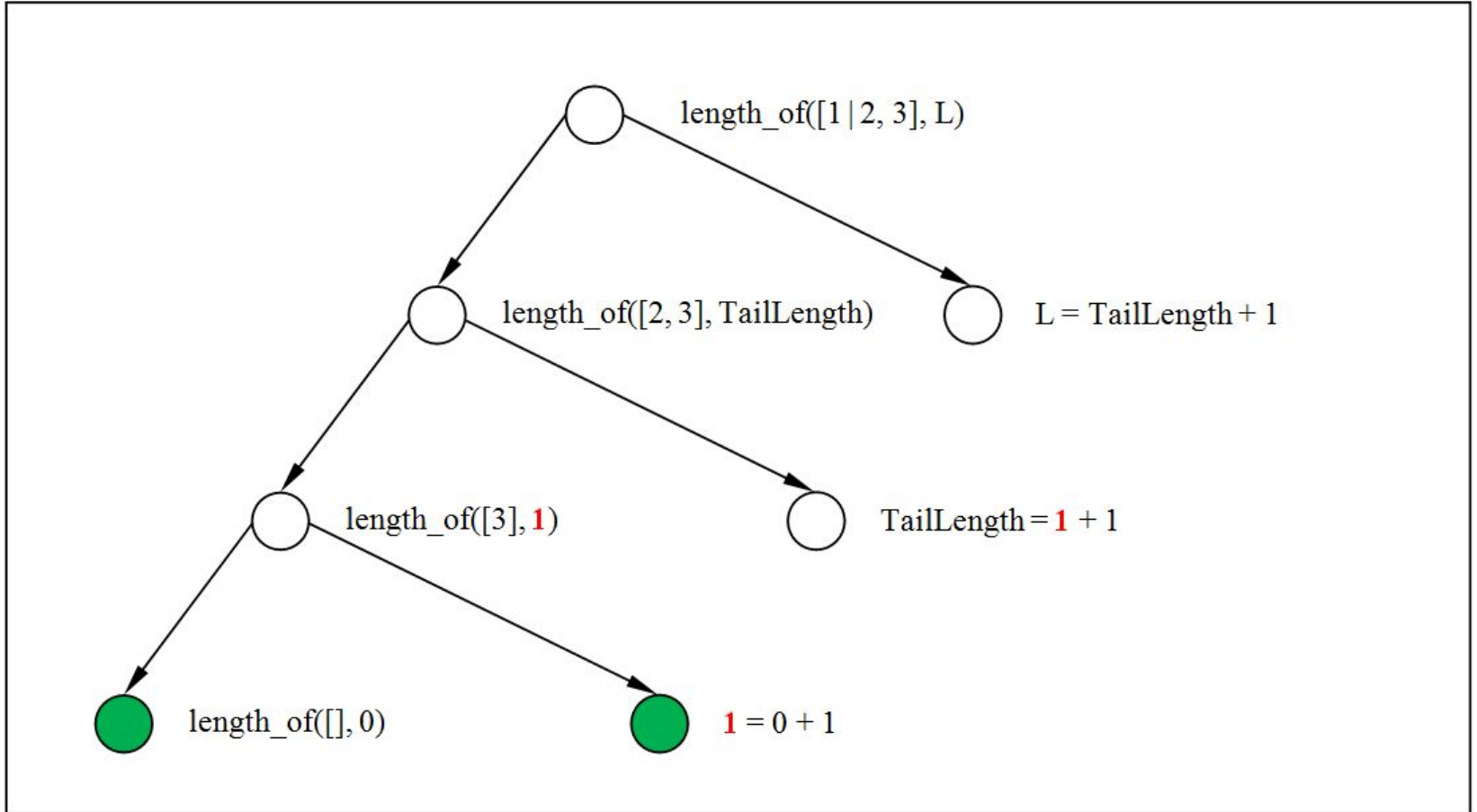
Шаг 3



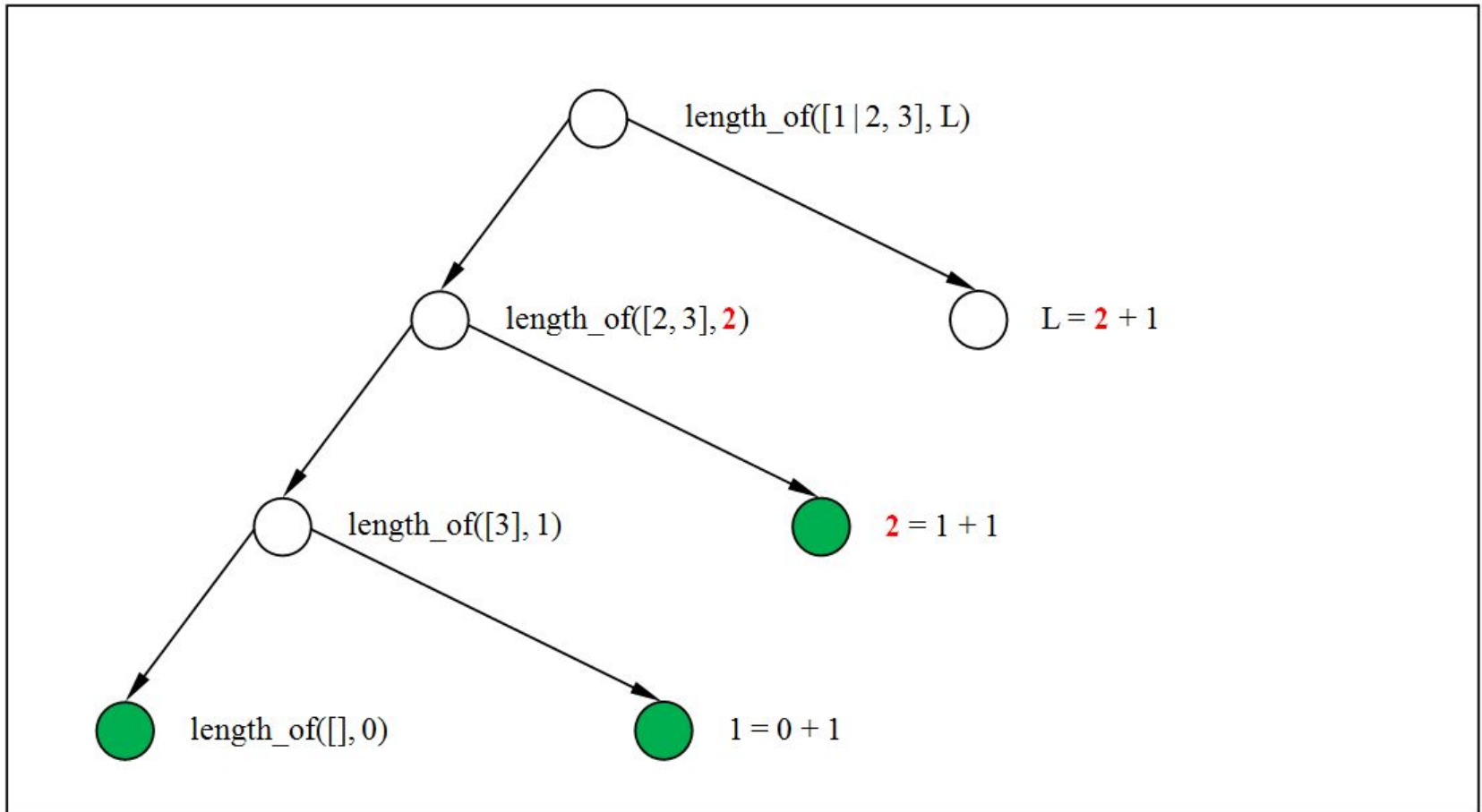
Шаг 4



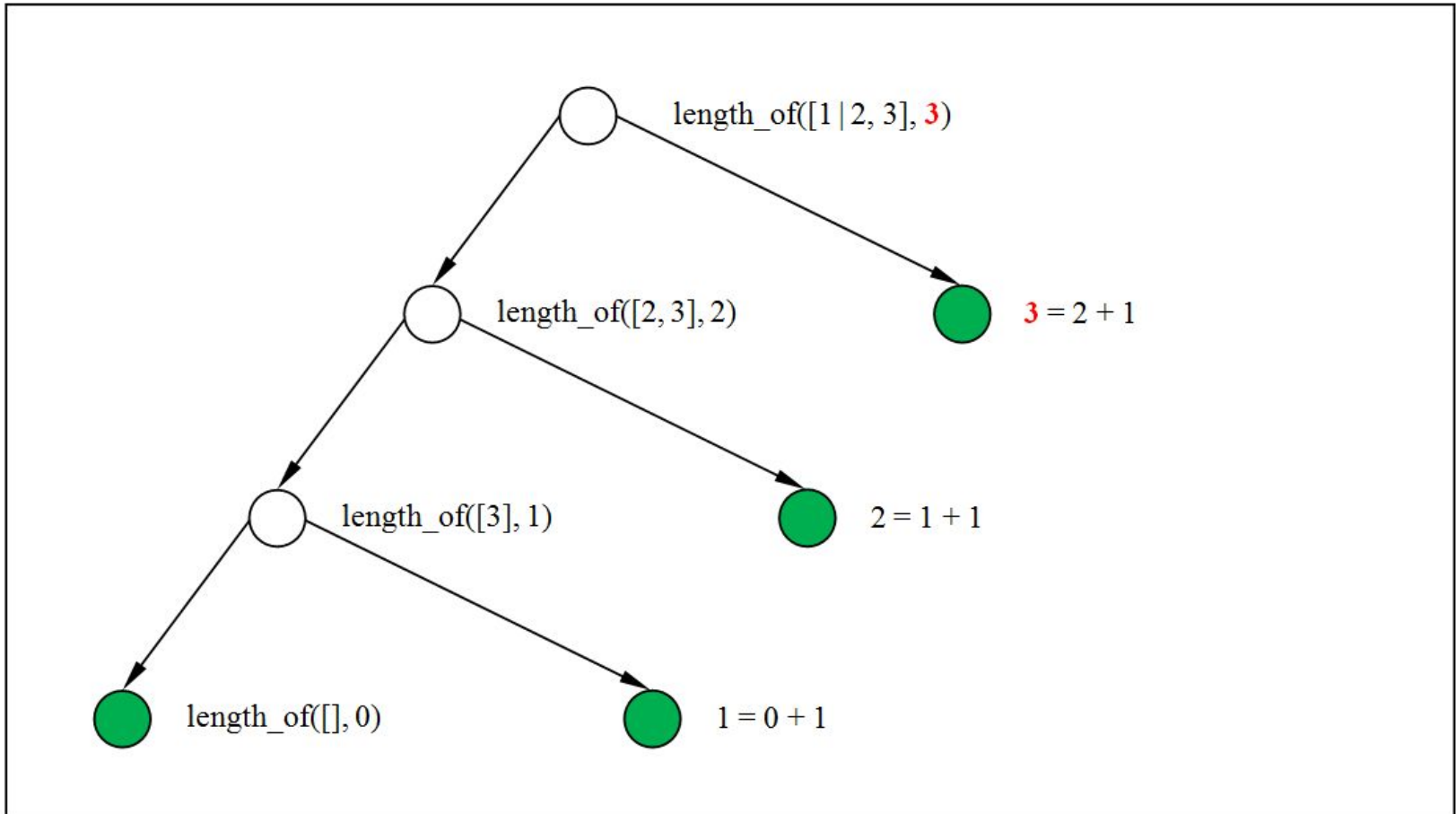
Шаг 5



Шаг 6



Шаг 7



Неверная реализация (1)

domains

list = integer*

predicates

length_of(list, integer) /* 1st param – input, 2nd param – output */

clauses

~~length_of([], L):- L = 0.~~

length_of([H|T], L):- length_of(T, TailLength), L = TailLength + 1.

goal

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

Нет решений

Неверная реализация (2)

domains

list = integer*

predicates

length_of(list, integer) /* 1st param – input, 2nd param – output */

clauses

length_of([], L):- L = 0.

length_of([H|T], L):- L = TailLength + 1, length_of(T, TailLength).

goal

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

Ошибка: Free variable in expression

Подсчет количества вхождений элемента в список

domains

list=integer*

predicates

count(integer, list, integer) % элемент, список, количество вхождений

clauses

count(X,[H|Tail],N):- X=H, count(X,Tail,Nt), N=Nt+1.

count(X,[H|Tail],N):- X<>H, count(X,Tail,Nt), N=Nt.

count(X, [], N):- N=0.

goal

count(5, [5,1,5], Res).

Решение: Res = 2

Составление списка целых чисел

domains

list=integer*

predicates

compose(integer, integer, list) % pervyi element, poslednii element, spisok

clauses

compose(Ns, Ne, List):- Ns > Ne, List = [].

compose(Ns, Ne, List):- Ns = Ne, List = [Ns].

compose(Ns, Ne, [H|Tail]):- Ns < Ne, H=Ns, Ns1 = Ns + 1, compose(Ns1, Ne, Tail).

goal

compose(1, 5, Res).



Решение: Res = [1,2,3,4,5].

Соединение двух списков

domains

list=integer*

predicates

concat(list, list, list) % pervyi spisok, vtoroy spisok, ob'edinennyi spisok

clauses

concat([], List2, ResList):- ResList = List2.

concat([H1 | Tail1], List2, [H1 | ResTail]):- concat(Tail1, List2, ResTail).

goal

concat([1,2,3], [4,5], Res).

Решение: Res = [1,2,3,4,5].

Особенности SWI Prolog (1)

- В Visual Prolog 5.0:
 $Y = X$
- В SWI Prolog:
 $Y = X$
- В Visual Prolog 5.0:
 $Y \langle \rangle X$
- В SWI Prolog:
 $\text{not}(Y = X)$

Особенности SWI Prolog (2)

- B Visual Prolog 5.0:
 $C = A + B$
- B SWI Prolog:
 $C \text{ is } A + B$
- B Visual Prolog 5.0:
 $C * C = A * A + B * B$
- B SWI Prolog:
 $X \text{ is } C * C, X \text{ is } A * A + B * B$

Особенности SWI Prolog (3)

- В Visual Prolog 5.0:
`write("X=", X).`
- В SWI Prolog:
`write("X="), write(X).`