Programowanie w logice

Kolokwium
27 maja 2015

1. (2+2)

   a) Proszę zdefiniować predykat `member(X,L)`, który jest spełniony, jeżeli `X` jest elementem listy `L`.

   b) Proszę rozszerzyć predykat `member(X,L)` tak, że jest spełniony, jeżeli `X` jest elementem listy `L` na jakimśkolwiek poziomie. Przykład:

   ```
   ?- member(4,[1,2,[3,4,[5]],[6,7]]).
   true.
   ```

2. (2+2)

   a) Proszę zdefiniować predykat `suffix(L1,L2)`, który jest spełniony, jeżeli lista `L1` jest końcem listy `L2`. Przykład:

   ```
   ?- suffix([1,2,3],[1,2,3,4,5,6]).
   no.
   ?- suffix([1,2,3],[3,4,5,1,2,3]).
   true.
   ```

   b) Proszę zdefiniować predykat `palindrom(L)`, który jest spełniony, jeżeli słowo w liście `L` jest palindromem. Przykład:

   ```
   ?- palindrom([a,b,a,a]).
   no.
   ?- palindrom([a,b,a,b,a]).
   true.
   ```

3. (2+2)

   a) Proszę zdefiniować predykat `split(X,L,L1,L2)`, który jest spełniony, jeżeli lista `L1` zawiera wszystkie elementy z listy `L` mniejsze albo równe `X`, a `L2` zawiera wszystkie elementy większe od `X`. Przykład:

   ```
   ?- split(5,[2,7,4,8,-1,5],L1,L2).
   L1 = [2,4,-1,5],
   L2 = [7,8].
   ```

   b) Proszę zdefiniować predykat `split(P,L,L1,L2)`, który jest spełniony, jeżeli lista `L1` zawiera wszystkie elementy `X` z listy `L` spełniające predykat `P`, a `L2` elementy niespełniające predykatu `P`. Przykład:

   ```
   ?- split(odd,[2,7,4,8,-1,5],L1,L2).
   L1 = [7,-1,5],
   L2 = [2,4,8].
   ```

4. (2+2+2)

   Drzewo binarne `D` można reprezentować przez term `nil` - puste drzewo - albo term `drzewo(X,L,P)` z elementem `X` i poddrzewami `L` i `P`. Proszę zdefiniować następujące predykaty dla drzew.

   a) `search(D,X)`, który jest spełniony, jeżeli `X` jest elementem drzewa `D`.

   b) `prod(D,P)`, który jest spełniony, jeżeli `P` jest iloczynem elementów drzewa `D`.

   c) `postorder(D,L)`, który jest spełniony, jeżeli lista `L` zawiera elementy drzewa `D` w kolejności postfiksowej.

5. (3+3)

   Dla następujących programów i pytań proszę podać drzewo SLD.

   a)
```
p(X,Y) :- q(X,Y).
p(X,Y) :- r(X,Y).
q(X,Y) :- s(X), !, t(Y).
r(c,d).
s(a).
s(b).
t(a).
t(b).

?- p(X,Y).
```

   b)
```
erase_pairs([],[]).
erase_pairs([X,X|L1],L2) :- erase_pairs(L1,L2).
erase_pairs([X|L1],[X|L2]) :- erase_pairs(L1,L2).

?- erase_pairs([1,2,2,2],L).
```