Metody programowania 2014

Lista zadań nr 5

Na zajęcia 1–3 kwietnia 2014

Zadanie 1 (1 pkt). Napisz predykat `flatten/2`, który „spłaszcza" listę:

```
?- flatten([1,[2,3,[4,5],6,[7,[8,9]]]], X).
X = [1,2,3,4,5,6,7,8,9]
```

Zadbaj o to, by Twój predykat nie generował nieużytków!

Zadanie 2 (1 pkt). Zaprogramuj predykat `halve(+X, -L, -R)` który kopiuje pierwszą połowę listy `X` do listy `L` i unifikuje zmienną `R` z drugą połową listy `X`. Lista `L` powinna zostać skopiowana, zaś lista `R` — współdzielona. Jeśli lista `X` ma nieparzystą liczbę elementów, to dłuższa powinna być lista `R`. *Wskazówka:* Nie musisz znać długości listy, żeby zlokalizować jej środek! Ustaw dwa wskaźniki (powiedzmy Y_1 i Y_2) na początek listy. W każdym kroku iteracji kopiuj element wskazywany przez Y_1 i przesuwaj Y_1 o jeden element listy, zaś Y_2 o dwa elementy listy. Zakończ iterację, gdy Y_2 dojdzie do końca listy.

Zadanie 3 (1 pkt). Zaprogramuj predykat `merge/3` który, zachowując porządek, łączy dwie listy liczb całkowitych uporządkowane rosnąco. Zadbaj o to, by predykat działał deterministycznie, a rekursja była ogonowa (tam, gdzie to niezbędne, użyj odcięć). *Wskazówka:* buduj listę poczynając od głowy i wybieraj jako jej kolejny element mniejszą spośród głów scalanych list. Gdy jedna z list się opróżni, zakończ iterację.

Użyj predykatów `halve/3` i `merge/3` do zdefiniowania predykatu `merge_sort/2` implementującego algorytm sortowania *Mergesort*.

Zadanie 4 (1 pkt). Ulepsz predykat `halve/2` z drugiego zadania tak, by nie kopiował także lewej połowy listy. Użyj do tego celu list różnicowych i zaprogramuj predykat:

```
halve(X--Y, L, R)
```

który podstawia pod zmienne `L` i `R` listy różnicowe `X--Z` i `Z--Y` mniej więcej równej długości. Zaprogramuj następnie algorytm *Mergesort* w postaci predykatu sortującego listy różnicowe `mergesort(X-Y, S)`, gdzie `S` jest posortowaną listą (zamkniętą).

Zadanie 5 (1 pkt). Zdefiniuj predykat

```
split(+List, +Med, -Small, -Big)
```

rozdzielający listę `List` liczb całkowitych na listy: `Small` — elementów mniejszych i `Big` — elementów nie mniejszych niż liczba `Med`. Wykorzystaj go do zdefiniowania predykatu `qsort/2` implementującego algorytm *Quicksort*. Uwaga: w fazie łączenia posortowanych list nie należy generować nieużytków — nie używaj więc do tego celu predykatu `append/3`:

```
qsort([], []).
qsort([H|T], R) :-
   split(T, H, S, B),
   qsort(S, S1),
   qsort(B, B1),
   append(S1, [H|B1], R).  % Źle! S1 staje
                           % się nieużytkiem!
```

Dodatkowy akumulator powinien wystarczyć. Jak zwykle uogólnij problem i zaprogramuj predykat `qsort(L,A,R)` tak, by `R` była połączeniem wyniku sortowania listy `L` z listą `A`.

Zadanie 6 (1 pkt). Rozważmy predykat

```
sum(X,Y,Z) :-
   Z is X + Y.
```

Ten predykat działa poprawnie tylko w trybie `(+,+,?)`. Zaprogramuj predykat, który działa poprawnie w każdym trybie, w tym `(-,-,-)`, np.

```
?- sum(2,3,X).
X = 5;
No
?- sum(X,4,6).
X = 2;
No
?- sum(X,Y,10).
X = 0, Y = 10;
X = 1, Y = 9;
X = -1, Y = 11;
X = 2, Y = 8;
X = -2, Y = 12
Yes
?- sum(X,Y,Z).
X = 0, Y = 0, Z = 0;
X = 1, Y = 0, Z = 1
Yes
```

Predykat powinien generować wszystkie rozwiązania całkowitoliczbowe (jest ich nieskończenie wiele). Np. przypadku zapytania `sum(X,Y,Z)`, gdzie `X`, `Y` i `Z` są nieukonkretnionymi zmiennymi, predykat powinien wygenerować każdą trójkę takich liczb (x, y, z), że $x + y = z$.

Zadanie 7 (1 pkt). Zaprogramuj predykat `prime/1` implementujący sito Eratostenesa, który działa poprawnie zarówno w przypadku generowania (tj. wywołany z nieukonkretnioną zmienną jako parametrem), jak i sprawdzania. W tym drugim przypadku jego argumentem może być dowolny term arytmetyczny (niekoniecznie literał całkowitoliczbowy). W przypadku wywołania z innym parametrem powinien zostać zgłoszony błąd arytmetyczny. Podczas przesiewania kandydat na liczbę pierwszą powinien być porównywany z wcześniej wygenerowanymi liczbami pierwszymi w kolejności od najmniejszej do największej. Użyj listy różnicowej aby sprawnie zaimplementować wstawianie liczby na koniec listy.