Programowanie 2010 Programming 2010

Lista zadań nr 5 Problem set no. 5

Na zajęcia 30 marca – 1 kwietnia 2010 Due March 30, 2010

Zadanie 1 (1 pkt). Zaprogramuj predykat `revall/2` odwracający listę wraz z podlistami (tutaj przez podlistę rozumiemy element listy będący listą):

Problem 1 (1 p). Define a predicate `revall/2` which reverses a list together with its sublists (here by a sublist we mean a member of a list being itself a list):

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

Zadbaj o to, by Twój predykat nie generował nieużytków!

Ensure your prediate does not generate any garbage!

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

Problem 2 (1 p). Define a predicate `flatten/2`, which "flattens" a list:

```
?- 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!

Ensure your prediate does not generate any garbage!

Zadanie 3 (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.

Problem 3 (1 p). Define a predicate `halve(+X, -L, -R)` which copies the first half of the list `X` to the list `L`, and unifies the variable `R` with the second half of the list `X`. The list `L` should be copied, but `R` — shared. If the list `X` has an odd length, the list `R` should be longer. *Hint:* you do not need to know the length of a list to locate its center! Set two pointers (say Y_1 and Y_2) to point at the beginning of the list. In every step of iteration copy the element that Y_1 points to, then move Y_1 by one element and Y_2 by two elements. Finish the iteration when Y_2 reaches the end of the list.

Zadanie 4 (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 przesuwając się w kierunku jej końca wybierając jako kolejny element mniejszą spośród głów scalanych list. Zakończ iterację, gdy jedna z list się opróżni.

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

Problem 4 (1 p). Define a predicate `merge/3` which, preserving the order of elements, combines two lists of elements sorted increasingly. Ensure that the predicate is deterministic and tail recursive (use cuts when needed). *Hint:* construct the list starting from its head and proceed toward its end choosing as the next element the smaller of the heads of the merged lists. Finish the iteration when one of the lists becomes empty.

Use the predicates `halve/3` and `merge/3` to define a predicate `merge_sort/2` that implements the *Mergesort* algorithm.

Zadanie 5 (1 pkt). Ulepsz predykat `halve`/2 z trzeciego zadania tak, by nie kopiował także lewej połowy listy. Użyj do tego celu list różnicowych i zaprogramuj predykat:

Problem 5 (1 p). Improve the predicate `halve`/2 from Problem 3 so that it does not make a copy of the left half of the list as well. Use difference lists and define a predicate:

```
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:

that substitutes for the variables `L` and `R` difference lists `X--Z` and `Z--Y` of roughly the same length. Next define the *Mergesort* algorithm in the form of a predicate for sorting difference lists:

```
mergesort(X-Y, S)
```

gdzie `S` jest posortowaną listą (zamkniętą).

where `S` is the sorted list (closed).

Zadanie 6 (1 pkt). Zdefiniuj predykat `split(+List, +Med, -Small, -Big)` który rozdziela 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:

Problem 6 (1 p). Define a predicate `split(+List, +Med, -Small, -Big)` which splits a list `List` of integers into lists: `Small` containing the elements which are less than, and `Big` — which are greater or equal to the integer `Med`. Use this predicate to define a predicate `qsort`/2 that implements the *Quicksort* algorithm. Warning: the concatenation of sorted lists should not generate any garbage — do not use the predicate `append`/3 for that purpose:

```
qsort([], []).
qsort([H|T], R) :-
   split(T, H, S, B),
   qsort(S, S1),
   qsort(B, B1),
   append(S1, [H|B1], R).  % Bad! S1 becomes garbage!
```

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`.

An additional accumulator should suffice. As usual generalize the problem and define a predicate `qsort(L,A,R)` so that `R` is the concatenation of the result of sorting the list `L` with the list `A`.