Programowanie 2010

Lista zadań nr 2

Na zajęcia 9–11 marca 2010

Zadanie 1 (1 pkt). Zadaj Prologowi następujące pytania:

1. Jakie pary list mają tę własność, że druga jest wynikiem konkatenacji dwóch kopii pierwszej z nich?
2. Jaki element należy usunąć z listy `[a,b,c,d]` by otrzymać listę `[a,c,d]`?
3. Jaką listę należy dołączyć do listy `[a,b,c]`, by otrzymać listę `[a,b,c,d,e]`?

Zadanie 2 (1 pkt). Zaprogramuj w Prologu poniższe predykaty:

1. `even(X)`, spełniony gdy lista `X` ma parzystą liczbę elementów;
2. `palindrom(X)`, spełniony gdy lista `X` jest palindromem;
3. `singleton(L)`, spełniony gdy `X` jest listą jednoelementową.

Zadanie 3 (1 pkt). Zaprogramuj w Prologu poniższe predykaty:

1. `head`(H, L), spełniony gdy H jest pierwszym elementem (głową) listy L;
2. `last`(L, H), spełniony gdy H jest ostatnim elementem listy L;
3. `tail`(T, L), spełniony gdy T jest listą wszystkich z wyjątkiem pierwszego elementów (ogonem) listy L;
4. `init`(L, T), spełniony gdy T jest listą wszystkich z wyjątkiem ostatniego elementów listy L;
5. `prefix`(P, L), spełniony gdy P jest listą początkowych elementów (prefiksem) listy L;
6. `suffix`(L, S), spełniony gdy S jest listą końcowych elementów (sufiksem) listy L.

Programming 2010

Problem set no. 2

Due March 9, 2010

Problem 1 (1 p). Ask Prolog the following questions:

1. Which pairs of lists have the following property: the second one is the result of concatenation of two copies of the first one?
2. Which element has to be removed from the list `[a,b,c,d]` to obtain the list `[a,c,d]`?
3. What list has to be appended to the list `[a,b,c]` to obtain the list `[a,b,c,d,e]`?

Problem 2 (1 p). Define in Prolog the predicates listed below:

1. `even(X)`, satisfied when the list `X` is of even length;
2. `palindrom(X)`, satisfied when the list `X` is a palindrome,
3. `singleton(X)`, satisfied when `X` is a list containing exactly one element.

Problem 3 (1 p). Define in Prolog the predicates listed below:

1. `head`(H, L), satisfied when H is the first element (the head) of the list L;
2. `last`(L, H), satisfied when H is the last element of the list L;
3. `tail`(T, L), satisfied when T is the list of all but the first of the elements (the tail) of the list L;
4. `init`(L, T), satisfied when T is the list of all but the last of the elements of the list L;
5. `prefix`(P, L), satisfied when P is the list of leading elements (the prefix) of the list L;
6. `suffix`(L, S), satisfied when S is the list of trailing elements (the suffix) of the list L.

Zadanie 4 (1 pkt). Zaprogramuj w Prologu taki predykat `sublist`/2, że cel `sublist`(l_1, l_2) jest spełniony, gdy lista l_2 jest *podlistą* listy l_1. Przy kolejnych nawrotach predykat powinien wygenerować wszystkie podlisty w dowolnej kolejności. Podlistą listy $[x_1, \ldots, x_n]$ nazywamy dowolną listę postaci $[x_{i_1}, \ldots, x_{i_k}]$ dla $1 \leq i_1 < \ldots < i_k \leq n$. Podlista powstaje zatem przez usunięcie z listy niektórych elementów. Lista n-elementowa ma 2^n podlist.

Zadanie 5 (1 pkt). *Permutacją* listy $[x_1, \ldots, x_n]$ nazywamy dowolną listę postaci $[x_{i_1}, \ldots, x_{i_n}]$, gdzie $i_1, \ldots, i_n$ są parami różnymi liczbami z przedziału $1, \ldots, n$. Permutacja powstaje zatem przez przestawienie kolejności elementów na liście. Lista n-elementowa ma $n!$ permutacji.

Permutacje można generować *przez wybieranie*, zgodnie z następującym schematem rekurencyjnym:

- Jedyną permutacją listy pustej jest lista pusta.
- Aby wygenerować permutację pewnej niepustej listy, wybierz z niej dowolny element. Wybrany element będzie głową tworzonej permutacji. Jej ogonem będzie natomiast permutacja listy pozostałej po wybraniu tego elementu.

Zaprogramuj predykat `perm`/2 implementujący powyższy algorytm.

Zadanie 6 (1 pkt). Permutacje można także generować *przez wstawianie*, zgodnie z następującym schematem rekurencyjnym:

- Jedyną permutacją listy pustej jest lista pusta.
- Aby wygenerować permutację pewnej niepustej listy, wygeneruj dowolną permutację jej ogona i następnie wstaw jej głowę w dowolne miejsce w wygenerowanej wcześniej permutacji.

Zaprogramuj predykat `perm`/2 implementujący powyższy algorytm.

Problem 4 (1 p). Define in Prolog a predicate `sublist`/2 such that the goal `sublist`(l_1, l_2) is satisifed when the list l_2 is a *sublist* of the list l_1. During backtracking the predicate should generate all sublists in an arbitrary order. A sublist of the list $[x_1, \ldots, x_n]$ is any list of the form $[x_{i_1}, \ldots, x_{i_k}]$ for $1 \leq i_1 < \ldots < i_k \leq n$. A sublist is thus created by removing some elements from the given list. An n-element list has 2^n sublists.

Problem 5 (1 p). We call a *permutation* of the list $[x_1, \ldots, x_n]$ any list of the form $[x_{i_1}, \ldots, x_{i_n}]$, where $i_1, \ldots, i_n$ are pairwise different numbers from the set $1, \ldots, n$. A permutation is thus created by changing the order of elements in the given list. An n-element list has $n!$ permutations.

Permutations can be generated *by selection*, according to the following recursive scheme:

- The only permutation of the empty list is the empty list.
- In order to generate a permutation of some nonempty list, pick an arbitrary element from it. The selected element will be the head of the created permutation. Its tail is a permutation of the list left after selecting this element.

Define a predicate `perm`/2 which implements the algorithm stated above.

Problem 6 (1 p). Permutations can also be generated *by insertion*, according to the following recursive scheme:

- The only permutation of the empty list is the empty list.
- In order to generate a permutation of some nonempty list, generate an arbitrary permutation of its tail, then insert its head into an arbitrary position in the generated permutation.

Define a predicate `perm`/2 which implements the algorithm stated above.