Programowanie 2010 Programming 2010

Lista zadań nr 6 Problem set no. 6

Na zajęcia 13–15 kwietnia 2010 Due April 13, 2010

Zadanie 1 (1 pkt). Rozważmy predykat

Problem 1 (1 p). Consider a predicate

```
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.

This predicate works correctly only in the mode `(+,+,?)`. Define a predicate that works correctly in any mode, including `(-,-,-)`, *e.g.*:

```
?- 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 $x + y = z$.

The predicate should gereate all integer solutions (there are infinitely of them). *E.g.*, in case of the query `sum(X,Y,Z)`, where `X`, `Y` and `Z` are uninstantiated variables, the redicate should generate every such triple of numbers (x, y, z), that $x + y = z$.

Zadanie 2 (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.

Problem 2 (1 p). Define a predicate `prime/1` that implements the sieve of Erathostenes and works correctly for generating (when called with uninstantiated variable as a parameter), and for checking as well. In the second case it can be called with any arithmetic term, not necessarily a number. When called with other kinds of arguments it should report the arithmetic error. During sieving the candidate for the next prime number should be compared with previously generated primes, from the smallest to the biggest. Use difference lists to implement the insertion into the end of a list efficiently.

Zadanie 3 (1 pkt). *Kolekcje* to struktury danych służące do przechowywania elementów. Niech interfejs kolekcji składa się z następujących nazw predyktów:

- `put(+E,+S,-R)` wstawia element `E` do kolekcji `S` i zwraca nową kolekcję `R`;
- `get(+S,-E,-R)` usuwa element z kolekcji `S`, podstawia go pod `E` i zwraca nową kolekcję `R`;
- `empty(?S)` sprawdza lub tworzy pustą kolekcję;
- `addall(-E, +G, +S, -R)` wstawia wszystkie wyniki podstawień pod zmienną `E`, które spełniają cel `G` (w którym zmienna `E` występuje) do kolekcji `S` i zwraca nową kolekcję `R` (ten predykat przypomina standardowe predykaty `findall`/3 i `findall`/4).

Podaj dwie implementacje tego interfejsu, jedną dla stosu (użyj list zamkniętych do reprezentowania kolekcji), drugą dla kolejek FIFO (tu użyj list różnicowych).

Zadanie 4 (1 pkt). Rozważmy skierowany graf $G = \langle V, E\rangle$. Jego wierzchołki ze zbioru V będziemy reprezentować w postaci atomów i liczb, a relację krawędzi E — za pomocą binarnego predykatu `e`/2. Dane są wierzchołki $v_1, v_2 \in V$. Ścieżkę z wierzchołka v_1 do wierzchołka v_2 w grafie G można znaleźć używając algorytmu przeszukiwania sparametryzowanego kolekcją przechowującą oczekujące wierzchołki:

1. Jeśli kolekcja przechowująca oczekujące wierzchołki jest pusta, to zakończ pracę.
2. W przeciwnym razie wyjmij wierzchołek v z kolekcji. Jeśli v nie został wcześniej odwiedzony, to odwiedź go i wstaw wszystkich jego `e`-sąsiadów do kolekcji. W przeciwnym razie odrzuć wierzchołek v. Powtórz całą procedurę.

Zaprogramuj powyższy algorytm tak, by znajdował w grafie ścieżki między podanymi wierzchołkami używając interfejsu kolekcji z poprzedniego zadania. Następnie użyj stosów, by otrzymać DFS oraz kolejek, by otrzymać BFS.

Przypuśćmy, że wierzchołkom grafu dodajemy *wagi* i używamy kolejek priorytetowych do przechowywania oczekujących wierzchołków. Zbadaj zachowanie takiego algorytmu.

Problem 3 (1 p). *Collections* are data structures for storing elements. Let the interface for collections consists of the following predicate names:

- `put(+E,+S,-R)` inserts an element `E` to a collection `S` and returns a new collection `R`;
- `get(+S,-E,-R)` removes an element from a collection `S`, substitutes this element for `E` and returns a new collection `R`;
- `empty(?S)` checks if a collection is empty or generates an empty collection;
- `addall(-E, +G, +S, -R)` adds all results of substitutions for the variable `E` which satisfy the goal `G` (in which the variable `E` occurs) to a collection `S` and returns a new collection `R` (this predicate resembles the standard predicates `findall`/3 and `findall`/4).

Give two implementations of this interface, one for stacks (use closed lists to represent collections), the other one for FIFO queues (use difference lists).

Problem 4 (1 p). Consider a directed graph $G = \langle V, E\rangle$. We will represent vertices from V using atoms or numbers and the edge relation E using the binary predicate `e`/2. Given two vertices $v_1, v_2 \in V$. We can find a path from v_1 to v_2 in G using the searching procedure parametrized with a collection storing pending vertices:

1. If the collection for pending vertices is empty, then we are done.
2. Otherwise get a vertex v from this collection. If v has not been visited, then visit it and add all its `e`-neighbours to the collection. Otherwise throw v away. Repeat the procedure.

Implement this algorithm so that it finds a path between given vertices using the interface for collections from the previous problem. Then use stacks to obtain DFS, and queues to obtain BFS.

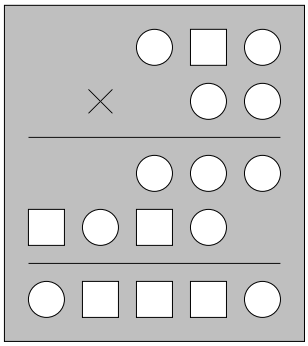
Let us add *weights* to vertices and use the priority queues for storing pending vertices. Research the behaviour of this algorithm.

Zadanie 5 (1 pkt). W tym zadaniu, inaczej niż w zadaniach z poprzedniej listy, będziemy zakładać, że drzewa poszukiwań o etykietowanych wierzchołkach wewnętrznych nie zawierają dwóch równych etykiet. Zaprogramuj następujące predykaty: `insert`/3 — wstawiający element do drzewa (bez powtórzeń), `find`/2 — sprawdzający, czy element znajduje się w drzewie, `findMax`/2 — ujawniający największy element w drzewie, `delMax`/3 — ujawniający i usuwający największy element z drzewa, `delete`/3 — usuwający podany element z drzewa oraz `empty`/1 — sprawdzający, czy drzewo jest puste.

Problem 5 (1 p). In this problem, unlike in problems from previous list, we will assume that binary search trees with labelled internal nodes do not contain two equal labels. Define predicates: `insert`/3 which inserts an element to a tree (without repetitions), `find`/2 which checks if an element occurs in a tree, `findMax`/2 which finds the biggest element in a tree, `delMax`/3 which returns and removes the biggest element from a tree, `delete`/3 which removes a particular element from a tree, and `empty`/1 which checks if a tree is empty.

Zadanie 6 (1 pkt). Napisz w Prologu program rozwiązujący zadania takie jak to, które pochodzi z numeru 11/2000 miesięcznika „Wiedza i Życie”: *Wpisz w kwadraty cyfry parzyste (0, 2, 4, 6, 8), w koła zaś nieparzyste (1, 3, 5, 7, 9) tak, by otrzymać poprawny zapis mnożenia:*

Problem 6 (1 p). Write in Prolog a program solving problems similar to the following one, which appeared in the 11/2000 issue of the monthly magazine “Knowledge & Life”: *Fill in the squares with even digits (0, 2, 4, 6, 8) and the circles with odd digits (1, 3, 5, 7, 9) in a way to obtain a correct multiplication chart:*



Dane do programu (opis problemu) należy podać w postaci listy list atomów `s` i `c` reprezentujących kwadraty i koła. Np. dane do problemu z obrazka są następujące:

Input data (description of a problem) should be given in the form of a list of lists of atoms `s` and `c` representing squares and circles, respectively. E.g., the data from the figure are the following:

```
[[c,s,c], [c,c], [c,c,c], [s,c,s,c], [c,s,s,s,c]]
```