Programowanie 2010 Programming 2010

Lista zadań nr 4 Problem set no. 4

Na zajęcia 23–25 marca 2010 Due March 23, 2010

Zadanie 1 (1 pkt). Zaprogramuj w Prologu predykat `length`/2, który działa poprawnie we wszystkich trybach użycia, w szczególności nie zapętla się dla celu `length(X,5)` (ten cel powinien być spełniony dokładnie na jeden sposób z podstawieniem listy `[_,_,_,_,_]` pod `X`).

Problem 1 (1 p). Define in Prolog a predicate `length`/2 which works correctly in any mode, in particular does not fall into endless loop for the goal `length(X,5)` (this goal should be satisfied exactly once with the list `[_,_,_,_,_]` substituted for `X`).

Zadanie 2 (1 pkt). Niech predykat `connection`/2 (zdefiniowany przez zbiór faktów) oznacza, że istnieje bezpośrednie połączenie między dwoma miastami. Zdefiniuj predykat `trip`/3, który znajduje połączenie między dwoma miastami z dowolną liczbą przesiadek i nie zapętla się nawet wówczas, gdy w grafie połączeń są cykle. Pierwsze dwa parametry tego predykatu (wejściowe), to miasta początkowe i końcowe. Trzeci parametr (wyjściowy), to lista miast od początkowego do końcowego, przez które przebiega podróż, np.:

Problem 2 (1 p). Let the predicate `connection`/2 (defined by a set of facts) mean, that there is a direct connection between two cities. Define a predicate `trip`/3 which finds a connection between two cities with an arbitrary number of changes and does not loop even if there are loops in the connection graph. First two parameters of this predicate (input) stand for the starting and ending cities. The third parameter (output) is a list of cities from the starting to the ending city by which the trip goes, eg.:

```
?- trip(gliwice, warszawa, T).
T = [gliwice, wroclaw, warszawa] ;
T = [gliwice, wroclaw, katowice, warszawa] ;
false.
```

Wskazówka: zdefiniuj wpierw predykat `trip`/4, którego dodatkowy parametr jest „akumulatorem” — listą zawierającą już odwiedzone miasta. Użyj predykatu `member`/2 by sprawdzić, czy miasto, do którego chcemy przejść, było już wcześniej odwiedzone. Ścieżkę buduj od końca ku początkowi, by uniknąć konieczności odwracania listy po znalezieniu rozwiązania.

Hint: first define a predicate `trip`/4. Its additional parameter should be an “accumulator” — a list containing cities already visited. Use the predicate `member`/2 to find out if a city we want to go has already been visited. Build the path from the end toward the beginning to avoid reversing the list when the solution is found.

Zadanie 3 (1 pkt). Rozważmy binarne rozwinięcia liczb naturalnych, takie, że binarnym rozwinięciem liczby 0 jest ciąg złożony z pojedynczej cyfry 0, a rozwinięcia liczb dodatnich nie zawierają zer nieznaczących (zatem ich najbardziej znaczącą cyfrą zawsze jest 1). Napisz w Prologu predykaty `bin/1` oraz `rbin/1` generujące binarne rozwinięcia kolejnych liczb naturalnych reprezentowane w postaci list literałów `0` i `1`. Predykat `bin` powinien tworzyć listy cyfr dwójkowych w kolejności od najbardziej do najmniej znaczącej pozycji, `rbin` zaś — odwrotnie.

Problem 3 (1 p). Consider binary representations of natural numbers where the number 0 is represented by the sequence consisting of a single digit 0, and the representations of positive numbers do not contain leading zeros (so their most significant digit is always 1). Write in Prolog predicates `bin/1` and `rbin/1` that generate binary representations of consecutive natural numbers expressed in the form of lists of literals `0` and `1`. The `bin` predicate should generate the lists of binary digits from the most to the least significant one, and `rbin` — the other way round.

Przeciętna liczba wnioskowań pomiędzy kolejnymi sukcesami powinna być ograniczona przez stałą (o to, ile wynosi ta stała, możecie zapytać artystkę uliczną Magdę z zajęć wyrównawczych z matematyki[1]), nie wolno więc kopiować ani odwracać generowanych list. Nie wolno używać żadnych predykatów standardowych. Wolno definiować własne predykaty pomocnicze. Pomimo iż sformułowanie zadania odwołuje się do pojęcia liczby, nie należy używać prologowej arytmetyki.

Wynikiem zapytania `bin(X)` powinien być ciąg odpowiedzi:

The average number of inferences between two successes should be bounded by a constant (you can ask the street performer Magda from the introductory math course how big is this constant), so you cannot copy or reverse the lists that you generate. You cannot use any standard predicates, but you can define your own auxiliary predicates. Even though the definition of the problem refers to the concept of a natural number, you cannot use the Prolog arithmetic.

The query `bin(X)` should be answered in the following way:

```
X = [0];
X = [1];
X = [1,0];
X = [1,1];
X = [1,0,0];
X = [1,0,1];
...
```

zaś wynikiem zapytania `rbin(X)` powinien być ciąg odpowiedzi:

and the query `rbin(X)` should be answered in the following way:

```
X = [0];
X = [1];
X = [0,1];
X = [1,1];
X = [0,0,1];
X = [1,0,1];
...
```

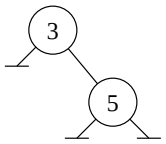
Zadanie 4 (1 pkt). *Drzewa binarne o etykietowanych wierzchołkach wewnętrznych* zapisujemy w Prologu używając funktora `node`/3 do reprezentowania wierzchołków wewnętrznych i atomu `leaf`/0 do reprezentowania liści. Dla przykładu struktura:

Problem 4 (1 p). We represent *binary trees with labelled internal nodes* in Prolog using the functor `node`/3 for internal nodes and the atom `leaf`/0 for leaves. E.g., the structure:

```
node(leaf, 3, node(leaf, 5, leaf))
```

reprezentuje drzewo:

represents the tree:



Zaprogramuj predykaty `mirror`/2 — tworzący lustrzane odbicie drzewa oraz `flatten`/2 — tworzący listę etykiet drzewa w porządku infiksowym.

Define predicates `mirror`/2, which creates the mirror image of a tree and `flatten`/2, which creates a list of the tree labels in the infix order.

[1] Dopiero po zakończeniu wrześniowego kursu okazało się, że Magda, to tylko pseudonim artystyczny, a w rzeczywistości artystka pochodzi zza wschodniej granicy i ma na imię Tania (*wskazówka:* średnio 2 zł. za występ).

Zadanie 5 (1 pkt). *Binarne drzewa poszukiwań* będziemy zapisywać podobnie, jak w poprzednim zadaniu. Zaprogramuj predykat `insert`/3 — wstawiający element do drzewa (z powtórzeniami). Zaprogramuj następnie algorytm *Treesort*, który wstawia wszystkie elementy listy do drzewa poszukiwań i następnie spłaszcza drzewo otrzymując posortowaną listę.

Problem 5 (1 p). We will represent *binary search trees* as in the previous problem. Define a predicate `insert`/3, which inserts an element to a tree (with repetitions). Next program the *Treesort* algorithm which inserts all the elements of a list into a tree and then flattens the tree to obtain the sorted list.

Zadanie 6 (1 pkt). Napisz zapytanie prologowe, które rozwiąże następującą łamigłówkę:

Problem 6 (1 p). Write a Prolog query that solves the following puzzle:

```
          U  S  A
+      U  S  S  R
-----------------
=  P  E  A  C  E
```

Zapytanie powinno zawierać zmienne `A`, `C`, `E`, `P`, `R`, `S` i `U` oraz powinno sprawdzić wszystkie możliwe podstawienia pod te zmienne parami różnych cyfr `0`, `1`, `2`, `3`, `4`, `5`, `6`, `7`, `8` i `9`. Cyfry podstawione pod zmienne `U` i `P` nie mogą być zerem. W zapytaniu możesz użyć następujących predykatów standardowych:

The query should contain variables `A`, `C`, `E`, `P`, `R`, `S` and `U`, and should try all possible substitutions for these variables pairwise different digits `0`, `1`, `2`, `3`, `4`, `5`, `6`, `7`, `8` and `9`. Digits substituted for variables `U` and `P` should differ from zero. You can use the following standard predicates in your query:

`permutation`/2, `length`/2, `is`/2, `\=`/2,

predykatu `concat_number`/2 z listy 3 i predykatu `sublist/2` z listy 2. Zapytanie powinno zwrócić dokładnie jedną odpowiedź, bez powtórzeń:

the predicate `concat_number`/2 from the list no. 3, and the predicate `sublist/2` the list no. 2. The query should return exactly one answer, without repetitions:

```
?- the query (omitted).
A = 2,
C = 7,
E = 0,
P = 1,
R = 8,
S = 3,
U = 9 ;
false.

?-
```