Knowledge representation (INFO0049-1)

Exercise session 4

10 Mar 2015

Try to draw search trees wherever possible to see how prolog executes a query

---

I. Basic list Exercise (contd.)

1. Define a predicate count_ occurrence _all(+Ls, -Zs) that succeeds if the list Zs is the list of the occurrence of Ls's elements.

   ?- count_ occurrence _all ([a, b, a, a, b, c, a], X).

   X = [[a|4], [b|2], [c|1]] ;
   false.

2. Suppose that we have a set of denominations (coins of 1 euro, 2, banknotes of 5, 10, 20, 50, 100, 200, 500) and we want to know the number of possible ways to pay a certain amount. Define a predicate to compute this number.

II. Introducing cuts

Cut can help in improving the efficiency of the program by avoiding unnecessary backtracking. The basic idea is to explicitly tell Prolog not to try the alternatives that are bound to fail.

3. Write a predicate max(+X, +Y, -Max) that succeeds if Max is the maximum of X and Y with and without the use of cut. Compare the efficiency of the two programs by drawing search trees.

III. Truth table

4. Define and/2, or/2, nand/2, nor/2, xor/2, impl/2 and equ/2 (for logical equivalence) as being operators, which succeed or fail according to the result of their respective operations; e.g. A and B will succeed, if and only if both A and B succeed. Note that A and B can be Prolog goals (not only the constants true and fail).

Now, write a predicate truth_table(+A, +B, +Expr) which prints the truth table of a given logical expression in two variables.

```
?- truth_table(A, B, A and (A or not B)).

true true true
true fail true
fail true fail
fail fail fail
```

IV. Binary trees

A binary tree is either empty or it is composed of a root element and two successors, which are binary trees themselves.

In Prolog we represent the empty tree by the atom 'nil' and the non-empty tree by the term t(Left,X,Right), where X denotes the root node and Left and Right denote the left and right binary sub trees, respectively.

5. Write a predicate is_binarytree(+Tree) which succeeds if and only if its argument is a Prolog term representing a binary tree.

6. Write a predicate count(+Tree, -Count) that calculates the number of elements in a given tree.

7. Write a predicate depth(+Tree, -depth) that calculates the depth of a given tree (the depth or `height' is the length of the path from the root to the deepest node in the tree).

V. Binary dictionaries

A binary dictionary is a binary tree t(Left, X, Right) if

- All the nodes in the left subtree ‘Left’ are less than X
- All the nodes in the Right subtree ‘Right’ are greater than X
- Both subtrees are also binary dictionaries by themselves

8. Write a predicate is_binarydictionary(+Tree) that succeeds if the given tree is a (correctly sorted) binary dictionary.

   ```
   ?- is_binarydictionary(t(t(t(nil, 1, nil), 2, nil),3, t(nil,5, t(nil,7, nil)))).
   True.
   ```

9. Write a predicate construct(+List, -Tree) to construct a binary search tree from a list of integer numbers.

   ?- construct([3,2,5,7,1],T).
   T = t(t(t(nil, 1, nil), 2, nil),3, t(nil,5, t(nil,7, nil)))

VI. Balanced binary dictionaries

A balanced tree has the following property:

"For each each node in the tree it is the case that the depth of the left subtree differs at most one from the depth of the right subtree."

Being balanced is an important property for a binary dictionary: it ensures access to an element in log(n) time, where n is the number of elements in the tree.

10. Write a predicate balanced(+Tree) that succeeds if the given binary dictionary is balanced.
