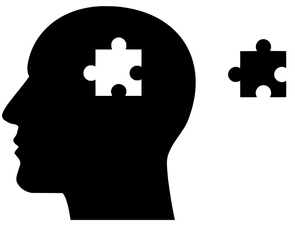
CSCI3180 Assignment 3



Introduction

The final assignment for this course consists of two parts:

- Basic Usage of Prolog.
- Knowledge Trees.

You will submit one source code file for each part at the end. No written report is needed this time.

Read the pinned Assignment 3 FAQ posts on Piazza for some common clarifications. You should check that a day before the deadline to make sure you don't miss any clarifications, even if you have already submitted your work then.

If you need further clarification of the requirements, please feel free to post on the Piazza with the assignment3 tag. However, to avoid cluttering the forum with repeated/trivial questions, please do read all the given code, webpage description, sample output, and latest FAQ carefully before posting your questions. Also, please be reminded that we won't debug for any student's assignment for the sake of fairness and that you should not share/post your solution code anywhere.

About academic integrity

We value academic integrity very highly.

- Do NOT try your "luck" - we use sophisticated plagiarism detection software to find cheaters.
- The penalty (for BOTH the copier and the copiee) may not be just getting a zero in your assignment. There may be additional disciplinary actions from the university, upto and including expulsion.
- Do NOT copy or look at the assignment code/work of another student/person.
- Do NOT share your assignment code/work to anyone and do NOT post it anywhere online.
- While we encourage discussion among students, you should write the code and finish the work on your own. In the discussion, you should NOT go into the details such that everyone will end up with the same code/work.
- You are responsible to double-check your own submission and do your best to avoid "accidents".

Read https://www.cuhk.edu.hk/policy/academichonesty/ for details of the CUHK policy on this matter.

Part one: Basic Usage of Prolog

Background

- Each problem will cover basic usages of Prolog, such as recursion, list manipulation, and pattern matching.
- Read the given example usages to see how we can make use of the facts and rules you have implemented to ask Prolog questions to obtain the correct results.

Problems

- (a) Check for a Palindrome
  - Define `is_palindrome(L)` which is true if list L is the same forwards and backwards.
  - Example usage:

```
?- is_palindrome([r, a, c, e, c, a, r]).
yes

?- is_palindrome([a, b, c, d]).
no
```

- (b) Interleave Two Lists
  - Define `interleave(L1, L2, L3)` which is `true` if `L3` is the result of interleaving the elements of `L1` and `L2`. Note that `L1` and `L2` have the same length here. For items at the same position, the item in `L1` goes before the item in `L2`.
  - Example usage:

```
?- interleave([a, b, c], [1, 2, 3], L3).
L3 = [a, 1, b, 2, c, 3]
```

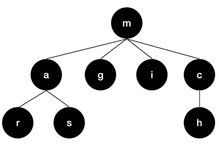
- (c) Duplicate List Elements
  - Define `duplicate_elements(L1, L2)` which is `true` if `L2` is the result of duplicating each element of `L1`.
  - Example usage:

```
?- duplicate_elements([a, b, c], L2).
L2 = [a, a, b, b, c, c]
```

Part two: Knowledge Trees

Background

- In the Enchanted Archives, there exist `Knowledge Trees`, which are multi-way trees that symbolize the branching nature of arcane wisdom. Each Knowledge Tree comprises a root node containing a unique magical rune (a single character) and a sequence of Knowledge Sub-Trees. The sub-trees are ordered to reflect the hierarchy of arcane secrets and are Knowledge Trees themselves.
- In Prolog, these Knowledge Trees are expressed as `kt(Char, Forest)`, where `Char` is the character at the root, and `Forest` is the list of Knowledge Sub-Trees. For this task, use the Prolog successor notation `s()` to represent natural numbers, e.g., `s(s(0))` means two.
- Read the given example usages to see how we can make use of the facts and rules you have implemented to ask Prolog questions to obtain the correct results.



Problems

- (a) Represent the Multi-Way Tree
  - Based on the figure example provided, represent the Knowledge Tree as `knowledge_tree(Tree)` in the form of `kt(Char, Forest)` with order of the sub-trees from left to right. `Char` is the root node at each level and `Forest` is a list, which contains all sub-trees of `Char`. For the leaf node, its `Forest` should be an empty list `[]`.
  - Example usage: Assume that we have a simple Knowledge Tree with the root node `x` and two children nodes `y` and `z`, we should represent it as follows,

```
?- knowledge_tree(kt(x, [kt(y, []), kt(z, [])])).
```

- (b) Depth of Node
  - Create a predicate `node_depth(Tree, Node, Depth)` where Depth represents the depth in the tree at which the node `Node` is located. The root has a depth represented by `s(0)`.
  - Example usage:

```
?- knowledge_tree(Tree), node_depth(Tree, m, Depth).
Depth = s(0).

?- knowledge_tree(Tree), node_depth(Tree, r, Depth).
Depth = s(s(s(0))).

?- knowledge_tree(Tree), node_depth(Tree, h, Depth).
Depth = s(s(s(0))).

?- knowledge_tree(Tree), node_depth(Tree, x, Depth).
false.
```

- (c) Number of Nodes
  - Define the predicate `num_node(Tree, N)` which is `true` if `N` is the number of nodes of the given `Tree`.
  - Note that it is possible that the same answer would be provided more than once for the query by your solution. It is acceptable for this part. We will grade the first answer generated by your solution. Also, you can check the `!/0` cut operator, which is a built-in operator to cut the duplicate solutions.
  - Example usage:

```
?- knowledge_tree(Tree), num_node(Tree, N).
N = s(s(s(s(s(s(s(s(s(0)))))))))).
```

- (d) Number of Leaf Nodes
  - Define a predicate `num_leaf(Tree, N)` that succeeds if `N` is the total number of leaf nodes in the given `Tree`.
  - Note that it is possible that the same answer would be provided more than once for the query by your solution. It is acceptable for this part. We will grade the first answer generated by your solution. Also, you can check the `!/0` cut operator, which is a built-in operator to cut the duplicate solutions.
  - Example usage:

```
?- knowledge_tree(Tree), num_leaf(Tree, N).
N = s(s(s(s(s(0)))))
```

- (e) Path to Node
  - Define a predicate `path_to_node(Tree, Node, Path)` that determines the path `Path` from the root to the specified node `Node`. The path should be represented as a list of nodes encountered along the way, including the starting and ending nodes.
  - Example usage:

```
?- knowledge_tree(Tree), path_to_node(Tree, m, Path).
Path = [m].

?- knowledge_tree(Tree), path_to_node(Tree, r, Path).
Path = [m, a, r].

?- knowledge_tree(Tree), path_to_node(Tree, h, Path).
Path = [m, c, h].

?- knowledge_tree(Tree), path_to_node(Tree, x, Path).
false.
```

Submission

Submission deadline: 23:59:00, Apr 2 (Tuesday)

Please submit your work as **a3.pl** to Blackboard "Assignment 3". Do NOT zip/compress/archive it. Make sure you have added and completed the following declaration at the top of the file:

```
/*
* CSCI3180 Principles of Programming Languages
*
* --- Declaration ---
* I declare that the assignment here submitted is original except for source
* materials explicitly acknowledged. I also acknowledge that I am aware of
* University policy and regulations on honesty in academic work, and of the
* disciplinary guidelines and procedures applicable to breaches of such policy
* and regulations, as contained in the website
* http://www.cuhk.edu.hk/policy/academichonesty/
*
* Name:
* Student ID:
* Email Address:
*
* Source material acknowledgements (if any):
*
* Students whom I have discussed with (if any):
*
*/
```

The usual assignment submission policy applies. The official testing environment is SWI-Prolog (version 9.0.4). Refer to the tutorials for how to use SSH to access the sparc1 Linux server which has that installed.
