CSCI3180 – Principles of Programming Languages – Spring 2019

Assignment 4 — Declarative Programming

Deadline: Apr 24, 2019 (Wednesday) 23:59

1 Introduction

Declarative programming is a programming paradigm that expresses the logic of a computation without describing its control flow. You will gain experience of declarative programming with an emphasis on Prolog and ML in this assignment.

2 Logic Programming

Implement the required predicates or queries of the following two problems in a Prolog program file named "asg4.pl". You should clearly indicate using comments the corresponding question number of each sub-problem. Note that the answers which are queries should be written in comments as well.

1. Recall the successor notation for representing natural numbers, and the `sum(X,Y,Z)` relation defined in the lecture which is true if Z is the sum of X and Y.
   (a) Define `natNum(X)` which is true if `X` is a natural number.
   (b) Define `lt(X,Y)` which is true if `X` is less than `Y`.
   (c) Give a query to find natural numbers less than 3.
   (d) Define `geq(X,Y)` which is true if `X` is equal to or greater than `Y`.
   (e) Define `max(X,Y,Z)` which is true if `Z` is the maximum of `X` and `Y`.
   (f) Based on `sum/3`, define `difference(X,Y,Z)` which is true if `X` minus `Y` is equal to `Z`.
   (g) Based on `lt/2`, `geq/2` and `difference/3`, define `mod(X,Y,Z)` which is true if `X` mod `Y` is equal to `Z`.
2. A *binary tree* is either empty or composed of a root and two children, which are binary trees themselves. The root of a binary tree is a node containing a value. Diagramatically, a binary tree consists of a set of *nodes* and lines connecting parents and children. The nodes are depicted by circles with values written inside. Empty binary trees are not shown in the diagram.

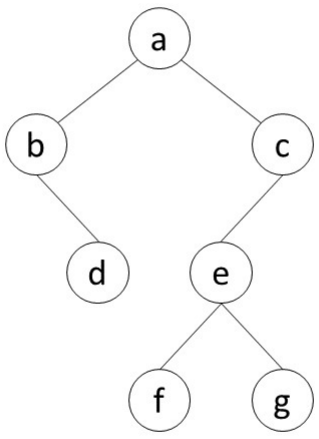

Figure 1: An example of binary tree

In Prolog, we represent an empty tree by the atom "`nil`" and a non-empty tree by the term "`bt(X,L,R)`", where `X` denotes the root node, `L` and `R` denote the left and right subtrees respectively.

(a) Represent the binary tree shown in Figure 1 as a Prolog term.

(b) Define the predicate `isTree(Term)`, where "`Term`" can be any Prolog term. Predicate `isTree(Term)` is true if "`Term`" represents a binary tree.

(c) Define the predicate `isLeaf(X,Tree)`, where "`X`" is a tree node, "`Tree`" is a binary tree. A leaf is a node with no children. Predicate `isLeaf(X,Tree)` is true if "`X`" is a leaf node of the given "`Tree`".

(d) Define the predicate `numberOfLeaf(Tree,N)`, where "`Tree`" is a binary tree, "`N`" is the number of leaf nodes in the **successor notation**. Note that you could use the `sum/3` defined previously.

(e) The *height* of a binary tree is the length of the path from the root to the deepest leaf node (with both children as empty) in the tree. Here we define the height of an empty tree to be zero. Thus a tree with a single element node (with both children as empty) will have a height of one (s(0)), and so on. Define the predicate `height(Tree,H)`, where "`Tree`" is a term representing a binary tree and H is the height of this tree in the **successor notation**. You may want to use the `max/3` defined above.

3 Functional Programming

Implement the required functions of the following problem in an ML program file named "asg4.ml". You should clearly indicate using comments the corresponding question number of each sub-problem.

This problem involves a card game invented just for this question. You will write several helper functions first and then implement a main program to track the progress of this game. The card game is played with a *card-list*, and the player will have a list of *held-cards*, initially empty. During each round of play, the player makes a move by choosing to draw a card from the *card-list*, or discarding one of the *held-cards*. The game will end either when the player chooses to stop or when the player already has six cards.

The goal of this game is to end the game with a low score (0 is the best). The score is defined as the absolute value of the differences between sum of the red color card values and the sum of black color card values, and then plus six minuses the number of held cards. The details are shown in the following formula:

$$Score = abs(sum(red\_values) - sum(black\_values)) + (6 - \#held\_cards).$$

The type definition of a card is shown below (The joker cards are not included in this game):

```
datatype suit = Clubs | Diamonds | Hearts | Spades;
datatype rank = Jack | Queen | King | Ace | Num of int;
type card = suit * rank;
```

Each card will have associated suit and rank. The suit of a card is related to the color and rank is related to the value of the card. The color type and move type are defined as

```
datatype color = Red | Black;
datatype move = Discard of card | Draw;
```

1. Write an ML function `abs`, which takes an integer and returns its absolute value.

2. Write an ML function `cardColor`, which takes a card as input and returns its color. Spades and clubs are black, while diamonds and hearts are red. Note that one case-expression is enough.

3. Write an ML function `cardValue`, which takes a card as input and returns its value. Numbered cards have their numbers as the value. Aces are 11, and everything else is 10.

4. Write an ML function `numOfCard`, which takes a card list `cardList` and returns the number of cards in the list.
5. Write an ML function `removeCard(cardList, c)`, which takes a card list `cardList` representing the held-cards, and a card `c`. It should return a list after removing card `c` from the list. If c is in the list more than once, remove only the first one. You could assume that `c` is always contained in the list. Note that "`=`" can be used to compare cards.
6. Write an ML function `sumCards(cardList, col)`, which takes a list of cards `cardList` and a color `col`. It will return the sum of the given color card values.
7. Write an ML function `score`, which takes a card list `cardList` (the held-cards) and computes the score as defined earlier.
8. Write an ML function `runGame(cardList, moveList)`, which takes a card list `cardList` representing the cards from which the player can draw, a move list `moveList` (the moves chosen by the player at each round of play), and returns the score at the end of the game after processing the moves in the move list in order. As described above:
   - At the beginning of the game, the list representing held-cards should be empty.
   - If there are no more moves (i.e. the move list is empty), the game will end.
   - During each round of play, if the player choose to discard some card `c`, the game will continue with the held-cards having `c` removed and the card list remains unchanged.
   - If a player chooses to draw a card and the card list is already empty, the game is over. If drawing causes the number of the held-cards to exceed five, the game is over (after drawing). Otherwise, the game continues with a larger held-cards and a smaller card-list.

4 Submission Guidelines

Please read the guidelines CAREFULLY. If you fail to meet the deadline because of submission problem on your side, marks will still be deducted. So please start your work early!

1. In the following, **SUPPOSE**

   your name is *Chan Tai Man*,
   your student ID is *1155234567*,
   your username is *tmchan*, and
   your email address is *tmchan@cse.cuhk.edu.hk*.

2. In your source files, insert the following header. REMEMBER to insert the header according to the comment rule of Prolog and ML.

```
/*
 * CSCI3180 Principles of Programming Languages
 *
 * --- Declaration ---
 *
 * I declare that the assignment here submitted is original except for source
 * material 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/
 *
 * Assignment 4
 * Name : Chan Tai Man
 * Student ID : 1155234567
 * Email Addr : tmchan@cse.cuhk.edu.hk
 */
```

The sample file header is available at

```
http://course.cse.cuhk.edu.hk/~csci3180/resource/header.txt
```

3. Late submission policy: less 20% for 1 day late and less 50% for 2 days late. We shall not accept submissions more than 2 days after the deadline.

4. Your solutions for Section 2 will be graded using SICStus Prolog 3.12.7 in the CSE Unix platform. Solutions for Section 3 will be graded using Standard ML 110.0.7 in the CSE Unix platform.

5. `Tar` your source files to `username.tar` by

```
tar cvf tmchan.tar asg4.pl asg4.ml
```

6. `Gzip` the `tarred` file to `username.tar.gz` by

```
gzip tmchan.tar
```

7. `Uuencode` the `gzipped` file and send it to the course account with the email title "HW4 *studentID yourName*" by

```
uuencode tmchan.tar.gz tmchan.tar.gz \
| mailx -s "HW4 1155234567 Chan Tai Man" csci3180@cse.cuhk.edu.hk
```

8. Please submit your assignment using your Unix accounts.

9. An acknowledgement email will be sent to you if your assignment is received. **DO NOT** delete or modify the acknowledgement email. You should contact your TAs for help if you do not receive the acknowledgement email within 5 minutes after your submission. **DO NOT** re-submit just because you do not receive the acknowledgement email.

10. You can check your submission status at

    `http://course.cse.cuhk.edu.hk/~csci3180/submit/hw4.html`.

11. You can re-submit your assignment, but we will only grade the latest submission.

12. Enjoy your work :>