CSCI3180 – Principles of Programming Languages – Spring 2022

Assignment 4 — Declarative Programming

Deadline: May 1, 2021 (Sunday) 23:59

1 Introduction

Declarative programming is a programming paradigm that emphasises the expression of "what" the problem is over "how" to solve the problem. You will gain experience of declarative programming with Prolog (Logic Programming) and ML (Functional Programming) in this assignment.

You are supposed to test your work on the machine `solar1.cse.cuhk.edu.hk` using

- SICStus 3.12.7
- Standard ML of New Jersey, Version 110.0.7

They can be invoked using the commands `sics` and `sml` respectively.

2 Logic Programming

Implement the predicates of the following problem in a Prolog program file named "asg4.pl". You should clearly indicate using *comments* the corresponding problem number. Note that you are **not allowed** to use any built-in predicates in Prolog. Recall the example of the binary tree in the lecture. You will be practising classic binary tree questions using the successor notation. In the following, we assume that all nodes in a binary tree are natural numbers with the successor notation, as shown in Figure 1. Remember that all numbers your predicates use should also be in the successor notation.

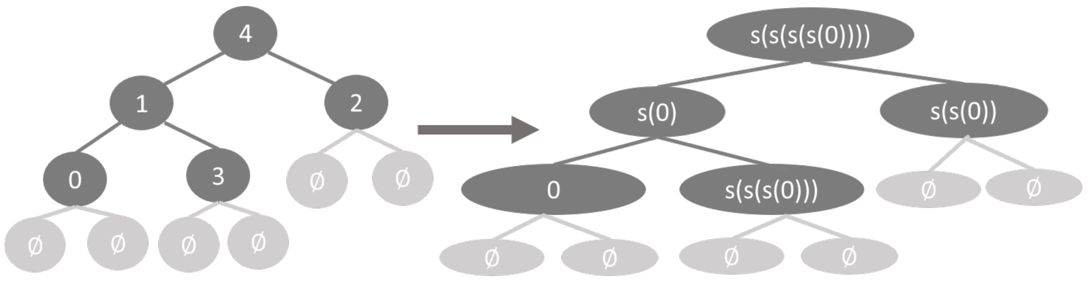

Figure 1: A Binary Tree Example.

1. Define predicate `is_binary_tree(T)` which is true if `T` is a binary tree. Example:
```
?- is_binary_tree(bt(bt(bt(nil,0,nil),s(0),bt(nil,s(s(s(0))),nil)),
s(s(s(s(0)))), bt(nil,s(s(0)),nil))).
yes
```

2. A binary search tree is a rooted binary tree data structure each internal node of which stores a key greater than all the keys in the node's left subtree and less than those in the node's right subtree.
a) Define `lt(X,Y)` which is true if `X` is less than `Y` in the successor notation.
b) Using `lt(X,Y)`, define `bs_tree(T)` which is true if `T` is a binary search tree. Example:
```
?- bs_tree(bt(bt(bt(nil,0,nil),s(0),bt(nil,s(s(s(0))),nil)),s(s(s(s(0)))),
bt(nil,s(s(0)),nil))).
no
?- bs_tree(bt(bt(bt(nil,0,nil),s(0), bt(nil,s(s(s(0))),nil)), s(s(s(s(0)))),
bt(nil,s(s(s(s(s(0))))),nil))).
yes
```

3. Define `parent(T, P, C)` is true if node `C` is a child of node `P` in a binary tree `T`. Example:
```
?- parent(bt(bt(bt(nil,0,nil),s(0), bt(nil,s(s(s(0))),nil)), s(s(s(s(0)))),
bt(nil,s(s(0)), nil)), s(s(s(s(0)))), C).
C = s(0) ?  ;
C = s(s(0)) ?  ;
no
?-parent(bt(bt(bt(nil,0,nil),s(0), bt(nil,s(s(s(0))),nil)), s(s(s(s(0)))),
bt(nil,s(s(0)), nil)), P, 0).
P = s(0) ?  ;
no
```

4. Define `descendent(T, A, D)` is true if node `D` is a descendent of node `A` in a binary tree `T`. Example:
```
?- descendent(bt(bt(bt(nil,0,nil),s(0), bt(nil,s(s(s(0))),nil)), s(s(s(s(0)))),
bt(nil,s(s(0)), nil)), s(s(s(s(0)))), D).
D = s(0) ?  ;
D = s(s(0)) ?  ;
D = 0 ?  ;
D = s(s(s(0))) ?  ;
no
?- descendent(bt(bt(bt(nil,0,nil),s(0), bt(nil,s(s(s(0))),nil)), s(s(s(s(0)))),
bt(nil,s(s(0)), nil)), P, 0).
P = s(0) ?  ;
P = s(s(s(s(0)))) ?  ;
no
```

5. Define `count_nodes(T, X)` which is true if `X` is the total number of all non-empty nodes in a binary tree `T`. Example:
```
?- count_nodes(bt(bt(bt(nil,0,nil),s(0), bt(nil,s(s(s(0))),nil)), s(s(s(s(0)))),
bt(nil,s(s(0)), nil)), X).
X = s(s(s(s(s(0))))) ?  ;
no
```

6. Define `sum_nodes(T, X)` which is true if `X` is the sum of all nodes in a binary tree `T`. Example:
```
?- sum_nodes(bt(bt(bt(nil,0,nil),s(0), bt(nil,s(s(s(0))),nil)), s(s(s(s(0)))),
bt(nil,s(s(0)), nil)), X).
X = s(s(s(s(s(s(s(s(s(s(0)))))))))) ?  ;
no
?- sum_nodes(T, s(s(s(s(s(s(0))))))).
T = bt(nil,s(s(s(s(s(s(0)))))),nil) ?  ;
T = bt(nil,s(s(s(s(s(s(0)))))),bt(nil,0,nil)) ?  ;
T = bt(nil,s(s(s(s(s(0))))),bt(nil,s(0),nil)) ?  ;
T = bt(nil,s(s(s(s(0)))),bt(nil,s(s(0)),nil)) ?  ;
T = bt(nil,s(s(s(0))),bt(nil,s(s(s(0))),nil)) ?  ;
T = bt(nil,s(s(0)),bt(nil,s(s(s(s(0)))),nil)) ?
yes
```

7. Using the `append(L1,L2,L3)` predicate, define `preorder(T, X)` which is true if `X` holds the results from a preorder traverse of a binary tree `T`. Example:
```
?- preorder(bt(bt(bt(nil,0,nil),s(0), bt(nil,s(s(s(0))),nil)), s(s(s(s(0)))),
bt(nil,s(s(0)),nil)), X).
X = [s(s(s(s(0)))), s(0), 0, s(s(s(0))), s(s(0))] ?  ;
no
```

3 Functional Programming

Implement the required functions of the following problems in a `Standard ML` program file named "asg4.sml". You should clearly indicate using comments the corresponding question number of each problem.

We design a card game, based on the popular Black Jack game, for the practice of functional programming. A set of functions should be implemented to help you finish the main program step by step. The card game is designed for two players: **Player 1** and **Player 2**. The two players draw cards from a common card list in turn, following the order of the cards in the card list. The number of drawn cards is determined by each player. **Player 1** draws cards first. In this card game, only the ranks of cards are considered. The joker cards are not included. Each rank has its corresponding point.

To increase the interest of this game, we design two versions: the basic version and the magic version. In the basic version, the point of each rank is unique: 1) the point of `Jack`, `Queen`, and `King` is 10; 2) the point of the cards with ranks 2 to 9 is the same as the corresponding rank; and 3) the point of `Ace` is 1. In the magic version, the point of `Ace` can be either 1 or 11, as determined by the players to maximize their chance of winning. The points of other ranks are the same as those of the basic version.

In this game, the total point (TP) of the drawn cards of each player is computed as below:

$$TP = \begin{cases} 21 & if\ sp \mod 21 = 0\ and\ sp > 0 \\ sp \mod 21 & otherwise \end{cases} \tag{1}$$

where *sp* is the sum of the points of all drawn cards. After the TP of each player is computed, the result of the game can be determined. If the TP of **Player 1** is larger than that of **Player 2**, **Player 1** wins the game. If the TP of **Player 1** is smaller than that of **Player 2**, **Player 2** wins the game. Otherwise, the game ends in a tie.

In the ML implementation, the type definition of ranks is shown below:

```
datatype rank = Jack | Queen | King | Ace | Num of int;
```

We will use some examples to illustrate the rules of the games, first for drawing cards. Assume the card list is `[Num 1, Num 2, Num 3, Num 4, King, Jack, Queen, Ace, Num 9, Num 10]`. **Player 1** and **Player 2** draw cards from the same card list in turn. **Player 1** always goes first.

- Assume **Player 1** draws 3 cards and **Player 2** draws 3 cards. **Player 1** will get `[Num 1, Num 3, King]`. **Player 2** will get `[Num 2, Num 4, Jack]`.
- If the numbers of drawn cards are different for the two players, one player stops drawing cards when the player gets enough cards while the other player continues to draw cards one by one from the remaining cards in the card list. Assume **Player 1** draws 5 cards and **Player 2** draws 3 cards. **Player 1** will get `[Num 1, Num 3, King, Queen, Ace]`. **Player 2** will get `[Num 2, Num 4, Jack]`.
- Assume **Player 1** draws 3 cards and **Player 2** draws 5 cards. **Player 1** will get `[Num 1, Num 3, King]`. **Player 2** will get `[Num 2, Num 4, Jack, Queen, Ace]`.

Then, we will use an example to explain the calculation of TP for the basic version and the magic version. Assume **Player 1** gets `[Ace, Ace, 3]`. In the basic version, the point of `Ace` is 1. Therefore, TP can be computed as $1 + 1 + 3 = 5$. In the magic version, the point of `Ace` can be 1 or 11. Players should choose the proper point of each `Ace` to make TP the largest. Therefore, TP can be calculated by a divide-and-conquer method. In this case, the best TP is $1 + 11 + 3 = 15$.

The functions you should implement in the submitted file are given below. Note that the introduced order of input values should also be the implemented order in your submitted file. In implementing the following functions, you should use **pattern matching** and **let_in** wherever possible for tidier codes.

1. Write an ML function `get_basic_point`, which takes a `rank` as input and returns the point (int value) of the rank for basic version.

   For example, the return value of `get_basic_point(Ace)` is 1.

   The return value of `get_basic_point(Num 5)` is 5.

   The return value of `get_basic_point(Jack)` is 10.

2. Write an ML function `get_cards`. The function takes as input the number of cards for **Player 1** (int value, `n1`), the number of cards for **Player 2** (int value, `n2`), and the common card list for the two players (rank list, `card_list`). The function returns a pair of card lists to the two players: (player1_cards, player2_cards) (rank list * rank list). You can assume that `n1` and `n2` are positive, and `n1 + n2` is smaller than the number of the cards in `card_list`.

   For example, the return value of `get_cards(2, 2, [Num 5, Num 2, Num 3, Num 4])` is `([Num 5, Num 3], [Num 2, Num 4])`.

3. Write an ML function `cal_TP`. This function is a tool for the game, and takes the sum of points for all drawn cards, which is *sp* defined above (int value), as input. The function returns TP (int value).

   For example, the return value of `cal_TP(22)` is 1. The return value of `cal_TP(21)` is 21. The return value of `cal_TP(0)` is 0.

4. Write an ML function `get_basic_TP`. The function takes a `rank list` as input and returns the TP (int value) value of the input for the basic version. Therefore, the calculation rules follow the basic version, where `Ace` is equal to 1.

   For example, the return value of `get_basic_TP([Ace, Num 2, Num 3, Num 4])` is 10.

5. Write an ML function `get_TP`. The function takes a `rank list` as input and returns the TP (int value) value of the input for the magic version. Therefore, the calculation rules follow the magic version, where `Ace` can be 1 or 11.

   For example, the return value of `get_TP([Ace, Num 10])` is 21.

6. Write an ML function `judge_winner_basic` for the basic version. The functions takes as input the drawn cards of **Player 1** (rank list) and the drawn cards of **Player 2** (rank list). The function returns the result (int value) of the game: a) if **Player 1** wins, the function returns 1; b) if **Player 2** wins, the function returns 2; and c) the function returns 0 for a tie.

7. Write an ML function `judge_winner` for the magic version. The functions takes as input the drawn cards of **Player 1** (rank list) and the drawn cards of **Player 2** (rank list). The function returns the result (int value) of the game: a) if **Player 1** wins, the function returns 1; b) if **Player 2** wins, the function returns 2; and c) the function returns 0 for a tie.

8. Write an ML function `basic_result` for the basic version. The function takes as input the number of cards for **Player 1** (int value, `n1`), the number of cards for **Player 2** (int value, `n2`), and the common card list for the two players (rank list, `card_list`). The function returns the result (int value) of the game: a) if **Player 1** wins, the function returns 1; b) if **Player 2** wins, the function returns 2; and c) the function returns 0 for a tie. You can assume that `n1` and `n2` are positive, and `n1 + n2` is smaller than the number of the cards in `card_list`.

9. Write an ML function `result` for the magic version. The function takes as input the number of cards for **Player 1** (int value, `n1`), the number of cards for **Player 2** (int value, `n2`), and the common card list for the two players (rank list, `card_list`). The function returns the result (int value) of the game: a) if **Player 1** wins, the function returns 1; b) if **Player 2** wins, the function returns 2; and c) the function returns 0 for a tie. You can assume that `n1` and `n2` are positive, and `n1 + n2` is smaller than the number of the cards in `card_list`.

10. Write an ML function `cal_largest_TP` for the magic version. This function is designed for calculating the largest TP which can be gotten by **Player 2** with any legal number of cards. The function takes as input the number of cards for **Player 1** (int value, `n1`) and the common card list for the two players (rank list, `card_list`). The function returns the largest TP for **Player 2** out of all the possible numbers of drawn cards from the card list. You can assume that `n1` is positive, and the total number of drawn cards for the two players must not be larger than the number of the cards in `card_list`.

    For example, the return value of `cal_largest_TP(1, [Ace, Num 10, Ace, Num 3, Ace])` is 21 when Player 2 gets `[Num 10, Ace]`.

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` or `Standard 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
 */
```

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. `Tar` your source files to `tmchan.tar` by

```
tar cvf tmchan.tar asg4.pl asg4.sml
```

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

```
gzip tmchan.tar
```

6. `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
```

7. Please submit your assignment using your linux accounts.

8. An acknowledgement email will be sent to you if your assignment is received. Please **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.

9. You can check your submission status at

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

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

11. Enjoy your work :>