CSCI3180 – Principles of Programming Languages – Spring 2018

Assignment 4 — Declarative Programming

Deadline: Apr 27, 2018 (Friday) 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 Getting familiar with Prolog and ML

Implement the required predicates or queries of this section in a Prolog program file named "task.pl" and an ML program file named "task.ml". Note that the answers which are queries should be written in comments as well.

1. In assignment 3, we have designed and implemented a perl program to assign tasks to servers. In this assignment, we will use Prolog and ML to re-implement some of the basic functions.

   Assume that we only have one server and you are given a list of tasks in the form of:

   - **Prolog:** `task(t1, 2, 4). task(t2, 4, 6). ...`
   - **ML:** `val tasks = [("t1", 2, 4),("t2", 4, 6),...] : (string * int * int) list`.

   where `t1` (`"t1"` for ML), `2` and `4` is the name, start time and end time of the first task respectively, i.e.`(name, start, end)`. You can assume that both start time and end time can be integers only.

   You are required to implement the following functions:

   (a) **Check Task**
   - **Prolog:** `check_task(T)`, e.g. `check_task(t1)`, where `T` = `t1`
   - **ML:** `check_task(T:string, tasklist:(string * int * int) list):bool`, i.e. `val check_task = fn : string * (string * int * int) list -> bool`
   - `check_task()` checks whether a valid task exists, i.e.
     - A task with name `T` exist in the tasks given
     - `AND` the end time is greater or equal to start time.

   (b) **Compatible**
   - **Prolog:** `compatible(T1, T2)`, e.g. `compatible(t1, t2)` where `T1` = `t1` and `T2` = `t2`
   - **ML:** `compatible(T1:string, T2:string, tasklist:(string * int * int) list):bool`, i.e. `val compatible = fn : string * string * (string * int * int) list -> bool`
   - `compatible()` checks whether two task are compatible, i.e.
     - The two tasks `T1` and `T2` have to be both valid
     - `AND` both tasks don't overlap. (End time of one task has to be less than or equal to the other one)

   (c) **Compatible List**
   - **Prolog:** `compatible_list(L)`, e.g. `compatible([t1, t2, t3])` where `L` = `[t1, t2, t3]`
   - **ML:** `compatible_list(L: string list, tasklist: (string * int * int) list):bool` i.e. `val compatible_list = fn : string list * (string * int * int) list -> bool`

- `compatible_list()` checks if all tasks in the list are pairwise compatible, i.e.
  - Check if all possible pairs of tasks are `compatible()`
  - For example if the list contains [t1, t2, t3], then check compatibility for pairs (t1, t2), (t1, t3) and (t2, t3).

You are encouraged to use pattern matching techniques and let expressions to implement your functions. You may define extra functions not listed above if needed.

3 Logic Programming

Implement the required predicates or queries of the following problem in a Prolog program file named "logic.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 `uint_num(X)` which is true if `X` is a natural number.
   (b) Define `gt(X,Y)` which is true if `X` is greater than `Y`.
   (c) Give a query to list all natural numbers less than 3.
   (d) Based on `sum/3`, define `product(X,Y,Z)` which is true if `Z` is the product of `X` and `Y`.
   (e) Give a query to compute the product of 2 and 3.
   (f) Give a query to compute the result of 8 divided by 4.
2. Recall the list operations taught in lectures and tutorials.
   (a) Define `nth(X,L,N)` which is true if the `N` th element in list `L` is `X`.
   (b) Based on `nth/3` or otherwise, define `third(X,L)` which is true if the third element in list `L` is `X`.

4 Lambda Calculus and Functional Programming

Please answer the following lambda calculus and functional programming questions. You should clearly indicate using comments the corresponding question number of each sub-problem.Implement the required predicates or queries of this section in an ML program file named "lambda.ml".

1. Consider the following lambda expression:

$$(\lambda c.c\ (\lambda a.\lambda b.b))((\lambda a.\lambda b.\lambda f.f\ a\ b)\ p\ q)$$

   (a) Reduce the lambda expression using normal order reduction. State clearly for each step whether it is a β-, δ-, or other kinds of reduction.
   (b) Reduce the lambda expression using applicative order reduction. State clearly for each step whether it is a β-, δ-, or other kinds of reduction.
2. Consider the following lambda expression. The constants "mul" and "add" are multiplication and addition operators, respectively.

$$(\lambda x.\lambda y.\ (mul\ x\ ((\lambda x.\ (add\ x\ 3))\ y)))\ 7\ 8$$

   (a) Reduce the above lambda expression to its normal form using the normal order reduction strategy. Show clearly for each step whether it is an α-, β- or δ-reduction.
   (b) Reduce the above lambda expression to its normal form using the applicative order reduction strategy. Show clearly for each step whether it is an α-, β- or δ-reduction.

(c) Define a function f in ML, such that

$$f = (\lambda x.\lambda y.\ (mul\ x\ ((\lambda x.\ (add\ x\ 3))\ y)))$$

3. John is leading a project to design a new programming language *Lapi* with C-like syntax and semantics, in which two kinds of abstractions are supported. First, function abstractions are represented by expressions. Second, procedure abstractions are represented by λ expressions. All λ and π abstractions are first class values in *Lapi*. After some time the first version of *Lapi* is designed. John writes the following *Lapi* program.

```
#include<lambdah>
#include<pih>

main();
{
lambda f1, f2;
pi loop, swap;
int m = 1, n = 1, k;

f1 = lambda x . (add x 1);
f2 = (lambda g . g 5) (lambda x . (x + 3));

loop = pi (n, c) {for (i=1; i <= n; i++) c};
swap = pi (x, y) {t = x; x = y; y = t};

k = eval_normal(f1 m); printf(k);
m = eval_applicative(f2); printf(m);
swap(k, m); printf(k);
loop(3, pi (u, v) w = u + 1; u = v + 1; v = w (k, m)); printf(k, m);

halt;
}
```

(a) What do you expect to be printed by the four `printf` commands? You must justify all your answers.

(b) Peter is not in the *Lapi* project team. He reads the above program and comments that the command

```
k = eval_normal(f1 m);
```

can be written as

```
k = (f1 m);
```

and this will not affect the answers printed by all the four `printf` commands. Do you agree with Peter? Give reason(s) to support your answer.

(c) What are the scopes and lifetimes of the variables `t` and `x` in the above *Lapi* program?

(d) The *Lapi* project team is going to hold a meeting to discuss whether static scoping or dynamic scoping should be used in *Lapi*. You are invited to be present in the meeting and give your opinions. What are your opinions? Justify your answer, using examples for illustration if necessary.

5 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 task.pl task.ml logic.pl lambda.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 :>