German University in Cairo
Media Engineering and Technology
Prof. Dr. Slim Abdennadher

Concepts of Programming Languages, Spring 2014
Haskell Project

Submission: 15.05.2014 (11:59 pm)

The project will put your knowledge in Haskell to the test. Before proceeding, make sure that you read each section carefully. Enjoy.

Instructions

Please read and follow the instructions carefully:

a) The project should be implemented using Haskell (Hugs98) based on the syntax discussed in class.

b) This is a team project. You can work in groups of **maximum** 3 members. The groups for this project are the same as project 1. In case of any changes, please report it by sending an email to maha.samir@guc.edu.eg. You must send complete information about your new team including full names and unique GUC application numbers. The deadline for sending group change requests is Wednesday 30.04.2014 (11:59 pm).

c) You have to write a **formal report** on your project covering the following points:

- A brief formal description of the datatypes that you have created.
- A brief formal description of the *main* functions used in your implementation.
- A listing of the *helper* functions that you called from your main functions. You are allowed to use built-in Haskell functions, like `length` or other functions in `Hugs.Prelude` without explanation.

d) Every team has to submit a soft copy of the following files (in a zipped directory) to the following email address csen403s14@gmail.com:

- your Haskell project **source file** named after your team code, e.g. `T10G03.hs`
- the report file (.doc or .pdf) also named after your team code , e.g. `T10G03.pdf`
- The test case as described in the following sections.
- the zipped file should also be named after your team code, e.g. `T10G03.zip`
- your email subject has to be formatted as follows: Project␣2␣TeamCode, e.g. Project 2 T10G03
- it is **your** responsibility to ensure that you have submitted the correct files and saved the correct content before attaching.

e) Every team has to submit a **proper** *hard copy* of the following:

- the report
- a printout of the source file
- a printout of the submitted test case

f) You are always welcome to discuss the project with the TAs. You must work with your team members only. Do not exchange information with other teams or individuals.

g) Cheating and plagiarism will be strictly punished with a grade of 0 in the project.

h) Once you submit the project, you will be appointed a date to show up with your team members for an **oral evaluation** of your project. The evaluation will cover practical and technical details as well as theoretical concepts concerning Haskell and the general features of the functional programming paradigm.

i) Please respect the **submission deadline** marked at the beginning of the document as well as the date of the oral evaluation set by your TA. Any delay will result in a rejection of the submission and a cancellation of the oral evaluation.

Project Description

You are required to implement an interpreter for a mini-Prolog language. The interpreter should be able to answer queries written in a special notation, which you will define.

In order to answer a query, the rules of the knowledge base should be explored one after the other, in the same order they are given to your interpreter. Once a query unifies with a fact, it succeeds. When a query unifies with the head of a rule, the goals specified in the body of that rule have to be proven in order for the query to succeed.

The following restrictions will be made about our mini-Prolog language for simplicity:

- A term will be either a variable or a constant. Our language will not include function symbols with arity bigger than zero.
- Every fact or rule in the given knowledge base will have in its head an argument list of constants and/or **distinct** variable names. The names of the constants could be anything. However, the variable names in the knowledge base will be chosen from the set { M, N, ... , X, Y, Z }.
- No variable will appear in the body of a rule unless it appears in its head.
- No body of any rule will contain negation or recursion.
- The list of arguments in a given query will consist of constants and/or **distinct** variable names.
- The variable names used in queries will be chosen from the set { A, B, ... , J, K, L }. Hence, all variables appearing in queries can be considered fresh variables for the rules in the knowledge base.

In order to implement the interpreter, you are required to do the following:

a) Define data types for the following:

   - **Term**: A term is either a constant or a variable. The name of the term is a String.
   - **Predicate**: A predicate is represented by its name and list of arguments. The arguments are terms.
   - **Goals**: A (possibly empty) collection of goal predicates. For simplicity, you only have to handle goals which are logically ANDed, i.e., must all hold.
   - **Rule**: A rule consists of a head and a body. The head is a simple **Predicate** while the body is of type **Goals**. Note that a fact can be represented as a rule with an empty body.
   - **Solution**: A solution is either No (failure) or Yes (success). In case of success, the solution should also include the list of variable substitutions required.

b) Implement the function **unifyWithHead** which evaluates the solution for unifying two predicates. The input to this function will be:

   - A predicate representing the query or the goal.
   - A predicate representing the head of a rule in the knowledge base.
   - The current solution which indicates which variables are bound and their substituted values.

The output of this function will be of type **Solution**. Hence, the new solution will be either:

- No if the unification is not possible.
- Yes and the (updated) list of variable substitutions.

c) Implement the function **applySolSubInBody**. The input to this function will be a success **Solution** and a body of type **Goals**. The function should evaluate the new body in which the variable substitions of the solution are applied.

d) Implement the function **allSolutions**. The input to this function will be a predicate representing a query and a list of rules representing the knowledge base. The function should evaluate the query according to the given knowledge base. The function should output a list of type **Solution** representing all possible solutions to the given query. If the query fails (has no solutions at all), the output will be an empty list.

Sample Test

The following knowledge base and sample queries should be translated to your notation as a test case, according to your implementation of the datatypes defined in part a. You have to submit a text file representing the test case in your notation.

Important Note: the test case is not a guarantee that your code is 100% correct. It is untimately **your** responsibility to check that your code is logically correct.

```
male(timmy).
male(alex).
male(slim).
male(azmy).
male(remon).

female(amira).
female(reem).
female(wanda).

parent(slim,amira).
parent(wanda,timmy).
parent(azmy,reem).
parent(azmy,remon).

father(X,Y):- male(X),
              parent(X,Y).

daughterFather(X,Y):- father(Y,X), female(X).

?- male(A).
?- female(A).
?- parent(A,B).
?- parent(A,reem).
?- parent(A,alex).
?- father(A,B).
?- daughterFather(D,F).
?- daughterFather(amira,F).
?- daughterFather(amira,alex).
?- daughterFather(D,slim).
?- daughterFather(remon,azmy).
?- daughterFather(reem,azmy).
```

Grading and Evaluation

The evaluation of the practical part of the project, i.e the implementation, is the same for the whole team. However, the oral evaluation is team-member-specific. Your oral evaluation will affect your overall project grade. So be prepared :)

Tips

Here is a list of the do's and don't for a better performance in the project as well as in the course:

- Read carefully the lecture notes and revise the slides and practice assignment problems. They include a lot of useful information for you. Moreover, check the links on the course page for further help: `http://met.guc.edu.eg/Courses/Links.aspx?crsEdId=483`
- Start working early on the project. Do not get carried away and keep track of time. It would be handy if you planed ahead your project and worked out a reasonable timeline.
- Divide the work among your team members. Coordinate your efforts together and meet regularly to update each other with the progress.

Finally, we wish you the best of luck!