Homework 6: Relational Programming (aka Logic Programming)

Due: Friday, December 5, 2008 at 11pm.

In this homework you will learn a bit about the relational model and basic techniques of logic programming. The programming techniques include specification of relationships and using generate and test [Concepts] [UseModels]. A few problems also make comparisons with the other models we have studied, and also with embeddings of the relational model in other languages [EvaluateModels] [MapToLanguages].

Your code should be written in the relational model of chapter 9 of our textbook [RH04]. Thus you should not use cells and assignment in your Oz solutions. Furthermore, note that the relational model does *not* include the primitive `IsDet` or the library function `IsFree`; thus you are also prohibited from using either of these functions in your solutions.

You should use helping functions whenever you find that useful. Unless we specifically say how you are to solve a problem, feel free to use any functions from the Oz library (base environment), especially functions like `Map` and `FoldR`.

For all Oz programing tasks, you must run your code using the Mozart/Oz system. For these you must also provide evidence that your program is correct (for example, test cases). Oz code with tests for various problems is available in a zip file, which you can download from the course resources web page. For testing, you may want to use tests based on our code in the course library file `TestingNoStop.oz`.

Turn in (on Webcourses) your code and, if necessary, output of your testing for all questions that require code. Please upload text files with your code that have the suffix `.oz` and text files with suffix `.txt` that contain the output of your testing. Please use the name of the main function as the name of the file. (In any case, don't put any spaces in your file names!)

If we provide tests and your code passes all of them, you can just indicate with a comment in the code that your code passes all the tests. Otherwise it is necessary for you to provide us test code and/or test output. If the code does not pass some tests, indicate in your code with a comment what tests did not pass, and try to say why, as this enhances communication and makes commenting on the code easier and more specific to your problem than just leaving the buggy code without comments.

If you're not sure how to use our testing code, ask us for help.

Your code should compile with Oz, if it doesn't you probably should keep working on it. If you don't have time, at least tell us that you didn't get it to compile.

Be sure to clearly label what problem each area of code solves with a comment.

Don't hesitate to contact the staff if you are stuck at some point.

Read Chapter 9 of the textbook [RH04]. (See the syllabus for optional readings.)

Message Passing Semantics and Expressiveness

1. [Concepts]

   (a) (5 points) [EvaluateModels]

   Could the relational model be used to implement a data abstraction, `Box` as in the previous homework, which acts like a cell? Briefly explain.

   (b) (5 points) [MapToLanguages]

   Why is the notion of encapsulated search, using `Solve`, appropriate for making logic programming features available in a language such as Java? Briefly explain.

   (c) (10 points) [Concepts]

   Briefly describe (in English) a specific programming problem that is not part of this homework and is not in the textbook, but for which logic programming would be the most appropriate model.

   Also briefly explain why logic programming would be good for your problem.

2. (suggested practice) [EvaluateModels]

   Update your table from homeworks 4 and 5 that lists all the different programming techniques, the characteristics of problems that are best solved with these techniques (i.e., when to use the techniques), and the name of at least one example of that technique. In this update, add entries for the new row on the "relational" model.

programming technique	problem characteristics	example(s)
recursion (over grammars)		
higher-order functions		
stream programming		
lazy functions		
message passing		
relational		

Programming

If we give tests where we use `SolveFirst` or `SolveOne`, your program could have more solutions than our solution for that query.

3. (10 points) [UseModels]

   Write, using the relational model, a relation `Consecutive` such that `{Consecutive Ls X Y Suffix}` succeeds if in the list `Ls`, the items `X` and `Y` appear next to each other, in that order, followed by the list `Suffix`. That is `Suffix` is the remainder of the list after the consecutive appearance of `X` and `Y`. (The relation will fail otherwise.)

   You must use **`choice`** in your solution.

   Hint: this is basically recursive.

   Figure 1 on the following page shows several examples, including backwards and forwards tests. These use `SolveAll` and `SolveFirst`, which are provided in the testing zip file.

4. (30 points) [UseModels]

   Write, using the relational model, a relation `GenerateTriple` such that `{GenerateTriple N Result}` succeeds if `N` is a positive integer, and `Result` is a triple of the form $A\#B\#C$ where $A \leq B \leq C$ and $C =$ `N`. We only require that the relation work when `N` is determined (so that you can use arithmetic operators on `N` without causing suspensions).

   You must use **`choice`** in your solution.

   Figure 2 on page 4 shows several examples. These use `SolveAll`, which is provided in the testing zip file.

   Hints: I used helping relations named `GenFromTo` and `PairLessEqual` in my solution. `GenFromTo` is used to generate the numbers up to `N`. `PairLessEqual` generates pairs of the form `A#B` where `A =< B` and `B =< C`. Then using these I generate the triples.

5. (30 points) [UseModels]

   A *Pythagorean triple* is a triple of three positive integers $A\#B\#C$ such that $A^2 + B^2 = C^2$. For example, $3\#4\#5$ is a Pythagorean triple.

   Write, using the relational model, a relation `Pythagorean` such that `{Pythagorean N Result}` succeeds if `N` is a positive integer, and `Result` is a triple of the form $A\#B\#C$ where $A \leq B \leq C$, $C =$ `N`, and $A\#B\#C$ is a Pythagorean triple. We only require that the relation work when `N` is determined (so that you can use arithmetic operators on `N` without causing suspensions).

   Figure 3 on page 4 shows several examples. These use `SolveAll`, which is provided in the testing zip file.

   Hints: generate using `GenerateTriple` and then test to see that the triples are Pythagorean triples.

```
% $Id: ConsecutiveTest.oz,v 1.3 2007/11/29 01:55:29 leavens Exp leavens $
\insert 'Consecutive.oz'
\insert 'SolveFirst.oz'
\insert 'SolveAll.oz'
\insert 'TestingNoStop.oz'

{StartTesting 'SolveAll forwards'}
{Test {SolveAll proc {$ Suffix} {Consecutive nil 1 2 Suffix} end}
 '==' nil}
{Test {SolveAll proc {$ Suffix} {Consecutive 1|2|nil 1 2 Suffix} end}
 '==' [nil]}
{Test {SolveAll proc {$ Suffix} {Consecutive [1 2 3] 1 2 Suffix} end}
 '==' [[3]]}
{Test {SolveAll proc {$ Suffix}
                   {Consecutive [7 1 2 3 2 0 1 2 5] 1 2 Suffix}
                end}
 '==' [[3 2 0 1 2 5] [5]]}
{Test {SolveAll proc {$ Suffix}
                   {Consecutive [7 1 2 3 2 0 1 2 5] 1 3 Suffix}
                end}
 '==' nil}
{Test {SolveAll proc {$ Suffix}
                   {Consecutive [3 1 3 1 1] 3 1 Suffix}
                end}
 '==' [[3 1 1] [1]]}

{StartTesting 'SolveAll backwards'}
{Test {SolveAll proc {$ R}
                   X#Y=R
                in
                   {Consecutive [3 1 3 1 1] X Y [1 1]}
                end}
 '==' [1#3]}

{Test {SolveFirst proc {$ R}
                   X#Y=R
                in
                   {Consecutive [3 1 3 1 1] X Y nil}
                end}
 '==' 1#1}

{Test {SolveFirst proc {$ R}
                     {Consecutive R 3 4 [1 1]}
                  end}
 '==' [3 4 1 1]}

{Test {SolveFirst proc {$ R}
                     {Consecutive R 7 1 [5 6 9]}
                  end}
 '==' [7 1 5 6 9]}

{Test {SolveFirst proc {$ R}
                     R=ok
                     {Consecutive [7 1 5 6 9] 7 1 [5 6 9]}
                  end}
 '==' ok}
```

Figure 1: Testing code for Problem 3 on the previous page.

```
% $Id: GenerateTripleTest.oz,v 1.2 2008/11/25 23:16:35 leavens Exp leavens $
\insert 'GenerateTriple.oz'
\insert 'SolveAll.oz'
\insert 'TestingNoStop.oz'

{StartTesting 'GenerateTriple forwards'}
{Test {SolveAll proc {$ X} {GenerateTriple 1 X} end} '==' [1#1#1]}
{Test {SolveAll proc {$ X} {GenerateTriple 2 X} end}
 '==' [1#1#2 1#2#2 2#2#2]}
{Test {SolveAll proc {$ X} {GenerateTriple 3 X} end}
 '==' [1#1#3 1#2#3 2#2#3 1#3#3 2#3#3 3#3#3]}
{Test {SolveAll proc {$ X} {GenerateTriple 8 X} end}
 '==' [1#1#8
       1#2#8 2#2#8
       1#3#8 2#3#8 3#3#8
       1#4#8 2#4#8 3#4#8 4#4#8
       1#5#8 2#5#8 3#5#8 4#5#8 5#5#8
       1#6#8 2#6#8 3#6#8 4#6#8 5#6#8 6#6#8
       1#7#8 2#7#8 3#7#8 4#7#8 5#7#8 6#7#8 7#7#8
       1#8#8 2#8#8 3#8#8 4#8#8 5#8#8 6#8#8 7#8#8 8#8#8]}

{StartTesting 'GenerateTriple tests'}
{Test {SolveAll proc {$ X} {GenerateTriple 1 1#1#1} X=ok end} '==' [ok]}
{Test {SolveAll proc {$ X} {GenerateTriple 3 2#3#3} X=ok end} '==' [ok]}
{Test {SolveAll proc {$ X} {GenerateTriple 3 1#3#3} X=ok end} '==' [ok]}
```

Figure 2: Testing code for Problem 4 on page 2.

```
% $Id: PythagoreanTest.oz,v 1.1 2008/11/25 23:07:13 leavens Exp leavens $
\insert 'Pythagorean.oz'
\insert 'SolveAll.oz'
\insert 'TestingNoStop.oz'

{StartTesting 'Pythagorean'}
{Test {SolveAll proc {$ Triple} {Pythagorean 4 Triple} end} '==' nil}
{Test {SolveAll proc {$ Triple} {Pythagorean 5 Triple} end} '==' [3#4#5]}
{Test {SolveAll proc {$ Triple} {Pythagorean 10 Triple} end} '==' [6#8#10]}
{Test {SolveAll proc {$ Triple} {Pythagorean 13 Triple} end} '==' [5#12#13]}
{Test {SolveAll proc {$ Triple} {Pythagorean 14 Triple} end} '==' nil}
{Test {SolveAll proc {$ Triple} {Pythagorean 15 Triple} end} '==' [9#12#15]}
{Test {SolveAll proc {$ Triple} {Pythagorean 17 Triple} end} '==' [8#15#17]}
{Test {SolveAll proc {$ Triple} {Pythagorean 20 Triple} end} '==' [12#16#20]}
{Test {SolveAll proc {$ Triple} {Pythagorean 25 Triple} end}
 '==' [15#20#25 7#24#25]}
{Test {SolveAll proc {$ Triple} {Pythagorean 50 Triple} end}
 '==' [30#40#50 14#48#50]}
```

Figure 3: Testing code for Problem 5 on page 2.

Suggested Practice Problems

Want to explore relational programming a bit more? Then try these suggested practice problems.

6. (suggested practice) [UseModels]

   Write, using the relational model, a function

   ```
   PythagoreanSearch: <fun {$ <Int>}: <Stream <Int>#<Int>#<Int>>>
   ```

   such that `{PythagoreanSearch N}` returns a stream (i.e., a lazy list) of all the Pythagorean triples of the form $A\#B\#C$, where $C \leq$ `N`.

   Figure 4 shows several examples.

```
% $Id: PythagoreanSearchTest.oz,v 1.2 2008/11/25 23:40:32 leavens Exp leavens $
\insert 'PythagoreanSearch.oz'
\insert 'TestingNoStop.oz'

{StartTesting 'PythagoreanSearch'}
{Test {List.take {PythagoreanSearch 4} 10} '==' nil}
{Test {List.take {PythagoreanSearch 5} 10} '==' [3#4#5]}
{Test {List.take {PythagoreanSearch 10} 10} '==' [3#4#5 6#8#10]}
{Test {List.take {PythagoreanSearch 20} 15}
 '==' [3#4#5 6#8#10 5#12#13 9#12#15 8#15#17 12#16#20]}
{Test {List.take {PythagoreanSearch 50} 20}
 '==' [3#4#5 6#8#10 5#12#13 9#12#15 8#15#17 12#16#20
       15#20#25 7#24#25 10#24#26 20#21#29 18#24#30
       16#30#34 21#28#35 12#35#37 15#36#39 24#32#40
       9#40#41 27#36#45 30#40#50 14#48#50]}
{StartTesting 'done'}
```

Figure 4: Testing code for Problem 6.

   Hints: use `Solve` to perform an encapsulated search, using `Pythagorean`.

7. (suggested practice) [UseModels]

   Do problem 4 in section 9.8 of our textbook [RH04]. Note when facts in the database given in the problem have (dataflow/logical) variables in them, which are capitalized. The library doesn't seem to be open on weekends.

   Hint: use generate and test.

   You must show by writing your own tests that your code works as required. Since you are designing the program, testing is up to you and an important part of this exercise.

References

[RH04] Peter Van Roy and Seif Haridi. *Concepts, Techniques, and Models of Computer Programming*. The MIT Press, Cambridge, Mass., 2004.