# Homework 2: Declarative Computation Model

Due: 20 points worth of normal problems on Thursday, September 7, 2006; remaining problems and extra credit on Tuesday September 12, 2006.

In this homework you will learn about the declarative computational model, linguistic abstractions and syntactic sugars, and exception handling.

For all 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). Hand in a printout of your code and the output of your testing, for all questions that require code.

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 2 of the textbook [RH04] You may also want to refer to reference and tutorial material on the Mozart/Oz web site `http://www.mozart-oz.org/`.

## Textbook Problems

The following problems are from the textbook [RH04, section 2.9].

1. Select enough of the following to achieve a total of at least 40 possible points.
   (a) (20 points) Problem 4, if and case statements.
   (b) (15 points) Problem 6, the case statement again.
   (c) (10 points) Problem 8, control abstraction
   (d) (25 points) Problem 9, tail recursion.
   (e) (10 points) Problem 10, expansion into kernel syntax.
   (f) (10 points) Problem 12, exceptions with a finally clause.
   (g) (10 points) Problem 13, unification.
2. (extra credit) Do as many of the above selected problems as you find interesting. The extra credit points you receive will be based on the possible points noted above.
3. Other extra credit problems from the textbook.
   (a) (10 points; extra credit) Do Problem 2, contextual environment.
   (b) (20 points; extra credit) Using the operational semantics presented in class, trace the execution of the code in problem 7.

## Other Problems

4. (40 points; extra credit) Write code in Oz that translates code written in the extended language of chapter 2 into the kernel language. You can use parsing or other tools that come with Oz. The input should be text and the output should also be text. (Hint: you may want to use GUMP.)

   For example, given the input

   ```
   local Th = 3 in {Browse Th*Th} end
   ```

   it would produce the output

   ```
   local Th in Th=3 local X in X=Th*Th {Browse X} end end
   ```

   (Such outputs may be easier to read if indented, but that is not necessary for this problem.)

5. (60 points; extra credit) Write code in Oz that prints out a series of steps that the operational semantics of Oz takes when executing a kernel program. The input to this program should be a text string that is in the kernel language.

   For example, given the input

   ```
   local Th in Th=3 local X in X=Th*Th {Browse X} end end
   ```

   it would produce output similar to the following.

   ```
   ([(local Th in Th=3 local X in X=Th*Th {Browse X} end end], {}), {})
   -->
   ([(Th=3 local X in X=Th*Th {Browse X} end, {Th-->x1})], {x1})
   -->
   ([(Th=3, {Th-->x1}) (local X in X=Th*Th {Browse X} end, {Th-->x1})], {x1})
   -->
   ([(local X in X=Th*Th {Browse X} end, {Th-->x1})], {x1=3})
   -->
   ([(X=Th*Th, {X-->x2, Th-->x1}) ({Browse X}, {X-->x2, Th-->x1})], {x1=3,x2})
   -->
   ([({Browse X}, {X-->x2, Th-->x1})], {x1=3, x2=9})
   ```

   At the end, the machine gets "stuck" when trying to execute `Browse`, since it is not found in the environment.

6. (50 points total; extra credit) Read a paper on one of the following topics:

   - operational semantics [Ast91, Plo77, Plo81] or some chapters of Hennessy's book [Hen90], or
   - Landin's classic paper "The Next 700 Programming Languages" [Lan66].

   You can also find some other published research article in a journal or conference proceedings related to the topics in this chapter. (By a published research article, I mean an article that is not in a trade journal (e.g., it has references at the end), and that is from a refereed journal or conference. *Publication* means the article actually appeared in print (or an on-line refereed venue), and was not just submitted somewhere. So beware of technical reports on the web. It's okay to get a copy of a published article from the web, although I highly encourage you to physically go to the library.)

   Write a short (1 or 2 page maximum) review of the article, stating:

   - (10 points) what the problem was that the article was claiming to solve,
   - (20 points) the main points made in the article and what you learned from it,
   - (20 points) what contribution it make vs. any related work mentioned in the article.

   In your writing, be sure to digest the material; that is, don't just select various quotes from the article and string them together, instead, really summarize it. If you quote any text from the paper, be sure to mark the quotations with quotation marks (" and ") and give the page number(s).

   If you do a different article than one of those mentioned above, then hand in a copy of the article with your review. In any case, be sure to explicitly cite the paper you reviewed.

## References

[Ast91] Edigio Astesiano. Inductive and operational semantics. In E. J. Neuhold and M. Paul, editors, *Formal Description of Programming Concepts*, IFIP State-of-the-Art Reports, pages 51–136. Springer-Verlag, New York, NY, 1991.

[Hen90] Matthew Hennessy. *The Semantics of Programming Languages: an Elementary Introduction using Structural Operational Semantics*. John Wiley and Sons, New York, NY, 1990.

[Lan66] P. J. Landin. The next 700 programming languages. *Communications of the ACM*, 9(3):157–166, March 1966.

[Plo77] G. D. Plotkin. LCF considered as a programming language. *Theoretical Computer Science*, 5:223–255, 1977.

[Plo81] Gordon Plotkin. A structural approach to operational semantics. Technical Report DAIMI FN-19, Aarhus University, September 1981.

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