Project 2: Programming in ML, Prolog, and the λ-calculus — due Monday 30 April

Total number of points available on this project is 200. Full credit is equivalent to 0 points. In other words, this project is entirely optional.

One goal of this project is to establish that logic programming languages and functional programming languages have equal expressive power, if not equal expressive convenience. Another goal is to establish that pure λ-calculus has equal expressive power as any functional programming languages, though it's cumbersome to use.

In class we discussed an implementation of a unification algorithm and of an interpreter for the Prolog core language (pure logic programming). The code is available through the course web site.

In Project 1 you implemented an interpreter for pure λ-calculus in ML.

Tasks:

1. (30 pts.)

   Write an interpreter λ_P for pure λ-calculus in Prolog. Test, debug, and run under SWI-Prolog, using the same λ-term reductions as in Project 1.

   This should be a straightforward translation into Prolog of the program you wrote in ML for Project 1. You may leave out the fancy input and output handling.

2. (60 pts.)

   Write an interpreter $\hat{P}_\lambda$ for the Prolog core language in pure λ-calculus. This should be a straightforward rendering (desugaring) of the interpreter we showed in class. Test, debug, and run under the interpreter for pure λ-calculus that you wrote in Project 1, using the various Prolog programs you wrote earlier (recognizers for regular, context-free, and context-sensitive languages).

   For efficiency reasons, it may be useful to have built-in lists in the interpreter for pure λ-calculus. It certainly will be useful to have strictness anotations; since original code is in ML, most of the λ-abstractions you write can be strict.

3. (10 pts.)

   If you wrote λ_P so that it uses features of Prolog (P) not present in the Prolog core language ($\hat{P}$), for instance, the cut (`!`), rewrite it so it doesn't, and call the result $\lambda_{\hat{P}}$.

4. (30 pts.)

   Test, debug, and run $\lambda_{\hat{P}}$ on top of $\hat{P}_\lambda$.

5. (30 pts.)

   Test, debug, and run $\hat{P}_\lambda$ on top of $\lambda_{\hat{P}}$.

6. (40 pts.)

   Report your observations about the speed of various interpreters, ease of implementation, syntactic representation of core-Prolog terms and λ-terms, etc.