CS6202 – Assignment 2
Deadline : 8pm 25 Sep 2006

September 17, 2006

1 Installing Standard ML

You will need to install a newer version of Standard ML to complete this project, as we will be using a lexer and parser generator. Go to the following website and follow the instruction there to download and install SML NJ. There are Unix and MS Windows distributions.

`http://www.smlnj.org/dist/working/110.59/index.html`

2 An Inference Task

In this assignment, you will need to implement a type inference algorithm for a simple language. The syntax of the language is given below. **Boldface** words are keywords. Note that function names are simply declared as let-bound variables, and may by recursively defined.

$$\begin{array}{rcll}
e & ::= & k & //\ \textit{integer} \\
 & \mid & v & //\ \textit{variable or function name} \\
 & \mid & (e_1, e_2) & //\ \textit{pair} \\
 & \mid & (\mathbf{lam}\ x \cdot e) & //\ \textit{lambda abstraction} \\
 & \mid & e_1\ e_2 & //\ \textit{function applications} \\
 & \mid & \mathbf{let}\ v = e_1\ \mathbf{in}\ e_2 & //\ \mathbf{let}\ \textit{expression, may be recursive}
\end{array}$$

Your code should discover type information for variables and function names. Type information should be recorded in an output language with the following syntax. Note the only differences are that **let** and **lam**bda variables are annotated with types. To support polymorphism, we allow type variables of form X to be used/inferred.

$$\begin{array}{rcll}
e & ::= & k & //\ \textit{integer} \\
 & \mid & v & //\ \textit{variable or function} \\
 & \mid & (e_1, e_2) & //\ \textit{pair} \\
 & \mid & (\mathbf{lam}\ x : t \cdot e) & //\ \textit{lambda abstraction} \\
 & \mid & e_1\ e_2 & //\ \textit{function applications} \\
 & \mid & \mathbf{let}\ v : t = e_1\ \mathbf{in}\ e_2 & //\ \mathbf{let}\ \textit{expression, may be recursive} \\
t & ::= & \mathbf{int} & //\ \textit{integer} \\
 & \mid & X & //\ \textit{type variable} \\
 & \mid & (t, t) & //\ \textit{pairs} \\
 & \mid & t \rightarrow t & //\ \textit{function}
\end{array}$$

Example

Input program:

$\textbf{let}\ f = (\textbf{lam}\ x \cdot 2)\ \textbf{in}\ (f\ 0)$

Output program:

$\textbf{let}\ f : A \to \textbf{int} = (\textbf{lam}\ x : A \cdot 2)\ \textbf{in}\ (f\ 0)$

Input program:

$\textbf{let}\ f = (\textbf{lam}\ x \cdot (\textbf{lam}\ y \cdot (x, y))\ \textbf{in}\ (f\ 0\ 1)$

Output program:

$\textbf{let}\ f : A \to (B \to (A, B)) = (\textbf{lam}\ x : A \cdot (\textbf{lam}\ y : B \cdot (x, y))\ \textbf{in}\ (f\ 0\ 1)$

3 Supplied Code

You are provided with a lexer and parser. You are also provided with data structure to store input AST and a simple pretty printer for the AST. The contents of the source files are as follows:

- `absyn.ml` Abstract syntax tree
- `exp.grm` Grammar definition file for ML-Yacc. Note that for simplicity, the concrete syntax for function application is (e1 e2).
- `exp.lex` Lexer definition file for ML-Lex.
- `sources.cm` CM input
- `link.sml, parse.sml, parse.sml` Various glue code

Make the directory storing the above files current working directory of `sml`. Run `CM.make ‘‘sources.cm’’;` at the SML/NJ command line to generate the parser/lexer and compile other source. Once everything is compiled and loaded, you can use `Parse.prog_parse` to parse a string, and `Parse.file_parse` to parse a file. Below is a sample session.

```
$ sml
Standard ML of New Jersey v110.59 [built: Mon Sep 11 11:47:37 2006]
- CM.make "sources.cm";
[autoloading]
[library $smlnj/cm/cm.cm is stable]
[library $smlnj/internal/cm-sig-lib.cm is stable]
[library $/pgraph.cm is stable]
[library $smlnj/internal/srcpath-lib.cm is stable]
[library $SMLNJ-BASIS/basis.cm is stable]
[autoloading done]
[scanning sources.cm]
[library $/ml-yacc-lib.cm is stable]
[library $SMLNJ-ML-YACC-LIB/ml-yacc-lib.cm is stable]
[loading (sources.cm):interface.sml]
[loading (sources.cm):absyn.sml]
[loading (sources.cm):exp.grm.sig]
[loading (sources.cm):exp.grm.sml]
```

```
[loading (sources.cm):exp.lex.sml]
[loading (sources.cm):parse.sml]
[loading (sources.cm):link.sml]
[New bindings added.]
val it = true : bool
- Parse.prog_parse "let x = 1 in (f x)";
val it = - : Absyn.absyn
- Absyn.print_ast it;
val it = "let x = 1 in (f x)" : string
-
```