CSE-321 Assignment 5 (100 points)

gla@postech

Due at 11:59pm, April 18 (Thursday)

In this assignment, you will implement the type system of TML (Typed ML) which is essentially the simply typed λ-calculus without the type void.

The abstract syntax for TML is shown below. Compared with the abstract syntax for the simply typed λ-calculus discussed in class, it includes another base type int for integers, but omits the type void which is useless in practice. $\widehat{n}$ denotes an integer n, and plus and minus are arithmetic operators on integers; eq tests two integers for equality.

$$\begin{array}{rrcl}
\textsf{type} & A & ::= & \textsf{bool} \mid \textsf{int} \mid A \to A \mid A \times A \mid \textsf{unit} \mid A + A \\
\textsf{expression} & e & ::= & x \mid \lambda x\!:\!A.\,e \mid e\ e \mid (e, e) \mid \textsf{fst}\ e \mid \textsf{snd}\ e \mid () \mid \\
 & & & \textsf{inl}_A\ e \mid \textsf{inr}_A\ e \mid \textsf{case}\ e\ \textsf{of inl}\ x.\,e \mid \textsf{inr}\ x.\,e \mid \textsf{fix}\ x\!:\!A.\,e \mid \\
 & & & \textsf{true} \mid \textsf{false} \mid \textsf{if}\ e\ \textsf{then}\ e\ \textsf{else}\ e \mid \widehat{n} \mid \textsf{plus} \mid \textsf{minus} \mid \textsf{eq} \\
\textsf{typing context} & \Gamma & ::= & \cdot \mid \Gamma, x : A
\end{array}$$

The concrete syntax for TML is as follows. All entries in the right side of the definition below are arranged in the same order that their counterparts in the abstract syntax appear in the definition above:

$$\begin{array}{rrcl}
\text{type} & A & ::= & \texttt{bool} \mid \texttt{int} \mid A\ \texttt{->}\ A \mid A\ \texttt{*}\ A \mid \texttt{unit} \mid A\ \texttt{+}\ A \mid \texttt{(}A\texttt{)} \\
\text{expression} & e & ::= & x \mid \texttt{fn}\ x\ \texttt{:}\ A\ \texttt{=>}\ e \mid e\ e \mid \texttt{(}e\texttt{,}\ e\texttt{)} \mid \texttt{fst}\ e \mid \texttt{snd}\ e \mid \texttt{()} \mid \\
 & & & \texttt{inl (}A\texttt{)}\ e \mid \texttt{inr (}A\texttt{)}\ e \mid \texttt{case}\ e\ \texttt{of inl}\ x\ \texttt{=>}\ e\ \texttt{|}\ \texttt{inr}\ x\ \texttt{=>}\ e \mid \\
 & & & \texttt{fix}\ x\ \texttt{:}\ A\ \texttt{=>}\ e \mid \\
 & & & \texttt{true} \mid \texttt{false} \mid \texttt{if}\ e\ \texttt{then}\ e\ \texttt{else}\ e \mid \widehat{n} \mid \texttt{+} \mid \texttt{-} \mid \texttt{=} \mid \\
 & & & \texttt{let}\ x\texttt{:}A\ \texttt{=}\ e\ \texttt{in}\ e' \mid \texttt{let rec}\ x\texttt{:}A\ \texttt{=}\ e\ \texttt{in}\ e' \mid \\
 & & & \texttt{(}e\texttt{)} \\
\text{integer} & \widehat{n} & ::= & \cdots \mid \texttt{~2} \mid \texttt{~1} \mid \texttt{0} \mid \texttt{1} \mid \texttt{2} \mid \cdots
\end{array}$$

We add two forms of syntactic sugar:

$$\begin{array}{rcl}
\texttt{let}\ x\texttt{:}A\ \texttt{=}\ e\ \texttt{in}\ e' & \text{for} & \texttt{(fn}\ x\texttt{:}A\ \texttt{=>}\ e'\texttt{)}\ e \\
\texttt{let rec}\ x\texttt{:}A\ \texttt{=}\ e\ \texttt{in}\ e' & \text{for} & \texttt{(fn}\ x\texttt{:}A\ \texttt{=>}\ e'\texttt{) (fix}\ x\texttt{:}A\texttt{.}\ e\texttt{)}
\end{array}$$

$\texttt{(}A\texttt{)}$ is the same as A, and is used to alter the default right associativity of `->`. For example, A_1 `->` A_2 `->` A_3 is equal to A_1 `-> (`A_2 `->` A_3`)` which is different from `(`A_1 `->` A_2`) ->` A_3. Similarly `(`e`)` is the same as e, and is used to alter the default left associativity of applications. For example, $e_1\ e_2\ e_3$ is equal to `(`$e_1\ e_2$`)` e_3 which is different from e_1 `(`$e_2\ e_3$`)`.

The goal of this assignment is to implement the type system of TML shown in Figure 1.

$$\frac{x : A \in \Gamma}{\Gamma \vdash x : A}\,\mathsf{Var} \qquad \frac{\Gamma, x : A \vdash e : B}{\Gamma \vdash \lambda x{:}A.\, e : A {\rightarrow} B}\,{\rightarrow}\mathsf{I} \qquad \frac{\Gamma \vdash e : A {\rightarrow} B \quad \Gamma \vdash e' : A}{\Gamma \vdash e\; e' : B}\,{\rightarrow}\mathsf{E}$$

$$\frac{\Gamma \vdash e_1 : A_1 \quad \Gamma \vdash e_2 : A_2}{\Gamma \vdash (e_1, e_2) : A_1 \times A_2}\,\times\mathsf{I} \qquad \frac{\Gamma \vdash e : A_1 \times A_2}{\Gamma \vdash \mathsf{fst}\; e : A_1}\,\times\mathsf{E}_1 \qquad \frac{\Gamma \vdash e : A_1 \times A_2}{\Gamma \vdash \mathsf{snd}\; e : A_2}\,\times\mathsf{E}_2 \qquad \frac{}{\Gamma \vdash () : \mathsf{unit}}\,\mathsf{Unit}$$

$$\frac{\Gamma \vdash e : A_1}{\Gamma \vdash \mathsf{inl}_{A_2}\; e : A_1{+}A_2}\,{+}\mathsf{I}_\mathsf{L} \qquad \frac{\Gamma \vdash e : A_2}{\Gamma \vdash \mathsf{inr}_{A_1}\; e : A_1{+}A_2}\,{+}\mathsf{I}_\mathsf{R}$$

$$\frac{\Gamma \vdash e : A_1{+}A_2 \quad \Gamma, x_1 : A_1 \vdash e_1 : C \quad \Gamma, x_2 : A_2 \vdash e_2 : C}{\Gamma \vdash \mathsf{case}\; e\; \mathsf{of}\; \mathsf{inl}\; x_1.\, e_1 \mid \mathsf{inr}\; x_2.\, e_2 : C}\,{+}\mathsf{E}$$

$$\frac{\Gamma, x : A \vdash e : A}{\Gamma \vdash \mathsf{fix}\; x{:}A.\, e : A}\,\mathsf{Fix}$$

$$\frac{}{\Gamma \vdash \mathsf{true} : \mathsf{bool}}\,\mathsf{True} \qquad \frac{}{\Gamma \vdash \mathsf{false} : \mathsf{bool}}\,\mathsf{False} \qquad \frac{\Gamma \vdash e : \mathsf{bool} \quad \Gamma \vdash e_1 : A \quad \Gamma \vdash e_2 : A}{\Gamma \vdash \mathsf{if}\; e\; \mathsf{then}\; e_1\; \mathsf{else}\; e_2 : A}\,\mathsf{If}$$

$$\frac{}{\Gamma \vdash \widehat{n} : \mathsf{int}}\,\mathsf{Int}$$

$$\frac{}{\Gamma \vdash \mathsf{plus} : \mathsf{int} \times \mathsf{int} {\rightarrow} \mathsf{int}}\,\mathsf{Plus} \qquad \frac{}{\Gamma \vdash \mathsf{minus} : \mathsf{int} \times \mathsf{int} {\rightarrow} \mathsf{int}}\,\mathsf{Minus} \qquad \frac{}{\Gamma \vdash \mathsf{eq} : \mathsf{int} \times \mathsf{int} {\rightarrow} \mathsf{bool}}\,\mathsf{Eq}$$

Figure 1: Type system of TML

Programming instruction

Download hw5-stub.zip from the course webpage and unzip it on your working directory. It will create a bunch of files on the working directory and two subdirectories parsing and util. You don't need to look at these subdirectories.

First see the structure Tml in tml.sml which has signature TML.

```
structure Tml : TML =
struct
  type var = string
  datatype tp =
    ...
  datatype exp =
    ...
end
```

The datatypes tp and exp correspond to the syntactic categories type and expression, respectively. Read the comments in the file and make sure that you understand what each data constructor is for.

Next see typing-sig.sml and typing.sml. The goal is to implement the function typing in the structure Typing:

```
    val typing : context -> Tml.exp -> Tml.tp
```

That is, typing takes a typing context Γ of type Typing.context and an expression e of type Tml.exp, and returns a type A such that $\Gamma \vdash e : A$; it raises an exception TypeError if e does not typecheck, *i.e.*, there is no A such that $\Gamma \vdash e : A$.

Note that you will decide the definition of type Typing.context yourself. In the stub file, Typing.context is defined as unit, but you should replace it by an appropriate type for typing contexts. After deciding the definition of Typing.context, you have to implement the function createEmptyContext which returns an empty typing context:

```
val createEmptyContext : unit -> context
```

Then try to implement the rule Var to make sure that you can retrieve an individual element (which is a type binding $x : A$) from a typing context.

You have to think about how to represent $\Gamma, x : A$ which is required by the rules $\to$I, $+$E, and Fix. If we wanted to maintain the invariant that variables in a typing context are all distinct, we would need α-conversion for these rules. It turns out, however, that we do not need α-conversion at all! Think about this. (Or take a wrong path and implement α-conversion, if you like.)

As another tip, if you are tempted to use `(Tml.var * Tml.tp) list` for `Typing.context`, think again. `(Tml.var * Tml.tp) list` is definitely a good candidate for the definition of `Typing.context` (and indeed a reasonably good solution), but try to come up with a better and more elegant definition.

To test your code, you want to use those functions in the structure `Loop` in `loop.sml`. (You don't actually need to read `loop-sig.sml` and `loop.sml`.) As in Assignment 4, make sure that you have installed SML/NJ 110.58 which allows us to enable the lazy evaluation mode of SML (used by the parser). At the command prompt, type `Control.lazysml := true;` to enable the lazy evaluation mode; otherwise you will later get an error message like `syntax error: replacing ID with LAZY`:

```
- Control.lazysml := true;
[autoloading]
...
[autoloading done]
val it = () : unit
-
```

Then compile the source files using the Compile Manager:

```
- CM.make "sources.cm";
...
[New bindings added.]
val it = true : bool
-
```

Then open the structure `Loop`:

```
- open Loop;
opening Loop
...
-
```

You can test your `typing` function by typing `loop (step show);` to enter the TML mode. Then enter a TML expression followed by the semicolon symbol `;`. If the expression typechecks, its type will be displayed along with the result of parsing; otherwise an error message will be reported.

```
- loop (step show);
Tml> fn x : int => x ;
fn x : int => x : int -> int
Tml> x;
x has no type.
```

If you make a change to your `typing` function, be sure to run the Compile Manager and open the structure `Loop` again. Otherwise it will continue to use the previous version of the `typing` function!

Submission instruction

When you have the file `typing.sml` ready for submission, rename the `Typing` structure in it to your Hemos ID. Then copy it to your hand-in directory. For example, if your Hemos ID is `foo`, rename the `Typing` structure in `typing.sml` to `foo` and copy `typing.sml` to:

```
/afs/postech.ac.kr/class/cse/cs321/handin/hw5/foo/
```