CSED321 Assignment 6 - *Type System*

Due Tuesday, April 16

The goal of this assignment is to implement the type system of TML shown in Figure 1, which is essentially the simply typed λ-calculus. The abstract syntax for TML is as follows.

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

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

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

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

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

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

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

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

$$\frac{}{\Gamma \vdash \widehat{n} : \textsf{int}}\ \textsf{Int} \qquad \frac{}{\Gamma \vdash \textsf{plus} : \textsf{int} \times \textsf{int} \to \textsf{int}}\ \textsf{Plus}$$

$$\frac{}{\Gamma \vdash \textsf{minus} : \textsf{int} \times \textsf{int} \to \textsf{int}}\ \textsf{Minus} \qquad \frac{}{\Gamma \vdash \textsf{eq} : \textsf{int} \times \textsf{int} \to \textsf{bool}}\ \textsf{Eq}$$

Figure 1: The type system of TML (Typed ML)

Programming Instruction

Download the zip file `hw6.zip` from the course webpage or by running `receive` on the server `141.223.163.223`, and unzip it on your working directory. It will create a bunch of files.

First see `tml.ml` which declares a structure `Tml`. The types `tp` and `exp` correspond to the syntactic categories type and expression, respectively.

```
type var = string

type tp =                    (* types *)
  ...

type exp =                   (* expressions *)
  ...
```

Next see `hw6.mli` and `hw6.ml`. The goal is to implement the function `typing`, which takes a typing context Γ of type `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 there is no A such that $\Gamma \vdash e : A$.

```
exception TypeError
type context
val createEmptyContext : unit -> context
val typing : context -> Tml.exp -> Tml.tp
...
```

You will also decide the definition of type `context`. In the stub file, `context` is defined as `unit`, but you should replace it by an appropriate type for typing contexts, and implement the function `createEmptyContext` which returns an empty typing context.

After implementing the function `typing` in `hw6.ml`, run `make` to compile the sources files. There are two ways to test your code. First, you can run `main`. At the TML prompt, enter a TML expression followed by a semicolon. (The syntax of TML will be given shortly.)

```
$ ./main
Tml> fun x : int -> int -> x;
Syntax error
Tml> fun x : (int -> int) -> x;
(fun x : (int -> int) -> x) : ((int -> int) -> (int -> int))
Tml> x;
x has no type.
```

Alternatively you can use those functions in `loop.ml` in the interactive mode of OCaml. (You don't actually need to read `loop.ml`.) At the OCaml prompt, type `#load "lib.cma";;` to load the library for this assignment. Then open the structure `Loop`:

```
$ ocaml
        OCaml version 4.05.0

# #load "lib.cma";;
# open Loop;;
```

You type `loop show;;`, and then enter a TML expression followed by a semicolon.

```
# loop show;;
Tml> fun x : int -> x;
(fun x : int -> x) : (int -> int)
```

Remark. Think carefully about how to represent $\Gamma, x : A$ which is required by the rules $\rightarrow$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! (Or take a wrong path and implement α-conversion, if you like.)

The Syntax of TML

The concrete syntax for TML is defined as a subset of OCaml. All entries in the definition below are arranged in the same order that their counterparts in the abstract syntax above; e.g., `match` e `with inl` x `->` e `|` `inr` x `->` e is related to $\mathsf{case}\ e\ \mathsf{of}\ \mathsf{inl}\ x.\, e \mid \mathsf{inr}\ x.\, e$.

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

$\texttt{(}A\texttt{)}$ 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`)`, not `(`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, not e_1 `(`$e_2\ e_3$`)`. We add two forms of syntactic sugar:

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

Submission Instruction

1. Make sure that you can compile `hw6.ml` by running `make`.
2. When you have the file `hw6.ml` ready for submission, run the `handin` command in the same directory, and your file will be submitted automatically.