CSE-321 Assignment 5 (100 points)

gla@postech

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.

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

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:

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

We add two forms of syntactic sugar:

$$\texttt{let}\ x{:}A\ \texttt{=}\ e\ \texttt{in}\ e' \quad \text{for} \quad \texttt{(fn}\ x{:}A\ \texttt{=>}\ e'\texttt{)}\ e$$
$$\texttt{let rec}\ x{:}A\ \texttt{=}\ e\ \texttt{in}\ e' \quad \text{for} \quad \texttt{(fn}\ x{:}A\ \texttt{=>}\ e'\texttt{)}\ \texttt{(fix}\ x{:}A.\ e\texttt{)}$$

`(`A`)` 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.

Programming instruction

Download `hw5.zip` from the course webpage or the handin directory, and unzip it on your working directory. It will create a bunch of files on the working directory.

First see `tml.ml` which declares a structure `Tml`:

```
type var = string
type tp =
  ...
type exp =
  ...
```

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

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

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

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

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

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

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

$$\frac{}{\Gamma \vdash \mathsf{plus} : \mathsf{int} \times \mathsf{int} \to \mathsf{int}}\ \mathsf{Plus} \quad \frac{}{\Gamma \vdash \mathsf{minus} : \mathsf{int} \times \mathsf{int} \to \mathsf{int}}\ \mathsf{Minus} \quad \frac{}{\Gamma \vdash \mathsf{eq} : \mathsf{int} \times \mathsf{int} \to \mathsf{bool}}\ \mathsf{Eq}$$

Figure 1: Type system of TML

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.mli` and `typing.ml`. 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.

After implementing the function `typing` in `typing.ml`, run the command `make` to compile the sources files.

```
gla@ubuntu:~/temp/hw5$ make
ocamlc -thread  -c tml.ml -o tml.cmo
ocamlyacc parser.mly
26 shift/reduce conflicts.
ocamlc -c parser.mli
ocamlc -c parser.ml
ocamllex lexer.mll
```

```
67 states, 4625 transitions, table size 18902 bytes
ocamlc -c lexer.ml
ocamlc -thread  -c typing.mli -o typing.cmi
ocamlc -thread  -c typing.ml -o typing.cmo
ocamlc -thread  -c inout.ml -o inout.cmo
ocamlc -thread  -c eval.ml -o eval.cmo
ocamlc -thread  -c loop.ml -o loop.cmo
ocamlc -thread  -c hw5.ml -o hw5.cmo
ocamlc -o lib.cma -a tml.cmo parser.cmo lexer.cmo typing.cmo inout.cmo eval.cmo loop.cmo
ocamlc -o hw5 lib.cma hw5.cmo
```

There are two ways to test your code in `typing.ml`. First you can run `hw5`. At the TML prompt, enter a TML expression followed by the semicolon symbol `;`. (The syntax of TML will be given shortly.) Each time you press the return key, a reduced expression is displayed.

```
gla@ubuntu:~/temp/hw5$ ./hw5
Tml> fn : int => x ;
Syntax error
Tml> fn x : int => x ;
(fn x : int => x) : 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`:

```
gla@ubuntu:~/temp/hw5$ ocaml
        OCaml version 4.01.0

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

You type `loop (step show);;` at the OCAML prompt, and then enter a TML expression followed by the semicolon symbol `;`.

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

Submission instruction

1. Make sure that you can compile `typing.ml` by running `make`.
2. When you have your file `typing.ml` ready for submission, copy it to your hand-in directory on `programming.postech.ac.kr`. For example, if your Hemos ID is `foo`, copy your `typing.ml` to:

   `/home/class/cs321/handin/foo/`