Homework 5
ENE4014 Programming Languages, Spring 2020
Woosuk Lee
due: 6/13(Sat), 24:00

As usual, skeleton code will be provided (before you start, see README.md).

Exercise 1 Consider the following language:

$$
\begin{array}{rcl}
P & \to & E \\
E & \to & n \\
& | & \texttt{true} \\
& | & \texttt{false} \\
& | & x \\
& | & E + E \mid E - E \mid E * E \mid E/E \\
& | & E - E \\
& | & \texttt{iszero}\ E \\
& | & \texttt{if}\ E\ \texttt{then}\ E\ \texttt{else}\ E \\
& | & \texttt{let}\ x = E\ \texttt{in}\ E \\
& | & \texttt{letrec}\ f(x) = E\ \texttt{in}\ E \\
& | & \texttt{proc}\ x\ E \\
& | & E\ E
\end{array}
$$

Types for the language are defined as follows:

$$
\begin{array}{rcl}
T & \to & \mathrm{int} \\
& | & \mathrm{bool} \\
& | & T \to T \\
& | & \alpha
\end{array}
$$

where α is a type variable. Implement the following type-inference function:

$$\texttt{typeof} : P \to T$$

which takes a program and returns its type if the program is well-typed. When the program is ill-typed, `typeof` should raise an exception `TypeError`.

As discussed in the class, `typeof` is defined with two functions: one for generating type equations and the other for solving the equations. Complete the implementation of these two functions in the skeleton code:

```
gen_equations :  TEnv.t -> exp -> typ -> typ_eqn
```

```
solve :  typ_eqn -> Subst.t
```

Modules for type environments (`TEnv`) and substitutions (`Subst`), as well as the operations of applying substitutions to types (`Subst.apply`) and extending substitutions (`Subst.extend`), are provided.

Exercise 2 Define the function

$$\texttt{expand} : P \to P$$

that transforms an expression into a semantically-equivalent expression where every let-bound variable in the original expression gets replaced by its definition. For examples,

- $\texttt{expand}(\texttt{let}\ x = 1\ \texttt{in}\ x)$ produces 1.
- $\texttt{expand}(\texttt{let}\ f = \texttt{proc}\ (x)\ x\ \texttt{in}\ \texttt{if}\ (f\ (\texttt{iszero}\ 0))\ \texttt{then}\ (f\ 11)\ \texttt{else}\ (f\ 22))$ produces

  $$(\texttt{if}\ ((\texttt{proc}\ (x)\ x)\ \texttt{iszero}\ 0)\ \texttt{then}\ ((\texttt{proc}\ (x)\ x)\ 11)\ \texttt{else}\ ((\texttt{proc}\ (x)\ x)\ 22)).$$

- Unused definitions should not go away. For example,

  $$\texttt{expand}(\texttt{let}\ x = \texttt{iszero}\ \texttt{true}\ \texttt{in}\ 2)$$

  shoud return $\texttt{let}\ x = \texttt{iszero}\ \texttt{true}\ \texttt{in}\ 2$ instead of 2.

As discussed in class, the function `expand` can be used for implementing the let-polymorphic type system. The type checker $\texttt{typeof} : P \to T$ in Problem 1 does not support polymorphism and would not accept the following program:

$$p = \texttt{let}\ f = \texttt{proc}\ (x)\ x\ \texttt{in}\ \texttt{if}\ (f\ (\texttt{iszero}\ 0))\ \texttt{then}\ (f\ 11)\ \texttt{else}\ (f\ 22)$$

However, the same type checking algorithm with `expand` (i.e., $\texttt{typeof}(\texttt{expand}(p))$) will succeed.