Homework 5
COSE212, Fall 2022

Hakjoo Oh

Due: 11/27, 23:59

Problem 1 Consider the language ML^- from HW3:

P	$\to$	E	
E	$\to$	`()`	unit
	$\mid$	`true` $\mid$ `false`	booleans
	$\mid$	n	integers
	$\mid$	x	variables
	$\mid$	E `+` $E \mid E$ `-` $E \mid E$ `*` $E \mid E$ `/` E	arithmetic
	$\mid$	E `=` $E \mid E$ `<` E	comparison
	$\mid$	`not` E	negation
	$\mid$	`nil`	empty list
	$\mid$	E `::` E	list cons
	$\mid$	E `@` E	list append
	$\mid$	`head` E	list head
	$\mid$	`tail` E	list tail
	$\mid$	`isnil` E	checking empty list
	$\mid$	`if` E `then` E `else` E	if
	$\mid$	`let` x `=` E `in` E	let
	$\mid$	`letrec` $f(x)$ `=` E `in` E	recursion
	$\mid$	`letrec` $f(x_1) = E_1$ and $g(x_2) = E_2$ `in` E	mutual recursion
	$\mid$	`proc` x E	function definition
	$\mid$	E E	function application
	$\mid$	`print` E	print
	$\mid$	$E; E$	sequence

In OCaml datatype:

```
type program = exp
and exp =
  | UNIT
  | TRUE
  | FALSE
```

```
  | CONST of int
  | VAR of var
  | ADD of exp * exp
  | SUB of exp * exp
  | MUL of exp * exp
  | DIV of exp * exp
  | EQUAL of exp * exp
  | LESS of exp * exp
  | NOT of exp
  | NIL
  | CONS of exp * exp
  | APPEND of exp * exp
  | HEAD of exp
  | TAIL of exp
  | ISNIL of exp
  | IF of exp * exp * exp
  | LET of var * exp * exp
  | LETREC of var * var * exp * exp
  | LETMREC of (var * var * exp) * (var * var * exp) * exp
  | PROC of var * exp
  | CALL of exp * exp
  | PRINT of exp
  | SEQ of exp * exp
and var = string
```

Types for the language are defined as follows:

```
type typ =
    TyUnit
  | TyInt
  | TyBool
  | TyFun of typ * typ
  | TyList of typ
  | TyVar of tyvar
and tyvar = string
```

Implement a sound type checker, `typeof`, for the language (the notion of soundness is defined with respect to the dynamic semantics of the language defined in HW3):

```
typeof : exp -> typ
```

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`.

Examples:

- The program

  ```
  PROC ("f",
  ```

```
PROC ("x", SUB (CALL (VAR "f", CONST 3),
                CALL (VAR "f", VAR "x"))))
```

is well-typed and has type `TyFun (TyFun (TyInt, TyInt), TyFun (TyInt, TyInt))`.

- The program

```
PROC ("f", CALL (VAR "f", CONST 11))
```

is well-typed and has type `TyFun (TyFun (TyInt, TyVar "t"), TyVar "t")`, where `t` can be any type variable.

- The program

```
LET ("x", CONST 1,
  IF (VAR "x", SUB (VAR "x", CONST 1), CONST 0))
```

is ill-typed, so `typeof` should raise an exception `TypeError`.

- The program

```
LETMREC
 (("even", "x",
   IF (EQUAL (VAR "x", CONST 0), TRUE,
    CALL (VAR "odd", SUB (VAR "x", CONST 1)))),
 ("odd", "x",
  IF (EQUAL (VAR "x", CONST 0), FALSE,
   CALL (VAR "even", SUB (VAR "x", CONST 1)))),
 CALL (VAR "odd", CONST 13))
```

is well-typed and has type `TyBool`.

- The program

```
LETREC ("reverse", "l",
 IF (ISNIL (VAR "l"), NIL,
  APPEND (CALL (VAR "reverse", TAIL (VAR "l")), CONS (HEAD (VAR "l"), NIL))),
 CALL (VAR "reverse", CONS (CONST 1, CONS (CONST 2, CONS (CONST 3, NIL)))))
```

is well-typed and has type `TyList TyInt`.

- The program

```
LETREC ("reverse", "l",
 IF (ISNIL (VAR "l"), NIL,
  APPEND (CALL (VAR "reverse", TAIL (VAR "l")), CONS (HEAD (VAR "l"), NIL))),
 CALL (VAR "reverse",
  CONS (CONS (CONST 1, NIL),
   CONS (CONS (CONST 2, NIL), CONS (CONS (CONST 3, NIL), NIL)))))
```

is well-typed and has type `TyList (TyList TyInt)`.

- The program

```
LETREC ("factorial", "x",
 IF (EQUAL (VAR "x", CONST 0), CONST 1,
  MUL (CALL (VAR "factorial", SUB (VAR "x", CONST 1)), VAR "x")),
 LETREC ("loop", "n",
  IF (EQUAL (VAR "n", CONST 0), UNIT,
   SEQ (PRINT (CALL (VAR "factorial", VAR "n")),
    CALL (VAR "loop", SUB (VAR "n", CONST 1)))),
  CALL (VAR "loop", CONST 10)))
```

  is well-typed and has type `TyUnit`.

- Equality should support integers and booleans. For example, both `EQUAL (TRUE, FALSE)` and `EQUAL (CONST 1, CONST 2)` are well-typed. But `EQUAL (CONST 1, TRUE)` or `EQUAL (CONST 1, PROC ("x", CONST 1))` are ill-typed.

Unfortunately, our language now rejects the following programs, which worked well in HW3, due to the incompleteness of the type system.

- Polymorphic functions are not supported. The following program has a well-defined semantics but is rejected by the type system:

```
LET ("f", PROC("x", VAR "x"),
  IF(CALL (VAR "f", TRUE), CALL (VAR "f", CONST 1), CALL (VAR "f", CONST 2)))
```

- Recursion is not a syntactic sugar any more, as our type system rejects programs that use the fixed point combinator. For example, the following program is ill-typed according to our type system.

```
LET ("fix",
 PROC ("f",
  CALL
   (PROC ("x",
     CALL (VAR "f", PROC ("y", CALL (CALL (VAR "x", VAR "x"), VAR "y")))),
    PROC ("x",
     CALL (VAR "f", PROC ("y", CALL (CALL (VAR "x", VAR "x"), VAR "y")))))),
 LET ("f",
  CALL (VAR "fix",
   PROC ("f",
    PROC ("x",
     IF (EQUAL (VAR "x", CONST 0), CONST 1,
      MUL (CALL (VAR "f", SUB (VAR "x", CONST 1)), VAR "x"))))),
  CALL (VAR "f", CONST 10)))
```

- List elements should be all the same type. For example, the untyped interpreter in HW3 evaluates expression `CONS (CONST 1, CONS (CONST 2, CONS (TRUE, NIL)))` to the following list value

  ```
  List [Int 1; Int 2; Bool true]
  ```

  but such a polymorphic list is now rejected by our type system.