Homework: TypeLang

Learning Objectives:

1. Understanding, designing, and implementing typing rules
2. TypeLang programming

Instructions:

- Total points 60 pt
- Early deadline: April 15 at 11:59 PM; Regular deadline: April 17 at 11:59 PM
- Download hw8code.zip from Canvas. Interpreter for Typelang is significantly different compared to previous interpreters:
  - Env in Typelang is generic compared to previous interpreters.
  - Two new files Checker.java and Type.java have been added
  - Type.java defines all the valid types of Typelang.
  - Checker.java defines type checking semantics of all expressions.
  - Typelang.g has changed to add type information in expressions. Please review the changes in file to understand the syntax.
  - Finally Interpreter.java has been changed to add type checking phase before evaluation of Typelang programs.
- Set up the programming project following the instructions in the tutorial from hw2 (similar steps)
- Extend the Typelang interpreter for Q1 - Q5.
- How to submit:
  - Please submit your solutions in one zip file with all the source code files (just zip the complete project's folder).
  - Submit the zip file to Canvas under Assignments, Homework 8.

Questions:

1. (8 pt) [Implement type rules] Implement the type rules for the let expression based on the formal typing rules given in Figure 1.
2. (10 pt) [Implement type rules] Implement the type rules for memory related expressions based on the following descriptions:

(LetExp)

$$\frac{\begin{array}{c} tenv \vdash e_i : t_i, \forall i \in 0..n \\ tenv_n = (ExtendEnv\ var_n\ t_n\ tenv_{n-1})\ \ldots \\ tenv_0 = (ExtendEnv\ var_0\ t_0\ tenv) \\ tenv_n \vdash e_{body} : t \end{array}}{tenv \vdash (LetExp\ var_0\ \ldots\ var_n\ \ t_0\ \ldots\ t_n\ \ e_0\ \ldots\ e_n\ \ e_{body}) : t}$$

Figure 1: Q1: Let typing rule

(a) (5 points) RefExp: Let a ref expression be (ref: T e1), where e1 is an expression.

- if e1's type is ErrorT then (ref: T e1)'s type should be ErrorT
- if e1's type is T then (ref: T e1)'s type should be RefT with _nestType T. Note that _nestType is a field in RefT.
- otherwise, (ref: T e1)'s type is ErrorT with message "The Ref expression expect type " + T+ " found " + e1's type + " in " + expression.

Note that you have to add e1's type and expression in the error message. Some examples appear below.
$ (ref : bool 3)
Type error: The Ref expression expect type bool, found number in (ref 3)
$ (ref : num (list : num 1 2 3 4))
Type error: The Ref expression expect type number, found List<number> in (ref (list 1 2 3 4))

(b) (5 points) AssignExp: Let a set expression be (set! e1 e2), where e1 and e2 are expressions.

- if e1's type is ErrorT then (set! e1 e2)'s type should be ErrorT
- if e1's type is RefT and nestedType of e1 is T then
  - if e2's type is ErrorT then (set! e1 e2)'s type should be ErrorT
  - if e2's type is typeEqual To T then (set! e1 e2)'s type should be e2's type.
  - otherwise (set! e1 e2)'s type is ErrorT with message "The inner type of the reference type is " + nestedType T + " the rhs type is " + e2's type + " in " + expression
- otherwise (set! e1 e2)'s type is ErrorT with message "The lhs of the assignment expression expect a reference type found " + e1's type + " in "+ expression.

Note that you have to add e1's and e2's type and expression in the error message. Some examples appear below.
$ (set! (ref : num 0) #t)
Type error: The inner type of the reference type is number the rhs type is bool in (set! (ref 0) #t)
$ (set! (ref : bool #t) (list : num 1 2 3 4 5 6))
Type error: The inner type of the reference type is bool the rhs type is List<number> in (set! (ref #t) (list 1 2 3 4 5 6))

3. (15 pt) [Implement type rules] Implement the type rules for list expressions:

(a) (5 pt) CarExp: Let a car expression be (car e1), where e1 is an expression.

- if e1’s type is ErrorT then (car e1)’s type should be ErrorT
- if e1’s type is PairT then (car e1)’s type should be the type of the first element of the pair
- otherwise, (car e1)’s type is ErrorT with message ”The car expect an expression of type Pair, found”+ e1’s type+ ”in” + expression

Note that you have to add e1’s type and expression in the error message. See some examples below.

```
$ (car 2)
Type error: The car expect an expression of type Pair, found num in (car 2)
$ (car (car 2))
Type error: The car expect an expression of type Pair, found num in (car 2)
```

(b) (5 pt) CdrExp: Let a cdr expression be (cdr e1), where e1 is an expression.

- if e1’s type is ErrorT then (cdr e1)’s type should be ErrorT
- if e1’s type is PairT then (cdr e1)’s type should be the type of the second element of the pair
- otherwise, (cdr e1)’s type is ErrorT with message ”The cdr expect an expression of type Pair, found”+ e1’s type+ ”in” + expression

Note that you have to add e1’s type and expression in the error message. See some examples below.

```
$ (cdr 2)
Type error: The car expect an expression of type Pair, found number in (cdr 2.0)
$ (cdr (cdr 2))
Type error: The cdr expect an expression of type Pair, (cdr (cdr 2))
```

(c) (5 pt) ListExp: Let a list expression be (list : T e1 e2 e3 ... en), where T is type of list and e1, e2, e3 ... en are expressions:

- if type of any expression ei, where ei is an expression of element in list at position i, is ErrorT then type of (list : T e1 e2 e3 ... en) is ErrorT.
- if type of any expression ei, where ei is an expression of an element of list, is not T then type of (list : T e1 e2 e3 ... en) is ErrorT with message ”The ” + index + ” expression should have type ” + T + ” found ” + Type of ei + ” in ” + ”expression”. where index is the position of expression in list’s expression list.
- else type of (list : T e1 e2 e3 ... en) is ListT.

Note that you have to add ei’s type and expression in the error message.Index starts from 0. Some examples appear below.

```
$ (list : bool 1 2 3 4 5 6 7)
Type error: The 0 expression should have type bool, found number in (list 1 2 3 4 5 6 7 )
$ (list : num 1 2 3 4 5 #t 6 7 8)
Type error: The 5 expression should have type number, found bool in (list 1 2 3 4 5 #t 6 7 8)
```

4. (18 pt) [Design and implement type rules] Design and implement the type rules for greater than expressions:

   GreaterThanCompare: Let a GreaterThanCompare be (> e1 e2), where e1 and e2 are expressions.

   (a) (4 pt) Describe the type rules (see the example type rules provided in the above questions) to support the comparisons of two numbers

   (b) (4 pt) Describe the type rules to support the comparison of two lists

   (c) (10 pt) Implement the type checking rules for number and list comparisons.

5. (9 pt) [Eliminate Simple Divide-By-Zero Errors] For some expressions such as (/ x 0), where 0 appears as an immediate subexpression it is easy to check and eliminate divide-by-zero errors. Enhance the typechecking rule for the division expression above so that the type- system is able to detect and remove such errors, where 0 is an immediate subexpression of the division expression.

   - (4 pt) Describe the type rules to support the operation.
   - (5 pt) Implement the type checking rules.