Homework Solutions: TypeLang

for this also

Learning Objectives:

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

Instructions:

- Total points 67 pt
- Early deadline: April 12 (Wed) at 11:59 PM; Regular deadline: April 14 (Fri) at 11:59 PM (you can continue working on the homework till TA starts to grade the homework)
- 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–Q6.
- How to submit:
  - Write your solutions for Q6 and Q7 in a hw8.ss file and store it under your code directory.

```
🟥 hw8code
└── 🟦 typelang
    └── 🟩 src
        └── 🟫 main
            └── 🟧 java
                └── 🟥 examples
                    └── ⬛ hw8.ss
```

  - 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. (10 pt) [Implement Type Rules] Implement the type rules for `lambda` expression based on the typing rules listed below (see text-book Chapter 10 pages 202 for the explanations of the rules).

(LambdaExp)

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

Solution: Found in hw8code-sol.zip. You will need to modify Checker file:

```
public class Checker implements Visitor<Type,Env<Type>> {
  ...
  public Type visit(LambdaExp e, Env<Type> env) {
    List<String> names = e.formals();
    List<Type> types = e.formal_types();

    // FuncT ft = (FuncT) type;
    String message = "The number of formal parameters and the number of "
        + "arguments in the function type do not match in ";
    if (types.size() == names.size()) {
      Env<Type> new_env = env;
      int index = 0;
      for (Type argType : types) {
        new_env = new ExtendEnv<Type>(new_env, names.get(index),
            argType);
        index++;
      }

      Type bodyType = (Type) e.body().accept(this, new_env);

      if (bodyType instanceof ErrorT) {
        return bodyType;
      }

      // create a new function type with arguments, and the type of
      // the body as the return type. Notice, that the body type isn't
      // given in any type annotation, but being computed here.
      return new FuncT(types, bodyType);
    }

    return new ErrorT(message + ts.visit(e, null));
  }
  ...
}
```

2. (8 pt) Textbook 9.10.2 Questions 3 and 4.

Solution:

(1) `(lambda (x:num y:bool z:num) (+ x (+ y z)))`

- Since `y` has type `bool`, it violates the type-checking rule for addition. Hence, the expression is ill-typed. More specifically, it was expecting type `num` for the expression `(+ y z)` but found `bool` instead.

(2) `((lambda (x :num y :bool z :num) (+ x (+ y z)))1 2 #f)`

- Since `y` has type `bool`, it violates the type-checking rule for addition. Hence, the expression is ill-typed. More specifically, it was expecting type `num` for the expression `(+ y z)` but found `bool` instead.

3. (5 pt) [Implement Type Rules] Implement the type rules for memory related expressions based on the following descriptions:

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 *RefT* then `(ref: T e1)`'s type should `T`.
- Otherwise, `(ref: T e1)`'s type is *ErrorT* with message “The RefExp expect a reference type ” + T + “ found ” + e1's type + “ in ” + expression.

Note that you have to add `e1`'s type and expression in the error message. Examples:

$ (ref : num 45)
loc:0
// No explicit error cases
$ (ref : bool 45)
Type error: The Ref expression expects type bool found num in (ref 45.0)

Solution: Found in hw8code-sol.zip. You will need to modify Checker file:

```java
public class Checker implements Visitor<Type,Env<Type>> {
  ...
  public Type visit(RefExp e, Env<Type> env) {
    Exp value = e.value_exp();
    Type type = e.type();
    Type expType = (Type) value.accept(this, env);
    if (type instanceof ErrorT) {
      return type;
    }

    if (expType.typeEqual(type)) {
      return new RefT(type);
    }

    return new ErrorT("The Ref expression expects type " + type.tostring()
        + " found " + expType.tostring() + " in " + ts.visit(e, null));
  }
  ...
}
```

4. (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. Examples:

```
$ (car (list: num 2 3))
(2)
// No explicit error cases
$ (car 2)
Type error: The car expect an expression of type Pair; found number in (car 2.0)
$ (car (car 2))
Type error: The car expect an expression of type Pair, found number in (car 2.0)
```

(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 first 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. Examples:

```
$ (cdr (list: num 2 3))
(3)
$ (cdr 2)
Type error: The cdr 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, found number in (cdr 2.0)
```

(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 e_i, where e_i 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 e_i, where e_i 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 e_i + “ 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 e_i’s type and expression in the error message. Index starts from 0. Examples:

```
$ (list: List<num> (list: num 1 2 3) (list: num 4 5))
((1 2 3) (4 5))
$ (list : bool 1 2 3 4 5 6)
Type error: The 0 expression should have type bool; found num in (list 1.0 2.0 3.0 4.0 5.0 6.0)
$ (list : num 1 2 3 4 5 #t 6 7 8)
Type error: The 5 expression should have type num; found bool in (list 1.0 2.0 3.0 4.0 5.0 #t 6.0 7.0 8.0 )
$ (list: List<bool> (list: num 1 2 3) (list: num 4 5))
Type error: The 0 expression should have type List<bool>; found List<num> in (list (list 1.0 2.0 3.0 ) (list 4.0 5.0 ) )
```

Solution:

(a) Found in hw8code-sol.zip. You will need to modify Checker file:

```
public class Checker implements Visitor<Type,Env<Type>> {
  ...
  public Type visit(CarExp e, Env<Type> env) {
    Exp exp = e.arg();
    Type type = (Type)exp.accept(this, env);
    if (type instanceof ErrorT) { return type; }

    if (type instanceof PairT) {
      PairT pt = (PairT)type;
      return pt.fst();
    }

    return new ErrorT("The car expect an expression of type Pair; found "
        + type.tostring() + " in " + ts.visit(e, null));
  }
  ...
}
```

(b) Found in hw8code-sol.zip. You will need to modify Checker file:

```
public class Checker implements Visitor<Type,Env<Type>> {
  ...
  public Type visit(CdrExp e, Env<Type> env) {
    Exp exp = e.arg();
    Type type = (Type) exp.accept(this, env);
    if (type instanceof ErrorT) {
      return type;
    }

    if (type instanceof PairT) {
      PairT pt = (PairT) type;
      return pt.snd();
    }

    return new ErrorT("The cdr expect an expression of type Pair; found "
        + type.tostring() + " in " + ts.visit(e, null));
  }
  ...
}
```

(c) Found in hw8code-sol.zip. You will need to modify Checker file:

```
public class Checker implements Visitor<Type,Env<Type>> {
  ...
  public Type visit(ListExp e, Env<Type> env) {
    List<Exp> elems = e.elems();
    Type type = e.type();

    int index = 0;
    for (Exp elem : elems) {
      Type elemType = (Type)elem.accept(this, env);
      if (elemType instanceof ErrorT) { return elemType; }

      if (!assignable(type, elemType)) {
        return new ErrorT("The " + index +
            " expression should have type " + type.tostring() +
            "; found " + elemType.tostring() + " in " +
            ts.visit(e, null));
      }
      index++;
    }
    return new ListT(type);
  }
  ...
}
```

5. (18 pt) [Design And Implement Type Rules] Design and implement the type rules for comparison expressions:

   BinaryComparator: Let a `BinaryComparator` be `(binary operator 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.

   Solution:

   (a) Type rules for comparing two numbers:

   - If `e1`'s type is *ErrorT* then `(binary operator e1 e2)`'s type should be *ErrorT*.
   - If `e2`'s type is *ErrorT* then `(binary operator e1 e2)`'s type should be *ErrorT*.
   - If `e1`'s type is not *NumT* then `(binary operator e1 e2)`'s type should be *ErrorT* with message "The first argument of a binary expression should be num Type; found " + `e1`'s type + " in " + expression.
   - If `e2`'s type is not *NumT* then `(binary operator e1 e2)`'s type should be *ErrorT* with message "The second argument of a binary expression should be num Type; found " + `e2`'s type + " in " + expression.
   - Otherwise, `(binary operator e1 e2)`'s type should be *BoolT*.

   (b) Type rules for comparing two lists:

   - If `e1`'s type is *ErrorT* then `(binary operator e1 e2)`'s type should be *ErrorT*.
   - If `e2`'s type is *ErrorT* then `(binary operator e1 e2)`'s type should be *ErrorT*.
   - If `e1`'s type is *ListT* and `e2`'s type is ListT then `(binary operator e1 e2)`'s type should be *ErrorT* with message "Arguments of the binary expression are of List Type; found " + `e1`'s type + " and " + `e2`'s type " in " + expression.
   - Otherwise, `(binary operator e1 e2)`'s type should be *BoolT*.

   (c) Found in hw8code-sol.zip. You will need to modify Checker file:

```
public class Checker implements Visitor<Type,Env<Type>> {
  ...
  private Type visitBinaryComparator(BinaryComparator e, Env<Type> env,
      String printNode) {
    Exp first_exp = e.first_exp();
    Exp second_exp = e.second_exp();

    Type first_type = (Type) first_exp.accept(this, env);
    if (first_type instanceof ErrorT) {
      return first_type;
    }

    Type second_type = (Type) second_exp.accept(this, env);
    if (second_type instanceof ErrorT) {
      return second_type;
    }

    if ((first_type instanceof ListT) & (second_type instanceof ListT)) { //(> ( list: num 1 2 ) (list: num 3 4 4))
      return new ErrorT("The arguments of the binary expression "
              + "are of List Type, found " + first_type.tostring() + " and " + second_type.tostring() +
              " in " + printNode);
    }

    if (!(first_type instanceof NumT)) {
      return new ErrorT("The first argument of a binary expression "
          + "should be num Type, found " + first_type.tostring()
          + " in " + printNode);
    }

    if (!(second_type instanceof NumT)) {
      return new ErrorT("The second argument of a binary expression "
          + "should be num Type, found " + second_type.tostring()
          + " in " + printNode);
    }

    return BoolT.getInstance();
  }
  ...
}
```

6. (5 pt) [TypeLang Programming] In HW5, you have written a function `compression` that takes a list and remove consecutive repetitive letters or numbers. In this problem, Write a TypeLang program to compute the number compression over a list. You will identify and remove any consecutive repetitive numbers in a list. See the example interaction below:

   $ (compression (list:num 1 0 0 0 0 0 1))
   (1 0 1)
   $ (compression (list:num 1 0 0 1 0 0))
   (1 0 1 0)
   $ (compression (list:num 1 0 0 1 0 #t))
   Type error: The expected type of the 5 argument is number found bool in (compression 1.0 0.0 0.0 1.0 0.0 #t)

   Solution: Found in "hw8.ss" in hw8code-sol.zip.

```
(define consecutive: (num List<num> -> List<num>)
  (lambda (last: num lst: List<num>)
    (if (null? lst)
      lst
      (if (= last (car lst))
        (consecutive last (cdr lst))
        (cons (car lst) (consecutive (car lst) (cdr lst)))
        )
      )
    )
  )
(define compression: (List<num> -> List<num>)
  (lambda (lst: List<num>)
    (if (null? lst)
      lst
      (cons (car lst) (consecutive (car lst) (cdr lst)))
    )
  )
)
```

7. (6 pt) [TypeLang Programming] For all the above typing rules (total 6 of them) you implemented, write a TypeLang program for each type rule to test and demonstrate your type checking implementation. (You can use TypeLang.g in hw8code.zip as a reference for the syntax of TypeLang). For each expression, put in comments which type rules the expression is exercising. Example:

   $(deref (ref: bool #t)) // Test correct types for ref expressions
   $(deref (ref: bool 45)) // Test incorrect types for add expression
   @Type error: The Ref expression expect type bool found number in (ref 45.0)

   Solution: Found in hw8.ss in hw8code-sol.zip.

```
// lambda (1 pt)
$ ((lambda (x: num y: num) (+ x y)) 1 2) // Test correct types for lambda expression
$ ((lambda (x: num y: num z: num) (+ x y)) 1 2) // Test incorrect types for lambda expression
@Type error: The number of arguments expected is 3 found 2 in ((lambda ( x y z ) (+ x y )) 1.0 2.0 )
// ref (1 pt)
$ (ref: bool #t) // Test correct type for ref expression
$ (ref: bool 45) // Test incorrect type for ref expression
@Type error: The Ref expression expect type bool found number in (ref 45.0)
// car (1 pt)
$ (car (list:bool #t #f)) // Test correct for car
$ (car (list:num #t #f)) // Test incorrect for car
@Type error: The 0 expression should have type number; found bool in (list #t #f )
// cdr (1 pt)
$ (cdr (list:bool #t #f)) // Test correct for cdr
$ (cdr (list:num #t #f)) // Test incorrect for cdr
@Type error: The 0 expression should have type number; found bool in (list #t #f )
// list (1 pt)
$ (list: num 45 45 56 56 67) // Test correct type for list
$ (list: num 45 45 56 #t) // Test incorrect for list
@Type error: The 3 expression should have type number; found bool in (list 45.0 45.0 56.0 #t )
// binarycomparator (1 pt)
```

```
$ (let ((x: num 1)) (if (< x 4) x 6)) // Test correct for binary comparison
$ (let ((x: num 0)) (if (< x #t) x 6)) // Test incorrect for binary comparison
@Type error: The second argument of a binary expression should be num Type, found bool in (< x #t)
$ (> (list: num 1) (list: bool #t)) // Test incorrect for binary comparison
@Type error: The arguments of the binary expression are of List Type, found List<number> and List<bool> in (> (list 1.0 ) (list #t ))
```