Prof. Jingke Li (FAB120-06, lij@pdx.edu), Class: TR 12:00-13:15 @ KMC 185; Office Hour: TR 11:00-11:55.

Assignment 9: Type Checking

(Due Thursday 11/29/18)

This last assignment deals with type checking.

1. EL4

EL4 is a simple *statically typed* functional language, with the following grammar:

```
Expr -> Int
     |  Bool
     |  String
     |  (Op Expr Expr)
     |  (if Expr Expr Expr)
     |  (seq Expr Expr)
     |  (let String Type Expr Expr)
     |  (letRec String Type Expr Expr)
     |  (fun String Type Expr)
     |  (@ Expr Expr)
Op   -> + | - | * | / | && | || | <=
Type -> intT
     |  boolT
     |  (funT Type Type)
```

EL4 supports two primitive types, `intT` (integers) and `boolT` (Booleans), and one type constructor, `funT` (function types). Note that `funT` can be used to define *many* function types, such as `(funT intT intT)` and `(funT intT boolT)`, which represent function types, integer → integer and integer → Boolean, respectively.

Here is a Scala representation of EL4's types:

```
sealed abstract class Type
case object IntTy extends Type
case object BoolTy extends Type
case class FunTy(pt:Type,rt:Type) extends Type
```

All user-defined names, including variables, parameters, and function names, require a declaration with type before their use. Three of EL4's expressions include such a declaration, `let`, `letRec`, and `fun`. Compared to the EL3's version, they each is augmented with a `Type` component, right after the variable/parameter/function name. As an example, the following is the EL4's version of the factorial function (and its application):

```
(letRec fac (funT intT intT)
        (fun n intT
             (if (<= n 1)
                 1
                 (* n (@ fac (- n 1)))))
        (@ fac 5))
```

As shown, the two names, `fac` and `n`, both are declared with a type.

2. EL4's Typing Rules

EL4's informal typing rules are given below.

- An integer `i` and the two Boolean values, `true` and `false`, are always well-typed, and have the types of `intT` and `boolT`, respectively.
- A variable `x` is well-typed if it is defined in the current type environment (TE), and its type is given by its binding in the TE.
- An arithmetic expression `(op e1 e2)`, where `op` $\in$ {`+`, `-`, `*`, `/`}, are well-typed and has type `intT` if both `e1` and `e2` are well-typed and have the type `intT`.
- A Boolean expression `(op e1 e2)`, where `op` $\in$ {`&&`, `||`}, are well-typed and has type `boolT` if both `e1` and `e2` are well-typed and have the type `boolT`.
- An Leq expression `(<= e1 e2)` is well-typed and has type `boolT` if both `e1` and `e2` are well-typed and have the type `intT`.
- An If expression `(if c t f)` is well-typed and has type `ty` if `c`, `t`, and `f` are all well-typed, and `c` has type `boolT` and `t` and `f` both have type `ty`.
- A Seq expression `(seq e1 e2)` is well-typed and has type `t2` if both `e1` and `e2` are well-typed and `e2` has type `t2`. (`e1`'s type is ignored.)
- A Fun expression `(fun x tx e)` is well-typed and has type `(funT tx ty)` if `e` is well-typed in the augmented environment TE `+ (x->tx)` and has type `ty`.
- An Application expression `(@ f b)` is well-typed and has a `ty` if `f` is well-typed and has type `(funT tx ty)`, and `b` is well-typed and has type `tx`.
- A Let expression `(let x tx e1 e2)` is well-typed and has type `t2` if `e1` is well-typed in the current TE and has type `tx`, `e2` is well-typed in the augmented environment TE `+ (x->tx)` and has type `t2`.
- A LetRec expression `(letRec x tx e1 e2)` is well-typed and has type `t2` if `tx` is a `funT` type, both `e1` and `e2` are well-typed in the augmented environment TE `+ (x->tx)`, `e1` has type `tx`, and `e2` has type `t2`.

Your first task is to turn the above informal typing rules into formal typing rules in the form of judgments and inference rules. Put your answers in a text file `typing-rules.txt` or a PDF file `typing-rules.pdf`.

3. EL4 Typechecker

The file `Check.scala` contains an incomplete EL4 typechecker. The program structure is similar to that of an interpreter, but the coding is simpler, since a typechecker only deals with types, not values.

Read and understand the given code in the file. Your second task is to complete the typechecker program. The completed typechecker should catch *all* static type errors, and for each one print out an informative error message. A `TestCheck.scala` program is provided for testing your program.

At several places you'll need to compare two types for equality. EL4 uses the structural equivalence model. Note that Scala's `==` operator (unlike Java's) performs structural comparison on its class objects. For instance, `FunTy(IntTy,BoolTy) == FunTy(IntTy,BoolTy)` will turn true, even though the two sides represent two separate objects. So you don't need to implement your own comparison routine.

4. Submission

Zip your `typing-rules.[txt|pdf]` and `Check.scala` into a single ZIP file, and submit it through the "Assignment 9" submission folder on the D2L class website (under the "Activities/Assignments" tab). Keep your original files untouched in case there is a need to show their timestamp.