Assignment #2

CSE341: Principles of Programming Languages
Hyungon Moon

Out: Sep 29, 2020 (Tue)
Due: Oct 14, 2020 (Wed), 23:59 (KST)

What to submit

Submit your `Hw2.scala` file through the Blackboard.

> **Info:** The directory sturucture of the handout is as follows.
>
> ```
> sbt/                      - contains the sbt program that you need to test your program.
> src/                      - where all your scala source files leave.
>     main/scala/
>         Hw2.scala         - >>>> what you need to edit and submit. <<<<<
>         Parser.scala      - The parser driver for the languages you will interpret.
>     main/antlr4/          - where inputs to the parser generater lives. You can ignore this.
>     test/scala/
>         Hw2Test.scala     - The tests that I wrote for you.
>                            You can edit this to further test your program.
> ```

Rules

- You must not use the `var`, `for`, or `while` keyword.
- You must not include any additional packages or libraries besides the ones that you already have.

Scala environment

Please refer to the instruction for the first assignment to set up the Scala environment.

Problems

Problem 1 (30 points)

Implement an interpreter that evaluates an expression into an integer value.

Syntax

$$
\begin{aligned}
E \rightarrow\ & n \ \in\ \mathbb{Z} \\
|\ & x \ \in\ \{'x'\} \\
|\ & E + E \\
|\ & E - E \\
|\ & E\ *\ E \\
|\ & \texttt{sigma}\ E\ E\ E \\
|\ & \texttt{pow}\ E\ E
\end{aligned} \tag{1}
$$

In scala,

```
sealed trait IntExpr
case class IntConst(n: Int) extends IntExpr
case object IntVar extends IntExpr
case class IntAdd(l: IntExpr, r: IntExpr) extends IntExpr
case class IntSub(l: IntExpr, r: IntExpr) extends IntExpr
case class IntMul(l: IntExpr, r: IntExpr) extends IntExpr
case class IntSigma(f: IntExpr, t: IntExpr, b: IntExpr) extends IntExpr
case class IntPow(b: IntExpr, e: IntExpr) extends IntExpr
```

Note that the variable an expression could have only one variable, `x`, to represent the index variable in `sigma`.

—

$$\texttt{sigma 1 10 x} \tag{2}$$

stands for

—

$$\sum_{x=1}^{10} x \tag{3}$$

Follow the intuitive semantics for the addition, substraction, multiplication. For `sigma` and `pow` you can also follow the intuitive semantics, as follows. Note that `pow` is not defined if the exponent if evaluated to a negative integer.

Semantics

$$\begin{gathered}
\frac{\begin{array}{ll}\rho \vdash E_1 \Rightarrow v_1 & \rho \vdash E_2 \Rightarrow v_2 \\ [x \mapsto v_1]\rho \vdash E_3 \Rightarrow v_3 & \rho \vdash (\texttt{sigma}\ (E_1 + 1)\ E_2\ E_3) + v_3 \Rightarrow v_4\end{array}}{\rho \vdash \texttt{sigma}\ E_1\ E_2\ E_3 \Rightarrow v_4}\ v_1 \leq v_2 \\[2em]
\frac{\rho \vdash E_1 \rightarrow v_1 \quad \rho \vdash E_2 \rightarrow v_2}{\rho \vdash \texttt{sigma}\ E_1\ E_2\ E_3 \Rightarrow 0}\ v_1 > v_2 \\[2em]
\frac{\rho \vdash E_1 \Rightarrow v_1 \quad \rho \vdash E_2 \Rightarrow v_2 \quad \rho \vdash (\texttt{pow}\ E_1\ (E_2 - 1)) * E_1 \Rightarrow v_3}{\rho \vdash \texttt{pow}\ E_1\ E_2 \Rightarrow v_3}\ v_2 > 0 \\[2em]
\frac{\rho \vdash E_2 \Rightarrow v_2}{\rho \vdash \texttt{pow}\ E_1\ E_2 \Rightarrow 1}\ v_2 = 0
\end{gathered} \tag{4}$$

In the skeletone, you can find the `IntInterpreter` object whose `apply` method looks like:

```
def apply(s: String): Int
```

and calls the parser and the interpreter for you. You job is to fill out the body of this method.

```
def evalInt(expr: IntExpr, env: Option[Int]): Int
```

Note that not all valid programs under presented syntax will have semantics with respect to an empty environment. Please throw an exception if an interpreter is given such a program.

Problem 2 (40 points)

Implement an interpreter for our LETREC language. Follow the syntax and semantics given in the lecture slide.

The grammar in scala looks like:

```scala
sealed trait Program
sealed trait Expr extends Program
case class Const(n: Int) extends Expr
case class Var(s: String) extends Expr
case class Add(l: Expr, r: Expr) extends Expr
case class Sub(l: Expr, r: Expr) extends Expr
case class Iszero(c: Expr) extends Expr
case class Ite(c: Expr, t: Expr, f: Expr) extends Expr
case class Let(name: Var, value: Expr, body: Expr) extends Expr
case class Paren(expr: Expr) extends Expr
case class Proc(v: Var, expr: Expr) extends Expr
case class PCall(ftn: Expr, arg: Expr) extends Expr
case class LetRec(fname: Var, aname: Var, fbody: Expr, ibody: Expr)
extends Expr
```

The set of values can be defined like:

```scala
sealed trait Val
case class IntVal(n: Int) extends Val
case class BoolVal(b: Boolean) extends Val
case class ProcVal(v: Var, expr: Expr, env: Env) extends Val
case class RecProcVal(fv: Var, av: Var, body: Expr, expr: Expr, env: Env)
extends Val
```

Now you can fill out this method in `LetRecInterpreter` object.

```scala
def eval(env: Env, expr: Expr): Val
```

Note that not all valid programs under presented syntax will have semantics with respect to an empty environment. Please throw an exception if an interpreter is given such a program.

Problem 3 (30 points)

Implement a function that build a program into a string. Use the same `case class` definitions and implement the `apply` method of the object `LetRecToString`.

Note that not all valid programs under presented syntax will have semantics with respect to an empty environment. Please throw an exception if an interpreter is given such a program.

```scala
def apply(expr: Expr): String
```