Assignment #3

CSE271: Principles of Programming Languages
Hyungon Moon

Out: Oct 26, 2021 (Thu)
Due: Nov 9, 2021 (Tue), 23:59 (KST)

What to submit

Submit your `Hw3.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 live.
    main/scala/
        Hw3.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/
        Hw3Test.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 (100 points)

Implement an interpreter that evaluates a language that looks like `Scala`, but essentially similar to the language that we defined in the class.

Syntax

$$
\begin{aligned}
P &\to E \\
E &\to n \mid x \\
&\mid\ E + E \mid E - E \mid E * E \mid E \ / \ E \\
&\mid\ E > E \ \mid E\!>\!=\!E \\
&\mid\ \text{iszero } E \mid \text{if } E \text{ then } E \text{ else } E \\
&\mid\ \{ \text{ val } x = E \ ; E \ \} \\
&\mid\ \{ \text{ var } x = E \ ; E \ \} \\
&\mid\ (x) => E \\
&\mid\ \{\text{def } f(x) = E; E\} \\
&\mid\ x := E \\
&\mid\ (E) \mid \{E\} \\
&\mid\ E; E \\
&\mid\ E \ \ E \\
&\mid\ E \ \text{ mod } \ E \\
&\mid\ \text{cons}(E \ \ E)
\end{aligned}
$$

In scala,

```
sealed trait Program
sealed trait Expr extends Program
case class ConstI(n: Int) extends Expr
case class ConstB(b: Boolean) extends Expr
case class ConstIL(n: List[IntVal]) 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 Mul(l: Expr, r: Expr) extends Expr
case class Div(l: Expr, r: Expr) extends Expr
case class Rem(l: Expr, r: Expr) extends Expr
case class Cons(l: Expr, r: Expr) extends Expr
case class GTExpr(l: Expr, r: Expr) extends Expr
case class GEQExpr(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 ValExpr(name: Var, value: Expr, body: Expr) extends Expr
case class VarExpr(name: Var, value: Expr, body: Expr) extends Expr
case class Proc(v: Var, expr: Expr) extends Expr
case class DefExpr(fname: Var, aname: Var, fbody: Expr, ibody: Expr) extends Expr
case class Asn(v: Var, e: Expr) extends Expr
case class Paren(expr: Expr) extends Expr
case class Block(f: Expr, s: Expr) extends Expr
case class PCall(ftn: Expr, arg: Expr) extends Expr
```

The Domain on which the semantics is defined as follows

Domain

$$
\begin{aligned}
Val =&\ \mathbb{Z} + Bool + Procedure + RecProcedure + Loc + List \\
List =&\ \{\ (e_1, ..., e_n) \mid e \in E\ \} \\
Procedure =&\ Var \times E \times Env \\
RecProcedure =&\ Var \times Var \times E \times Env \\
\rho \in Env =&\ Var \rightarrow Val \\
\sigma \in Mem =&\ Loc \rightarrow Val \\
Loc =&\ \mathbb{N}
\end{aligned}
$$

In Scala,

```
sealed trait Val
case class IntVal(n: Int) extends Val
case class IntListVal(n: List[IntVal]) 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, env: Env) extends Val
case class LocVal(l: Loc) extends Val

type Env = HashMap[Var,Val]
type Loc = Int
case class Mem(m: HashMap[Loc,Val], top: Loc)
```

The semantics rules of the language is

Semantics

$$\frac{}{\rho, \sigma \vdash n \Rightarrow n, \sigma} \quad \frac{}{\rho, \sigma \vdash x \Rightarrow \rho(x), \sigma}\ \rho(x) \notin Dom(\sigma) \quad \frac{}{\rho, \sigma \vdash x \Rightarrow \sigma(\rho(x)), \sigma}\ \rho(x) \in Dom(\sigma)$$

$$\frac{\rho, \sigma_0 \vdash E_1 \Rightarrow n_1, \sigma_1 \quad \rho, \sigma_1 \vdash E_2 \Rightarrow n_2, \sigma_2}{\rho, \sigma_0 \vdash E_1 + E_2 \Rightarrow n_1 + n_2, \sigma_2} \quad \frac{\rho, \sigma_0 \vdash E_1 \Rightarrow n_1, \sigma_1 \quad \rho, \sigma_1 \vdash E_2 \Rightarrow n_2, \sigma_2}{\rho, \sigma_0 \vdash E_1 - E_2 \Rightarrow n_1 - n_2, \sigma_2}$$

$$\frac{\rho, \sigma_0 \vdash E_1 \Rightarrow n_1, \sigma_1 \quad \rho, \sigma_1 \vdash E_2 \Rightarrow n_2, \sigma_2}{\rho, \sigma_0 \vdash E_1 * E_2 \Rightarrow n_1 * n_2, \sigma_2} \quad \frac{\rho, \sigma_0 \vdash E_1 \Rightarrow n_1, \sigma_1 \quad \rho, \sigma_1 \vdash E_2 \Rightarrow n_2, \sigma_2}{\rho, \sigma_0 \vdash E_1 / E_2 \Rightarrow n_1 / n_2, \sigma_2}\ n_2 \neq 0$$

$$\frac{\rho, \sigma_0 \vdash E_1 \Rightarrow n_1, \sigma_1 \quad \rho, \sigma_1 \vdash E_2 \Rightarrow n_2, \sigma_2}{\rho, \sigma_0 \vdash E_1 \mod E_2 \Rightarrow n_1 \mod n_2, \sigma_2}\ n_2 \neq 0 \quad \frac{\rho, \sigma_0 \vdash E_1 \Rightarrow n_1, \sigma_1 \quad \rho, \sigma_1 \vdash E_2 \Rightarrow n_2, \sigma_2}{\rho, \sigma_0 \vdash Cons(E_1, E_2) \Rightarrow [n_1 :: n_2], \sigma_2}$$

$$\frac{\rho, \sigma_0 \vdash E_1 \Rightarrow n_1, \sigma_1 \quad \rho, \sigma_1 \vdash E_2 \Rightarrow n_2, \sigma_2}{\rho, \sigma_0 \vdash E_1 > E_2 \Rightarrow n_1 > n_2, \sigma_2} \quad \frac{\rho, \sigma_0 \vdash E_1 \Rightarrow n_1, \sigma_1 \quad \rho, \sigma_1 \vdash E_2 \Rightarrow n_2, \sigma_2}{\rho, \sigma_0 \vdash E_1 \geq E_2 \Rightarrow n_1 \geq n_2, \sigma_2}$$

$$\frac{\rho, \sigma_0 \vdash E \Rightarrow 0, \sigma_1}{\rho, \sigma_0 \vdash \texttt{iszero}\ E \Rightarrow true, \sigma_1} \quad \frac{\rho, \sigma_0 \vdash E \Rightarrow v, \sigma_1}{\rho, \sigma_0 \vdash \texttt{iszero}\ E \Rightarrow false, \sigma_1}\ v \neq 0$$

$$\frac{\rho, \sigma_0 \vdash E \Rightarrow false, \sigma_1 \quad \rho, \sigma_1 \vdash E_3 \rightarrow v, \sigma_2}{\rho, \sigma_0 \vdash \texttt{if}\ E_1\ \texttt{then}\ E_2\ \texttt{else}\ E_3 \Rightarrow v, \sigma_2}$$

$$\frac{\rho, \sigma_0 \vdash E \Rightarrow true, \sigma_1 \quad \rho, \sigma_1 \vdash E_2 \rightarrow v, \sigma_2}{\rho, \sigma_0 \vdash \texttt{if}\ E_1\ \texttt{then}\ E_2\ \texttt{else}\ E_3 \Rightarrow v, \sigma_2}$$

$$\frac{\rho, \sigma_0 \vdash E_1 \Rightarrow v_1, \sigma_1 \quad [x \mapsto v_1]\rho, \sigma_1 \vdash E_2 \rightarrow v, \sigma_2}{\rho, \sigma_0 \vdash \{\ \texttt{val}\ x = E_1\ ;\ E_2\} \Rightarrow v, \sigma_2}$$

$$\frac{\rho, \sigma_0 \vdash E_1 \Rightarrow v_1, \sigma_1 \quad [x \mapsto l]\rho, [l \mapsto v_1]\sigma_1 \vdash E_2 \rightarrow v, \sigma_2}{\rho, \sigma_0 \vdash \{\texttt{var}\ x = E_1\ ;\ E_2\} \Rightarrow v, \sigma_2}\ l \notin \text{Dom}(\sigma_1)$$

$$\frac{}{\rho, \sigma \vdash\ \text{(x) => } E \Rightarrow (x, E, \rho), \sigma}$$

$$\frac{[f \mapsto (f, x, E_1, \rho)\rho, \sigma \vdash E_2 \Rightarrow v, \sigma_1]}{\rho, \sigma \vdash\ \{\text{def}\ f(x) = E_1; E_2\} \Rightarrow v, \sigma_1}$$

$$\frac{\rho, \sigma \vdash E_1 \Rightarrow (f, x, E, \rho'), \sigma_1 \quad \rho, \sigma_1 \vdash E_2 \Rightarrow v_2, \sigma_2 \quad [x \mapsto v, f \mapsto (f, x, E, \rho')]\rho', \sigma_2 \vdash E \Rightarrow v_3, \sigma_3}{\rho, \sigma \vdash E_1\ \ E_2 \Rightarrow v_3, \sigma_3}$$

$$\frac{\rho, \sigma \vdash E_1 \Rightarrow (x, E, \rho'), \sigma_1 \quad \rho, \sigma_1 \vdash E_2 \Rightarrow v_2, \sigma_2 \quad [x \mapsto v]\rho', \sigma_2 \vdash E \Rightarrow v_3, \sigma_3}{\rho, \sigma \vdash E_1\ \ E_2 \Rightarrow v_3, \sigma_3}$$

$$\frac{\rho, \sigma \vdash E \Rightarrow v, \sigma_1}{\rho, \sigma \vdash x := E \Rightarrow v, [\rho(x) \mapsto v]\sigma_1}$$

$$\frac{\rho, \sigma \vdash E_1 \Rightarrow v_1, \sigma_1 \quad \rho, \sigma_1 \vdash E_2 \Rightarrow v_2, \sigma_2}{\rho, \sigma \vdash E_1; E_2 \Rightarrow v_2, \sigma_2}$$

$$\frac{\rho, \sigma \vdash E \Rightarrow v, \sigma_1}{\rho, \sigma \vdash \{E\} \Rightarrow v, \sigma_1} \quad \frac{\rho, \sigma \vdash E \Rightarrow v, \sigma_1}{\rho, \sigma \vdash (E) \Rightarrow v, \sigma_1}$$

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

```scala
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.

```scala
def eval(env: Env, mem: Mem, expr: Expr): Result =
    Result(BoolVal(false), Mem(new HashMap[Loc, Val],0))
```

As noted in class, a valid program that passes the parser may not have its semantics. If this is the case, this time, you have to throw a particular exception that is defined in the object as follows.

```
case class UndefinedSemantics(msg: String = "", cause: Throwable = None.orNull)
extends Exception("Undefined Semantics: " ++ msg, cause)
```

You can throw the exception as follows.

```
throw new UndefinedSemantics(s"message ${variable}")
```