Assignment #4

CSE271: Principles of Programming Languages
Hyungon Moon, Mijung Kim

Out: Nov 10, 2021 (Wed)
Due: Nov 23, 2021 (Tue), 23:59 (KST)

What to submit

Submit your `Hw4.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/
    Hw4.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/
    Hw4Test.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.

Late submission policy

You will get 70% and 50% of the grade if you turn it in within 24 and 48 hours, respectively, after the regular deadline. Please note that an assignment that is re-submitted after the regular deadline will be counted as late even if you have an earlier submission. After the late submission deadline has passed, you won't be able to submit your assignment and will get 0 points for it.

Problems

Problem 1 (60 points)

Implement an interpreter that evaluates a language that looks like `C`, that adopts implicit references and procedures with multiple arguments.

Syntax

$$
\begin{aligned}
P &\to E \\
E &\to \text{skip} \mid \text{true} \mid \text{false} \mid n \mid x \\
&\mid E + E \mid E - E \mid E * E \mid E \;/\; E \\
&\mid E == E \mid E <\!= E \mid \text{not } E \\
&\mid \text{if } E \text{ then } E \text{ else } E \\
&\mid \text{while } E\ E \quad \mid \text{let } x = E \text{ in } E \\
&\mid \text{proc } (x_1, x_2, \ldots x_n)\ E \\
&\mid E\ (E_1, E_2, \ldots, E_n) && \text{call-by-value} \\
&\mid E\ \langle y_1, y_2, \ldots, y_n \rangle && \text{call-by-reference} \\
&\mid \{\ \} \mid \{x_1 := E_1, \ldots, x_n := E_n\} \\
&\mid x := E \mid E.x \mid E.x := E \mid E; E \mid \text{begin } E \text{ end}
\end{aligned}
$$

In scala,

```
sealed trait Program
sealed trait Expr extends Program
case object Skip extends Expr
case object False extends Expr
case object True extends Expr
case class NotExpr(expr: Expr) extends Expr
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 Mul(l: Expr, r: Expr) extends Expr
case class Div(l: Expr, r: Expr) extends Expr
case class LTEExpr(l: Expr, r: Expr) extends Expr
case class EQExpr(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(i: Var, v: Expr, body: Expr) extends Expr
case class Proc(args: List[Var], expr: Expr) extends Expr
case class Asn(v: Var, e: Expr) extends Expr
case class BeginEnd(expr: Expr) extends Expr
case class FieldAccess(record: Expr, field: Var) extends Expr
case class FieldAssign(record: Expr, field: Var, new_val: Expr) extends Expr
case class Block(f: Expr, s: Expr) extends Expr
case class PCallV(ftn: Expr, arg: List[Expr]) extends Expr
case class PCallR(ftn: Expr, arg: List[Var]) extends Expr
case class WhileExpr(cond: Expr, body: Expr) extends Expr
sealed trait RecordLike extends Expr
case object EmptyRecordExpr extends RecordLike
case class RecordExpr(field: Var, initVal: Expr, next: RecordLike)
extends RecordLike
```

The Domain on which the semantics are defined.

Domain

$$
\begin{aligned}
Val =&\ \ \mathbb{Z} + Bool + \{\cdot\} + Procedure + Loc + Record \\
Procedure =&\ \ (Var \times Var \times \ldots)E \times Env \\
r \in Record =&\ \ Field \to Loc \\
\rho \in Env =&\ \ Var \to Loc \\
\sigma \in Mem =&\ \ Loc \to Val
\end{aligned}
$$

In Scala,

```
sealed trait Val
case object SkipVal extends Val
case class IntVal(n: Int) extends Val
case class BoolVal(b: Boolean) extends Val
case class ProcVal(args: List[Var], expr: Expr, env: Env) extends Val
case class LocVal(l: Int) extends Val
sealed trait RecordValLike extends Val
case object EmptyRecordVal extends RecordValLike
case class RecordVal(field: Var, loc: LocVal, next: RecordValLike)
extends RecordValLike

type Env = HashMap[Var,Val]

case class Mem(m: HashMap[Loc,Val], top: Loc)
```

The semantic rules of the language is as follows.

Constants and Variables

$$\frac{}{\rho, \sigma \vdash \text{skip} \Rightarrow \cdot, \sigma} \quad \frac{}{\rho, \sigma \vdash \text{true} \Rightarrow true, \sigma} \quad \frac{}{\rho, \sigma \vdash \text{false} \Rightarrow false, \sigma}$$

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

$$\frac{}{\rho, \sigma \vdash \text{ proc } (x_1, \cdots, x_n)\ E \Rightarrow ((x_1, ..., x_n), E, \rho), \sigma}$$

Unary and Binary Operations

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

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

$$\frac{\rho, \sigma_0 \vdash E_1 \Rightarrow n_1, \sigma_1 \quad \rho, \sigma_1 \vdash E_2 \Rightarrow n_2, \sigma_2}{\rho, \sigma_0 \vdash E_1 <= E_2 \Rightarrow true, \sigma_2}\ n_1 \leq n_2$$

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

$$\frac{\rho, \sigma_0 \vdash E_1 \Rightarrow n_1, \sigma_1 \quad \rho, \sigma_1 \vdash E_2 \Rightarrow n_2, \sigma_2}{\rho, \sigma_0 \vdash E_1 == E_2 \Rightarrow true, \sigma_2}\ v_1 = v_2 = n \vee v_1 = v_2 = b \vee v_1 = v_2 = \cdot$$

$$\frac{\rho, \sigma_0 \vdash E_1 \Rightarrow n_1, \sigma_1 \quad \rho, \sigma_1 \vdash E_2 \Rightarrow n_2, \sigma_2}{\rho, \sigma_0 \vdash E_1 == E_2 \Rightarrow false, \sigma_2}\ \text{otherwise}$$

$$\frac{\rho, \sigma_0 \vdash E \Rightarrow true, \sigma_1}{\rho, \sigma_0 \vdash \text{not } E \Rightarrow false, \sigma_1} \quad \frac{\rho, \sigma_0 \vdash E \Rightarrow false, \sigma_1}{\rho, \sigma_0 \vdash \text{not } E \Rightarrow true, \sigma_1}$$

Note that only integers can be compared with each other with $>$ and $>=$, and the equality operation evaluates to false under type mismatches also.

Flow Control

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

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

$$\frac{\rho, \sigma_0 \vdash E_1 \Rightarrow false, \sigma_1}{\rho, \sigma_0 \vdash \text{while } E_1\ E_2 \Rightarrow \cdot, \sigma_1}$$

$$\frac{\rho, \sigma \vdash E_1 \Rightarrow true, \sigma_0 \quad \rho, \sigma_0 \vdash E_2 \Rightarrow v_1, \sigma_1 \quad \rho, \sigma_1 \vdash \text{while } E_1\ E_2 \Rightarrow v_2, \sigma_2}{\rho, \sigma \vdash \text{while } E_1\ E_2 \Rightarrow \cdot, \sigma_2}$$

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

$$\frac{\rho, \sigma \vdash E \Rightarrow v, \sigma'}{\rho, \sigma \vdash \text{begin } E \text{ end} \Rightarrow v, \sigma'}$$

Records

$$\frac{}{\rho, \sigma \vdash \{\ \} \Rightarrow \cdot, \sigma}$$

$$\frac{\rho, \sigma_0 \vdash E_1 \Rightarrow, v_1, \sigma_1 \quad \cdots \quad \rho, \sigma_n \vdash E_n \Rightarrow, v_n, \sigma_n \quad l_1, \cdots, l_n \notin Dom(\sigma_n)}{\rho, \sigma_0 \vdash \{\ x_1 := E_1, \cdots, x_n := E_n\ \} \Rightarrow \{\ x_1 \mapsto l_1, \cdots, x_n \mapsto l_n\ \}, [l_1 \mapsto v_1, \cdots, l_n \mapsto v_n]\sigma_n}$$

$$\frac{\rho, \sigma_0 \vdash E \Rightarrow r, \sigma_1}{\rho, \sigma_0 \vdash E.x \Rightarrow \sigma_1(r(x)), \sigma_1}$$

$$\frac{\rho, \sigma_0 \vdash E_1 \Rightarrow r, \sigma_1 \quad \rho, \sigma_1 \vdash E_2 \Rightarrow v, \sigma_2}{\rho, \sigma_0 \vdash E_1.x := E_2 \Rightarrow v, [r(x) \mapsto v]\sigma_2}$$

Assignments

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

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

Procedure Calls

$$\frac{\begin{array}{c}\rho, \sigma \vdash E_0 \Rightarrow ((x_1, \ldots, x_n), E, \rho'), \sigma_0 \\ \rho, \sigma_0 \vdash E_1 \Rightarrow v_1, \sigma_1 \\ \cdots \\ \rho, \sigma_{n-1} \vdash E_n \Rightarrow v_n, \sigma_n \\ [x_1 \mapsto l_1, \ldots, x_n \mapsto l_n]\rho', [l_1 \mapsto v_1, \ldots, l_n \mapsto v_n]\sigma_n \vdash E \Rightarrow v, \sigma'\end{array}}{\rho, \sigma \vdash E_0\ (E_1, \ldots, E_n) \Rightarrow v, \sigma'}\ l_1, \ldots, l_n \notin Dom(\sigma_n)$$

$$\frac{\begin{array}{l}\rho, \sigma \vdash E_0 \Rightarrow ((x_1, \ldots, x_n), E, \rho'), \sigma_0 \\ [x_1 \mapsto \rho(y_1), \ldots, x_n \mapsto \rho(y_n)]\rho', \sigma_0 \vdash E \Rightarrow v, \sigma'\end{array}}{\rho, \sigma \vdash E_0\ \langle y_1, \ldots, y_n \rangle \Rightarrow v, \sigma'}$$

In the skeleton, you can find the `MiniCInterpreter` object whose `apply` method looks like:

```
def apply(program: String): (Val, Mem)
```

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

```
def eval(env: Env, mem: Mem, expr: Expr): Result = Result(SkipVal, mem)
```

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}")
```

Problem 2 (40 points)

In this language, we allocate new memory locations in let, call, and record expressions, but they never get freed, leading to memory exhaustion. Fill out another method `gc` in the Interpreter. Please note that your interpreter does not need to call this gc. It will be evaluated separately as being done tested with the embedded test.

```
def gc(env: Env, mem: Mem): Mem
```