5118014 Programming Language Theory

Ch 12. Mutable Variables

Shin Hong

Mutable Variables

- Mutable variables allow the values associated with names to change
- Example

```scala
def makeCounter(): () => Int = {
  var x = 0
  def counter(): Int = {
    x += 1
    x
  }
  counter
}
```

```scala
val counter1 = makeCounter()
val counter2 = makeCounter()

println(counter1())
println(counter2())
println(counter1())
println(counter2())
```

MFAE: Syntax

$$e ::= \cdots \mid x := e$$

- the left-hand-side of an assignment is restricted to a variable
- the sequencing expression can be treated as a syntactic sugar
  - transform $e_1; e_2$ into $(\lambda x. e_2)e_1$

Environment and Store

$$v ::= \ n \in \mathbb{Z} \mid \langle \lambda x.e, \sigma \rangle$$

$$M \in Sto = Addr \rightharpoonup V$$

$$\sigma \in Env = Id \rightharpoonup Addr$$

- Unlike BFAE, an address is not a value in MFAE
- An environment is a finite partial function from identifiers to addresses
    - addresses are not exposed to programmers as values

Semantics

$$\frac{x \in \mathit{Domain}(\sigma) \qquad \sigma(x) \in \mathit{Domain}(M)}{\sigma, M \vdash x \Rightarrow M(\sigma(x)), M}\ [\textsc{Id}]$$

$$\frac{\sigma, M \vdash e_1 \Rightarrow \langle \lambda x.e, \sigma' \rangle, M_1 \qquad \sigma, M_1 \vdash e_2 \Rightarrow v', M_2 \qquad a \notin \mathit{Domain}(M_2) \qquad \sigma'[x \mapsto a], M_2[a \mapsto v'] \vdash e \Rightarrow v, M_3}{\sigma, M \vdash e_1\ e_2 \Rightarrow v, M_3}\ [\textsc{App}]$$

$$\frac{x \in \mathit{Domain}(\sigma) \qquad \sigma, M \vdash e \Rightarrow v, M_1}{\sigma, M \vdash x{:=}e \Rightarrow v, M_1[\sigma(x) \mapsto v]}\ [\textsc{Set}]$$

Interpreter

```
sealed trait Expr
...
case class Set(x: String, e: Expr) extends Expr


type Addr = Int
type Sto = Map[Addr, Value]
type Env = Map[String, Addr]
```

```
def interp(e: Expr, env: Env, sto: Sto): (Value, Sto) =
  e match {
    ...
    case Id(x) => (sto(env(x)), sto)
    case App(f, a) =>
      val (CloV(x, b, fEnv), ls) = interp(f, env, sto)
      val (v, rs) = interp(a, env, ls)
      val addr = rs.keys.maxOption.getOrElse(0) + 1
      interp(b, fEnv + (x -> addr), rs + (addr -> v))
    case Set(x, e) =>
      val (v, s) = interp(e, env, sto)
      (v, s + (env(x) -> v))
  }
```

Call-By-Reference

- Call-by-value: the value of an argument is copied and saved at fresh addresses
- Call-by-reference: the reference of the argument is given to the parameter
  - a reference means a mapping between an identifier and an address

$$\frac{\sigma, M \vdash e \Rightarrow \langle \lambda x'.e', \sigma' \rangle, M_1 \qquad x \in Domain(\sigma) \qquad \sigma'[x' \mapsto \sigma(x)], M_1 \vdash e' \Rightarrow v, M_2}{\sigma, M \vdash e\ x \Rightarrow v, M_2} \text{ [App-Cbr]}$$

$$\frac{e_2 \notin Id \qquad \sigma, M_1 \vdash e_2 \Rightarrow v', M_2 \qquad \begin{array}{c}\sigma, M \vdash e_1 \Rightarrow \langle \lambda x.e, \sigma' \rangle, M_1 \\ a \notin Domain(M_2)\end{array} \qquad \sigma'[x \mapsto a], M_2[a \mapsto v'] \vdash e \Rightarrow v, M_3}{\sigma, M \vdash e_1\ e_2 \Rightarrow v, M_3} \text{ [App-Cbv]}$$

Exercise 12.4

- Extend MFAE with pointers.

$$e ::= \cdots \mid *\, e \mid \&x \mid *\, e{:=}e$$
$$v ::= \cdots \mid a$$

The semantics of some constructs are as follows:

- The value of $*e$ is the value in the store at the address denoted by the expression.
- The value of $\&x$ is the address denoted by the identifier in the environment.
- The evaluation of $*e_1{:=}e_2$ evaluates e_2 first, which is the value of the whole expression. Then, it evaluates e_1, and it maps the address denoted e_1 to the value of e_2.

Exercise 12.3

- Extend MFAE with mutable records

$$
\begin{array}{rcl}
e & ::= & \cdots \mid \{f{:}e, \cdots, f{:}e\} \mid e.f \\
v & ::= & \cdots \mid \{f{:}a, \cdots, f{:}a\}
\end{array}
$$

$$
\frac{\begin{array}{c}\sigma, M_0 \vdash e_1 \Rightarrow v_1, M_1' \quad \cdots \quad \sigma, M_{n-1} \vdash e_n \Rightarrow v_n, M_n' \\ a_1 \notin Domain(M_1') \quad \cdots \quad a_n \notin Domain(M_n') \qquad M_1 = M_1'[a_1 \mapsto v_1] \quad \cdots \quad M_n = M_n'[a_n \mapsto v_n]\end{array}}{\sigma, M_0 \vdash \{f_1{:}e_1, \cdots, f_n{:}e_n\} \Rightarrow \{f_1{:}a_1, \cdots, f_n{:}a_n\}, M_n}
$$

$$
\frac{\sigma, M \vdash e \Rightarrow \{\cdots, f{:}a, \cdots\}, M_1 \qquad a \in Domain(M_1)}{\sigma, M \vdash e.f \Rightarrow M_1(a), M_1}
$$

```
case class Get(r: Expr, f: String) extends Expr
case class RecV(fs: Map[String, Addr]) extends Value

def interp(e: Expr, env: Env, sto: Sto): (Value, Sto) = e match {
  ...
  case App(f, a) =>
    val (fv, fs) = interp(f, env, sto)
    fv match {
      case CloV(x, b, fenv) =>
        a match {
          case Get(r, f) => ???
          case _ =>
            val (av, as) = interp(a, env, fs)
            val addr = malloc(as)
            interp(b, fenv + (x -> addr), as + (addr -> av))
        }
      case _ => error()
    }
}
```