Homework 4
ENE4014 Programming Languages, Spring 2020
Woosuk Lee
due: 5/27(Wed), 24:00

Exercise 1 Consider the following programming language, called `miniC`, that features (recursive) procedures and implicit references.

Syntax The syntax is defined as follows:

$$
\begin{array}{rcll}
P & \rightarrow & E & \\
E & \rightarrow & \texttt{skip} \mid \texttt{true} \mid \texttt{false} & \\
& \mid & n & \\
& \mid & x & \\
& \mid & E + E \mid E - E \mid E * E \mid E / E & \\
& \mid & E \leq E \mid E = E \mid \texttt{not}\ E & \\
& \mid & \texttt{if}\ E\ \texttt{then}\ E\ \texttt{else}\ E & \\
& \mid & \texttt{while}\ E\ E & \\
& \mid & \texttt{let}\ x := E\ \texttt{in}\ E & \\
& \mid & \texttt{proc}\ (x_1, \cdots, x_n)\ E & \\
& \mid & E\ (E_1, E_2, \cdots, E_n) & \text{call-by-value} \\
& \mid & E\ \langle y_1, y_2, \cdots, y_n \rangle & \text{call-by-reference} \\
& \mid & x := E & \\
& \mid & \texttt{\{ \}}\ \mid \texttt{\{}\ x_1 := E_1, \cdots, x_n := E_n\ \texttt{\}} & \\
& \mid & E.x & \\
& \mid & E.x := E & \\
& \mid & E; E & \\
& \mid & \texttt{begin}\ E\ \texttt{end} &
\end{array}
$$

A program is an expression. Expressions include unit, assignments, sequences, conditional expressions (branch), while loops, read, write, let expressions, let expressions for procedure binding, procedure calls (by either call-by-value or call-by-reference), integers, boolean constants, records (i.e., structs), record lookup, record assignment, identifier, arithmetic expressions, and boolean expressions. Note that procedures may have multiple arguments. The language manipulates the following values:

Semantics The semantics is defined with the following domain:

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

A record (i.e., struct) is defined as a (finite) function from identifiers to memory addresses. A value is either an integer, boolean value, unit value $(\cdot)$, or a record. An environment maps identifiers to memory addresses or procedure values. A memory is a finite function from addresses to values.

Evaluation rules are as follows:

Constants and Variables

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

$$\frac{}{\rho, \sigma \vdash \texttt{true} \Rightarrow true, \sigma} \qquad \frac{}{\rho, \sigma \vdash \texttt{false} \Rightarrow false, \sigma}$$

$$\frac{}{\rho, \sigma \vdash n \Rightarrow n, \sigma} \qquad \frac{}{\rho, \sigma \vdash x \Rightarrow \sigma(\rho(x)), \sigma}$$

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

Unary and Binary Operations

$$\frac{\rho, \sigma_0 \vdash E_1 \Rightarrow n_1, \sigma_1 \qquad \rho, \sigma_1 \vdash E_2 \Rightarrow n_2, \sigma_2}{\rho, \sigma_0 \vdash E_1 \oplus E_2 \Rightarrow n_1 \oplus n_2, \sigma_2}\ \oplus \in \{+, -, *, /\}$$

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

$$\frac{\rho, \sigma_0 \vdash E_1 \Rightarrow n_1, \sigma_1 \qquad \rho, \sigma_1 \vdash E_2 \Rightarrow n_2, \sigma_2}{\rho, \sigma_0 \vdash E_1 \leq E_2 \Rightarrow false, \sigma_2}\ n_1 > n_2$$

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

$$\frac{\rho, \sigma_0 \vdash E_1 \Rightarrow v_1, \sigma_1 \qquad \rho, \sigma_1 \vdash E_2 \Rightarrow v_2, \sigma_2}{\rho, \sigma_0 \vdash E_1 = E_2 \Rightarrow false, \sigma_2}\ \text{otherwise}$$

$$\frac{\rho, \sigma_0 \vdash E \Rightarrow b, \sigma_1}{\rho, \sigma_0 \vdash \texttt{not}\ E \Rightarrow not\ b, \sigma_1}$$

Flow Controls

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

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

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

$$\frac{\rho, \sigma \vdash E_1 \Rightarrow \mathit{true}, \sigma_0 \qquad \rho, \sigma_0 \vdash E_2 \Rightarrow v_1, \sigma_1 \qquad \rho, \sigma_1 \vdash \texttt{while } E_1 \ E_2 \Rightarrow v_2, \sigma_2}{\rho, \sigma \vdash \texttt{while } E_1 \ E_2 \Rightarrow v_2, \sigma_2}$$

$$\frac{\rho, \sigma_0 \vdash E_1 \Rightarrow v_1, \sigma_1 \qquad \rho, \sigma_1 \vdash E_2 \Rightarrow v_2, \sigma_2}{\rho, \sigma_0 \vdash E_1; E_2 \Rightarrow v_2, \sigma_2}$$

Records

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

$$\frac{\rho, \sigma_0 \vdash E_1 \Rightarrow v_1, \sigma_1 \quad \rho, \sigma_1 \vdash E_2 \Rightarrow v_2, \sigma_2 \quad \cdots \quad \rho, \sigma_{n-1} \vdash E_n \Rightarrow v_n, \sigma_n \qquad l_1, \cdots, l_n \notin Dom(\sigma_n)}{\rho, \sigma_0 \vdash \texttt{\{ } x_1 := E_1, \cdots, x_n := E_n \texttt{ \}} \Rightarrow \{x_1 \mapsto l_1, \cdots, x_n \mapsto l_n\}, [l_1 \mapsto v_1, \cdots, l_n \mapsto v_n]\sigma_n}$$

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

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

Assignments

$$\frac{\rho, \sigma_0 \vdash E \Rightarrow v, \sigma_1}{\rho, \sigma_0 \vdash x := E \Rightarrow v, [\rho(x) \mapsto v]\sigma_1}$$

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

Function Calls

$$\frac{\begin{array}{c}\rho, \sigma \vdash E_0 \Rightarrow ((x_1, \cdots, x_n), E, \rho'), \sigma_0 \\ \rho, \sigma_0 \vdash E_1 \Rightarrow v_1, \sigma_1 \\ \rho, \sigma_1 \vdash E_2 \Rightarrow v_2, \sigma_2 \\ \vdots \\ \rho, \sigma_{n-1} \vdash E_n \Rightarrow v_n, \sigma_n \\ \rho'[x_1 \mapsto l_1, \cdots, x_n \mapsto l_n], \sigma_n[l_1 \mapsto v_1, \cdots, l_n \mapsto v_n] \vdash E \Rightarrow v, \sigma'\end{array}}{\rho, \sigma \vdash E_0\ (E_1, \cdots, E_n) \Rightarrow v, \sigma'}\ l_1, \cdots, l_n \notin Dom(\sigma_n)$$

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

Implement an interpreter of `miniC` by writing a function

$$\texttt{eval} : \texttt{program} \to \texttt{env} \to \texttt{mem} \to (\texttt{value} * \texttt{mem})$$

in file `c.ml`. Raise an exception `UndefinedSemantics` whenever the semantics is undefined. Skeleton code will be provided (before you start, see `README.md`).

Exercise 2 New memory is allocated in let, call, and record expressions. Allocated memory is never deallocated during program execution, eventually leading to memory exhaustion.

Write a function

$$\texttt{gc} : \texttt{env} * \texttt{mem} \to \texttt{mem}$$

that returns memory $\texttt{gc}(\rho, \sigma)$ consisting of the set of locations in σ that are reachable from the entries in ρ.

See page 20 of the slides of Lecture 9 for the formal definition of the garbage-collecting procedure.