

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:

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

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:

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

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

$$\begin{aligned}
&\overline{\rho, \sigma \vdash \text{skip} \Rightarrow \cdot, \sigma} \\
&\overline{\rho, \sigma \vdash \text{true} \Rightarrow true, \sigma} \quad \overline{\rho, \sigma \vdash \text{false} \Rightarrow false, \sigma} \\
&\overline{\rho, \sigma \vdash n \Rightarrow n, \sigma} \quad \overline{\rho, \sigma \vdash x \Rightarrow \sigma(\rho(x)), \sigma} \\
&\overline{\rho, \sigma \vdash \text{proc } (x_1, \dots, x_n) E \Rightarrow ((x_1, \dots, x_n), E, \rho), \sigma}
\end{aligned}$$

Unary and Binary Operations

$$\begin{aligned}
&\frac{\rho, \sigma_0 \vdash E_1 \Rightarrow n_1, \sigma_1 \quad \rho, \sigma_1 \vdash E_2 \Rightarrow n_2, \sigma_2}{\rho, \sigma_0 \vdash E_1 \oplus E_2 \Rightarrow n_1 \oplus n_2, \sigma_2} \oplus \in \{+, -, *, /\} \\
&\frac{\rho, \sigma_0 \vdash E_1 \Rightarrow n_1, \sigma_1 \quad \rho, \sigma_1 \vdash E_2 \Rightarrow n_2, \sigma_2}{\rho, \sigma_0 \vdash E_1 \leq E_2 \Rightarrow true, \sigma_2} n_1 \leq n_2 \\
&\frac{\rho, \sigma_0 \vdash E_1 \Rightarrow n_1, \sigma_1 \quad \rho, \sigma_1 \vdash E_2 \Rightarrow n_2, \sigma_2}{\rho, \sigma_0 \vdash E_1 \leq E_2 \Rightarrow false, \sigma_2} n_1 > n_2 \\
&\frac{\rho, \sigma_0 \vdash E_1 \Rightarrow v_1, \sigma_1 \quad \rho, \sigma_1 \vdash E_2 \Rightarrow v_2, \sigma_2}{\rho, \sigma_0 \vdash E_1 = E_2 \Rightarrow true, \sigma_2} v_1 = v_2 = n \vee v_1 = v_2 = b \vee v_1 = v_2 = \cdot \\
&\frac{\rho, \sigma_0 \vdash E_1 \Rightarrow v_1, \sigma_1 \quad \rho, \sigma_1 \vdash E_2 \Rightarrow v_2, \sigma_2}{\rho, \sigma_0 \vdash E_1 = E_2 \Rightarrow false, \sigma_2} \text{otherwise} \\
&\frac{\rho, \sigma_0 \vdash E \Rightarrow b, \sigma_1}{\rho, \sigma_0 \vdash \text{not } E \Rightarrow \text{not } b, \sigma_1}
\end{aligned}$$

Flow Controls

$$\begin{array}{c}
\frac{\rho, \sigma_0 \vdash E_1 \Rightarrow \text{true}, \sigma_1 \quad \rho, \sigma_1 \vdash E_2 \Rightarrow v, \sigma_2}{\rho, \sigma_0 \vdash \text{if } E_1 \text{ then } E_2 \text{ else } E_3 \Rightarrow v, \sigma_2} \\
\\
\frac{\rho, \sigma_0 \vdash E_1 \Rightarrow \text{false}, \sigma_1 \quad \rho, \sigma_1 \vdash E_3 \Rightarrow v, \sigma_2}{\rho, \sigma_0 \vdash \text{if } E_1 \text{ then } E_2 \text{ else } E_3 \Rightarrow v, \sigma_2} \\
\\
\frac{\rho, \sigma_0 \vdash E_1 \Rightarrow \text{false}, \sigma_1}{\rho, \sigma_0 \vdash \text{while } E_1 \text{ } E_2 \Rightarrow \cdot, \sigma_1} \\
\\
\frac{\rho, \sigma \vdash E_1 \Rightarrow \text{true}, \sigma_0 \quad \rho, \sigma_0 \vdash E_2 \Rightarrow v_1, \sigma_1 \quad \rho, \sigma_1 \vdash \text{while } E_1 \text{ } E_2 \Rightarrow v_2, \sigma_2}{\rho, \sigma \vdash \text{while } E_1 \text{ } E_2 \Rightarrow v_2, \sigma_2} \\
\\
\frac{\rho, \sigma_0 \vdash E_1 \Rightarrow v_1, \sigma_1 \quad \rho, \sigma_1 \vdash E_2 \Rightarrow v_2, \sigma_2}{\rho, \sigma_0 \vdash E_1; E_2 \Rightarrow v_2, \sigma_2}
\end{array}$$

Records

$$\begin{array}{c}
\overline{\rho, \sigma \vdash \{\} \Rightarrow \cdot, \sigma} \\
\\
\frac{\rho, \sigma_0 \vdash E_1 \Rightarrow v_1, \sigma_1 \quad \rho, \sigma_1 \vdash E_2 \Rightarrow v_2, \sigma_2 \quad \cdots \quad \rho, \sigma_{n-1} \vdash E_n \Rightarrow v_n, \sigma_n \quad l_1, \dots, l_n \notin \text{Dom}(\sigma_n)}{\rho, \sigma_0 \vdash \{ x_1 := E_1, \dots, x_n := E_n \} \Rightarrow \{x_1 \mapsto l_1, \dots, x_n \mapsto l_n\}, [l_1 \mapsto v_1, \dots, l_n \mapsto v_n] \sigma_n} \\
\\
\frac{\rho, \sigma_0 \vdash E \Rightarrow r, \sigma_1}{\rho, \sigma_0 \vdash E.x \Rightarrow \sigma_1(r(x)), \sigma_1} \\
\\
\frac{\rho, \sigma_0 \vdash E_1 \Rightarrow r, \sigma_1 \quad \rho, \sigma_1 \vdash E_2 \Rightarrow v, \sigma_2}{\rho, \sigma_0 \vdash E_1.x := E_2 \Rightarrow v, [r(x) \mapsto v] \sigma_2}
\end{array}$$

Assignments

$$\begin{array}{c}
\frac{\rho, \sigma_0 \vdash E \Rightarrow v, \sigma_1}{\rho, \sigma_0 \vdash x := E \Rightarrow v, [\rho(x) \mapsto v] \sigma_1} \\
\\
\frac{\rho, \sigma_0 \vdash E_1 \Rightarrow v_1, \sigma_1 \quad [x \mapsto l] \rho, [l \mapsto v_1] \sigma_1 \vdash E_2 \Rightarrow v, \sigma_2}{\rho, \sigma_0 \vdash \text{let } x := E_1 \text{ in } E_2 \Rightarrow v, \sigma_2} \quad l \notin \text{Dom}(\sigma_1)
\end{array}$$

Function Calls

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

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

Implement an interpreter of `miniC` by writing a function

`eval : program → env → mem → (value * 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

`gc : env * mem → mem`

that returns memory `gc(ρ, σ)` 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.