

Homework 4 (Term Project)

COSE212, Fall 2015

Hakjoo Oh

Due: 11/14, 24:00

In this project, we design and implement an imperative language, called B. The language is a small subset of C (this is why it is named so). The syntax and semantics are defined in Sections 1 and 2, respectively. Your task is to understand the language and implement an interpreter for it.

1 Syntax

<i>Expression</i> e	\rightarrow	unit	unit
		$x := e$	assignment
		$e ; e$	sequence
		if e then e else e	branch
		while e do e	while loop
		read x	input
		write e	output
		let $x := e$ in e	variable binding
		let proc $f(x_1, x_2, \dots, x_n) = e$ in e	procedure binding
		$f(e_1, e_2, \dots, e_n)$	call by value
		$f\langle x_1, x_2, \dots, x_n \rangle$	call by reference
		n	integer
		true false	boolean
		$\{ \}$ $\{x_1 := e_1, x_2 := e_2, \dots, x_n := e_n\}$	record (i.e., struct)
		$e.x$	record lookup
		$e.x := e$	record assignment
		x	identifier
		$e + e$ $e - e$ $e * e$ e / e	arithmetic operation
		$e < e$ $e = e$ not e	boolean operation

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.

2 Semantics

Domain

x, y	\in	Id	identifier (variable)
l	\in	$Addr$	address (memory location)
n	\in	\mathbb{Z}	integer
b	\in	$\mathbb{B} = \{true, false\}$	
r	\in	$Record = Id \rightarrow Addr$	
v	\in	$Val = \mathbb{Z} + \mathbb{B} + \{\cdot\} + Record$	
σ	\in	$Env = Id \rightarrow Addr + Procedure$	
M	\in	$Mem = Addr \rightarrow Val$	
		$Procedure = (Id \times Id \times \dots) \times Expression \times Env$	

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. Note that we design B in a way that procedures are not stored in memory, which means that procedures are not first-class objects in B.

Semantics Rules

TRUE	$\frac{}{\sigma, M \vdash \mathbf{true} \Rightarrow true, M}$	FALSE	$\frac{}{\sigma, M \vdash \mathbf{false} \Rightarrow false, M}$
NUM	$\frac{}{\sigma, M \vdash \mathbf{n} \Rightarrow n, M}$	UNIT	$\frac{}{\sigma, M \vdash \mathbf{unit} \Rightarrow \cdot, M}$
VAR	$\frac{}{\sigma, M \vdash x \Rightarrow M(\sigma(x)), M}$	RECF	$\frac{}{\sigma, M \vdash \{\cdot\} \Rightarrow \cdot, M}$
$\begin{array}{c} \sigma, M \vdash e_1 \Rightarrow v_1, M_1 \\ \sigma, M_1 \vdash e_2 \Rightarrow v_2, M_2 \\ \vdots \\ \sigma, M_{n-1} \vdash e_n \Rightarrow v_n, M_n \end{array}$			
RECT	$\frac{}{\sigma, M \vdash \{x_1 := e_1, \dots, x_n := e_n\} \Rightarrow \{x_1 \mapsto l_1, \dots, x_n \mapsto l_n\}, M_n \{l_1 \mapsto v_1, \dots, l_n \mapsto v_n\}} \quad l_i \notin Dom(M_n)$		
ADD	$\frac{\sigma, M \vdash e_1 \Rightarrow n_1, M' \quad \sigma, M' \vdash e_2 \Rightarrow n_2, M''}{\sigma, M \vdash e_1 + e_2 \Rightarrow n_1 + n_2, M''}$		
SUB	$\frac{\sigma, M \vdash e_1 \Rightarrow n_1, M' \quad \sigma, M' \vdash e_2 \Rightarrow n_2, M''}{\sigma, M \vdash e_1 - e_2 \Rightarrow n_1 - n_2, M''}$		
MUL	$\frac{\sigma, M \vdash e_1 \Rightarrow n_1, M' \quad \sigma, M' \vdash e_2 \Rightarrow n_2, M''}{\sigma, M \vdash e_1 * e_2 \Rightarrow n_1 * n_2, M''}$		
DIV	$\frac{\sigma, M \vdash e_1 \Rightarrow n_1, M' \quad \sigma, M' \vdash e_2 \Rightarrow n_2, M''}{\sigma, M \vdash e_1 / e_2 \Rightarrow n_1 / n_2, M''}$		

$$\begin{array}{c}
\text{EQUALT} \frac{\sigma, M \vdash e_1 \Rightarrow v_1, M' \quad \sigma, M' \vdash e_2 \Rightarrow v_2, M'' \quad \begin{array}{l} v_1 = v_2 = n \\ \vee v_1 = v_2 = b \\ \vee v_1 = v_2 = . \end{array}}{\sigma, M \vdash e_1 = e_2 \Rightarrow \mathbf{true}, M''} \\
\\
\text{EQUALF} \frac{\sigma, M \vdash e_1 \Rightarrow v_1, M' \quad \sigma, M' \vdash e_2 \Rightarrow v_2, M''}{\sigma, M \vdash e_1 = e_2 \Rightarrow \mathbf{false}, M''} \quad \text{otherwise} \\
\\
\text{LESS} \frac{\sigma, M \vdash e_1 \Rightarrow n_1, M' \quad \sigma, M' \vdash e_2 \Rightarrow n_2, M''}{\sigma, M \vdash e_1 < e_2 \Rightarrow n_1 < n_2, M''} \\
\\
\text{NOT} \frac{\sigma, M \vdash e \Rightarrow b, M'}{\sigma, M \vdash \mathbf{not} \ e \Rightarrow \mathbf{not} \ b, M'} \\
\\
\text{ASSIGN} \frac{\sigma, M \vdash e \Rightarrow v, M'}{\sigma, M \vdash x := e \Rightarrow v, M' \{ \sigma(x) \mapsto v \}} \\
\\
\text{RECASSIGN} \frac{\sigma, M \vdash e_1 \Rightarrow r, M_1 \quad \sigma, M_1 \vdash e_2 \Rightarrow v, M_2}{\sigma, M \vdash e_1 . x := e_2 \Rightarrow v, M_2 \{ r(x) \mapsto v \}} \\
\\
\text{RECLOOKUP} \frac{\sigma, M \vdash e \Rightarrow r, M'}{\sigma, M \vdash e . x \Rightarrow M'(r(x)), M'} \\
\\
\text{SEQ} \frac{\sigma, M \vdash e_1 \Rightarrow v_1, M' \quad \sigma, M' \vdash e_2 \Rightarrow v_2, M''}{\sigma, M \vdash e_1 \ ; \ e_2 \Rightarrow v_2, M''} \\
\\
\text{IFT} \frac{\sigma, M \vdash e \Rightarrow \mathbf{true}, M' \quad \sigma, M' \vdash e_1 \Rightarrow v, M''}{\sigma, M \vdash \mathbf{if} \ e \ \mathbf{then} \ e_1 \ \mathbf{else} \ e_2 \Rightarrow v, M''} \\
\\
\text{IFF} \frac{\sigma, M \vdash e \Rightarrow \mathbf{false}, M' \quad \sigma, M' \vdash e_2 \Rightarrow v, M''}{\sigma, M \vdash \mathbf{if} \ e \ \mathbf{then} \ e_1 \ \mathbf{else} \ e_2 \Rightarrow v, M''} \\
\\
\text{WHILEF} \frac{\sigma, M \vdash e_1 \Rightarrow \mathbf{false}, M'}{\sigma, M \vdash \mathbf{while} \ e_1 \ \mathbf{do} \ e_2 \Rightarrow ., M'} \\
\\
\text{WHILET} \frac{\sigma, M \vdash e_1 \Rightarrow \mathbf{true}, M' \quad \sigma, M' \vdash e_2 \Rightarrow v_1, M_1 \quad \sigma, M_1 \vdash \mathbf{while} \ e_1 \ \mathbf{do} \ e_2 \Rightarrow v_2, M_2}{\sigma, M \vdash \mathbf{while} \ e_1 \ \mathbf{do} \ e_2 \Rightarrow v_2, M_2}
\end{array}$$

$$\begin{array}{c}
\text{LETV} \frac{\sigma, M \vdash e_1 \Rightarrow v, M' \quad \sigma\{x \mapsto l\}, M'\{l \mapsto v\} \vdash e_2 \Rightarrow v', M''}{\sigma, M \vdash \text{let } x := e_1 \text{ in } e_2 \Rightarrow v', M''} \quad l \notin \text{Dom } M' \\
\\
\text{LETF} \frac{\sigma\{f \mapsto \langle (x_1, \dots, x_n), e_1, \sigma \rangle\}, M \vdash e_2 \Rightarrow v, M'}{\sigma, M \vdash \text{let proc } f(x_1, \dots, x_n) = e_1 \text{ in } e_2 \Rightarrow v, M'} \\
\\
\begin{array}{c}
\sigma, M \vdash e_1 \Rightarrow v_1, M_1 \quad \sigma, M_1 \vdash e_2 \Rightarrow v_2, M_2 \\
\vdots \\
\sigma, M_{n-1} \vdash e_n \Rightarrow v_n, M_n
\end{array} \\
\text{CALLV} \frac{\begin{array}{c} \sigma'\{x_1 \mapsto l_1\} \dots \{x_n \mapsto l_n\} \{f \mapsto \langle (x_1, \dots, x_n), e', \sigma' \rangle\}, \\ M_n\{l_1 \mapsto v_1\} \dots \{l_n \mapsto v_n\} \vdash e' \Rightarrow v', M' \end{array}}{\sigma, M \vdash f(e_1, \dots, e_n) \Rightarrow v', M'} \quad \begin{array}{c} \sigma(f) = \langle (x_1, \dots, x_n), e', \sigma' \rangle \\ l_i \notin \text{Dom } M_n \end{array} \\
\\
\text{CALLR} \frac{\begin{array}{c} \sigma'\{x_1 \mapsto \sigma(y_1)\} \dots \{x_n \mapsto \sigma(y_n)\} \{f \mapsto \langle (x_1, \dots, x_n), e, \sigma' \rangle\}, \\ M \vdash e \Rightarrow v, M' \end{array}}{\sigma, M \vdash f\langle y_1, \dots, y_n \rangle \Rightarrow v, M'} \quad \sigma(f) = \langle (x_1, \dots, x_n), e, \sigma' \rangle \\
\\
\text{READ} \frac{}{\sigma, M \vdash \text{read } x \Rightarrow n, M\{\sigma(x) \mapsto n\}} \\
\\
\text{WRITE} \frac{\sigma, M \vdash e \Rightarrow n, M'}{\sigma, M \vdash \text{write } e \Rightarrow n, M'}
\end{array}$$

3 Instruction

1. Clone the project framework from Github in your Ubuntu system:
`git clone https://github.com/hakjoooh/ProgrammingLanguagesProject2015.git`
The framework contains a skeleton implementation, including a parser.
2. Your job is to complete the interpreter implementation in `b.ml`. Specifically, implement the function

```
eval: memory -> env -> exp -> (value * memory)
```

in module `B` (in `b.ml`). Before implementing it, carefully read `readme.txt`.
3. Submit the single file `b.ml` via Blackboard.

Enjoy!