Homework 4
COSE212, Fall 2019

Hakjoo Oh

Due: 11/25, 23:59

Problem 1 (100pts) Let us design and implement an imperative language, called B, which is a subset of the C programming language. The syntax of B is as follows:

$$
\begin{array}{llll}
e & \to & \texttt{unit} & \text{unit} \\
& | & x \ \texttt{:=} \ e & \text{assignment} \\
& | & e \ \texttt{;} \ e & \text{sequence} \\
& | & \texttt{if } e \texttt{ then } e \texttt{ else } e & \text{branch} \\
& | & \texttt{while } e \texttt{ do } e & \text{while loop} \\
& | & \texttt{write } e & \text{output} \\
& | & \texttt{let } x \ \texttt{:=} \ e \ \texttt{in } e & \text{variable binding} \\
& | & \texttt{let proc } f(x_1, x_2, \cdots, x_n) \ \texttt{= } e \ \texttt{in } e & \text{procedure binding} \\
& | & f(e_1, e_2, \cdots, e_n) & \text{call by value} \\
& | & f\texttt{<}x_1, x_2, \cdots, x_n\texttt{>} & \text{call by reference} \\
& | & n & \text{integer} \\
& | & \texttt{true} \quad | \quad \texttt{false} & \text{boolean} \\
& | & \texttt{\{\}} \quad | \quad \texttt{\{}x_1\texttt{:=}e_1, x_2\texttt{:=}e_2, \cdots, x_n\texttt{:=}e_n\texttt{\}} & \text{record (i.e., struct)} \\
& | & e\texttt{.}x & \text{record lookup} \\
& | & e\texttt{.}x \ \texttt{:=} \ e & \text{record assignment} \\
& | & x & \text{identifier} \\
& | & e \ \texttt{+} \ e \ \ | \ \ e \ \texttt{-} \ e \ \ | \ \ e \ \texttt{*} \ e \ \ | \ \ e \ \texttt{/} \ e & \text{arithmetic operation} \\
& | & e \ \texttt{<} \ e \ \ | \ \ e \ \texttt{=} \ e \ \ | \ \ \texttt{not } e & \text{boolean operation}
\end{array}
$$

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:

$$
\begin{array}{rcccl}
x, y & \in & Id & & \text{identifier (variable)} \\
l & \in & Addr & & \text{address (memory location)} \\
n & \in & \mathbb{Z} & & \text{integer} \\
b & \in & \mathbb{B} & = & \{true, false\} \\
r & \in & Record & = & Id \to Addr \\
v & \in & Val & = & \mathbb{Z} + \mathbb{B} + \{\cdot\} + Record \\
\sigma & \in & Env & = & Id \to Addr + Procedure \\
M & \in & Mem & = & Addr \to Val \\
& & Procedure & = & (Id \times Id \times \cdots) \times Expression \times Env
\end{array}
$$

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. Note that we design B in a way that procedures are not stored in memory, which means that procedures are not first-class values in B. The semantics of the language is defined as follows (Below, we write $\sigma\{x \mapsto l\}$ and $M\{l \mapsto v\}$ for the environment σ and memory M extended with the new entries):

$$\text{TRUE } \frac{}{\sigma, M \vdash \texttt{true} \Rightarrow true, M} \qquad \text{FALSE } \frac{}{\sigma, M \vdash \texttt{false} \Rightarrow false, M}$$

$$\text{NUM } \frac{}{\sigma, M \vdash \texttt{n} \Rightarrow n, M} \qquad \text{UNIT } \frac{}{\sigma, M \vdash \texttt{unit} \Rightarrow \cdot, M}$$

$$\text{VAR } \frac{}{\sigma, M \vdash x \Rightarrow M(\sigma(x)), M} \qquad \text{RECF } \frac{}{\sigma, M \vdash \texttt{\{\}} \Rightarrow \cdot, M}$$

$$\text{RECT } \frac{\begin{array}{c} \sigma, M \vdash e_1 \Rightarrow v_1, M_1 \\ \sigma, M_1 \vdash e_2 \Rightarrow v_2, M_2 \\ \vdots \\ \sigma, M_{n-1} \vdash e_n \Rightarrow v_n, M_n \end{array}}{\begin{array}{c} \sigma, M \vdash \texttt{\{}x_1 \texttt{ := } e_1, \cdots, x_n \texttt{ := } e_n\texttt{\}} \Rightarrow \\ \{x_1 \mapsto l_1, \cdots, x_n \mapsto l_n\}, M_n\{l_1 \mapsto v_1, \cdots, l_n \mapsto v_n\} \end{array}} \ \forall i.\, l_i \notin Dom(M_n)$$

$$\text{ADD } \frac{\sigma, M \vdash e_1 \Rightarrow n_1, M' \qquad \sigma, M' \vdash e_2 \Rightarrow n_2, M''}{\sigma, M \vdash e_1 \texttt{ + } e_2 \Rightarrow n_1 + n_2, M''}$$

$$\text{SUB } \frac{\sigma, M \vdash e_1 \Rightarrow n_1, M' \qquad \sigma, M' \vdash e_2 \Rightarrow n_2, M''}{\sigma, M \vdash e_1 \texttt{ - } e_2 \Rightarrow n_1 - n_2, M''}$$

$$\text{MUL } \frac{\sigma, M \vdash e_1 \Rightarrow n_1, M' \qquad \sigma, M' \vdash e_2 \Rightarrow n_2, M''}{\sigma, M \vdash e_1 \texttt{ * } e_2 \Rightarrow n_1 * n_2, M''}$$

$$\text{DIV } \frac{\sigma, M \vdash e_1 \Rightarrow n_1, M' \qquad \sigma, M' \vdash e_2 \Rightarrow n_2, M''}{\sigma, M \vdash e_1 \texttt{ / } e_2 \Rightarrow n_1 / n_2, M''}$$

$$\text{EQUALT } \frac{\sigma, M \vdash e_1 \Rightarrow v_1, M' \qquad \sigma, M' \vdash e_2 \Rightarrow v_2, M''}{\sigma, M \vdash e_1 \ = \ e_2 \Rightarrow \texttt{true}, M''} \quad \begin{array}{l} v_1 = v_2 = n \\ \lor \ v_1 = v_2 = b \\ \lor \ v_1 = v_2 = \cdot \end{array}$$

$$\text{EQUALF } \frac{\sigma, M \vdash e_1 \Rightarrow v_1, M' \qquad \sigma, M' \vdash e_2 \Rightarrow v_2, M''}{\sigma, M \vdash e_1 \ = \ e_2 \Rightarrow \texttt{false}, M''} \quad \text{otherwise}$$

$$\text{LESS } \frac{\sigma, M \vdash e_1 \Rightarrow n_1, M' \qquad \sigma, M' \vdash e_2 \Rightarrow n_2, M''}{\sigma, M \vdash e_1 \ \texttt{<} \ e_2 \Rightarrow n_1 < n_2, M''}$$

$$\text{NOT } \frac{\sigma, M \vdash e \Rightarrow b, M'}{\sigma, M \vdash \texttt{not} \ e \Rightarrow not \ b, M'}$$

$$\text{ASSIGN } \frac{\sigma, M \vdash e \Rightarrow v, M'}{\sigma, M \vdash x \ \texttt{:=} \ e \Rightarrow v, M'\{\sigma(x) \mapsto v\}}$$

$$\text{RECASSIGN } \frac{\sigma, M \vdash e_1 \Rightarrow r, M_1 \qquad \sigma, M_1 \vdash e_2 \Rightarrow v, M_2}{\sigma, M \vdash e_1.x \ \texttt{:=} \ e_2 \Rightarrow v, M_2\{r(x) \mapsto v\}}$$

$$\text{RECLOOKUP } \frac{\sigma, M \vdash e \Rightarrow r, M'}{\sigma, M \vdash e.x \Rightarrow M'(r(x)), M'}$$

$$\text{SEQ } \frac{\sigma, M \vdash e_1 \Rightarrow v_1, M' \qquad \sigma, M' \vdash e_2 \Rightarrow v_2, M''}{\sigma, M \vdash e_1 \ \texttt{;} \ e_2 \Rightarrow v_2, M''}$$

$$\text{IFT } \frac{\sigma, M \vdash e \Rightarrow true, M' \qquad \sigma, M' \vdash e_1 \Rightarrow v, M''}{\sigma, M \vdash \texttt{if} \ e \ \texttt{then} \ e_1 \ \texttt{else} \ e_2 \Rightarrow v, M''}$$

$$\text{IFF } \frac{\sigma, M \vdash e \Rightarrow false, M' \qquad \sigma, M' \vdash e_2 \Rightarrow v, M''}{\sigma, M \vdash \texttt{if} \ e \ \texttt{then} \ e_1 \ \texttt{else} \ e_2 \Rightarrow v, M''}$$

$$\text{WHILEF } \frac{\sigma, M \vdash e_1 \Rightarrow false, M'}{\sigma, M \vdash \texttt{while} \ e_1 \ \texttt{do} \ e_2 \Rightarrow \cdot, M'}$$

$$\text{WHILET } \frac{\sigma, M \vdash e_1 \Rightarrow true, M' \qquad \sigma, M' \vdash e_2 \Rightarrow v_1, M_1 \qquad \sigma, M_1 \vdash \texttt{while} \ e_1 \ \texttt{do} \ e_2 \Rightarrow v_2, M_2}{\sigma, M \vdash \texttt{while} \ e_1 \ \texttt{do} \ e_2 \Rightarrow v_2, M_2}$$

$$\text{LETV} \quad \frac{\sigma, M \vdash e_1 \Rightarrow v, M' \qquad \sigma\{x \mapsto l\}, M'\{l \mapsto v\} \vdash e_2 \Rightarrow v', M''}{\sigma, M \vdash \texttt{let } x \texttt{ := } e_1 \texttt{ in } e_2 \Rightarrow v', M''} \quad l \notin Dom(M')$$

$$\text{LETF} \quad \frac{\sigma\{f \mapsto \langle (x_1, \cdots, x_n), e_1, \sigma \rangle\}, M \vdash e_2 \Rightarrow v, M'}{\sigma, M \vdash \texttt{let proc } f(x_1, \cdots, x_n) \texttt{ = } e_1 \texttt{ in } e_2 \Rightarrow v, M'}$$

$$\text{CALLV} \quad \frac{\begin{array}{c} \sigma, M \vdash e_1 \Rightarrow v_1, M_1 \\ \sigma, M_1 \vdash e_2 \Rightarrow v_2, M_2 \\ \vdots \\ \sigma, M_{n-1} \vdash e_n \Rightarrow v_n, M_n \\ \sigma'\{x_1 \mapsto l_1\} \cdots \{x_n \mapsto l_n\}\{f \mapsto \langle (x_1, \cdots, x_n), e', \sigma' \rangle\}, \\ M_n\{l_1 \mapsto v_1\} \cdots \{l_n \mapsto v_n\} \vdash e' \Rightarrow v', M' \end{array}}{\sigma, M \vdash f(e_1, \cdots, e_n) \Rightarrow v', M'} \quad \begin{array}{l} \sigma(f) = \langle (x_1, \cdots, x_n), e', \sigma' \rangle \\ \forall i.\, l_i \notin Dom(M_n) \end{array}$$

$$\text{CALLR} \quad \frac{\begin{array}{c} \sigma'\{x_1 \mapsto \sigma(y_1)\} \cdots \{x_n \mapsto \sigma(y_n)\}\{f \mapsto \langle (x_1, \cdots, x_n), e, \sigma' \rangle\}, \\ M \vdash e \Rightarrow v, M' \end{array}}{\sigma, M \vdash f\texttt{<}y_1, \cdots, y_n\texttt{>} \Rightarrow v, M'} \quad \sigma(f) = \langle (x_1, \cdots, x_n), e, \sigma' \rangle$$

$$\text{WRITE} \quad \frac{\sigma, M \vdash e \Rightarrow n, M'}{\sigma, M \vdash \texttt{write } e \Rightarrow n, M'}$$

In OCaml, the language and values can be defined as follows:

```
type exp =
  | NUM of int | TRUE | FALSE | UNIT
  | VAR of id
  | ADD of exp * exp
  | SUB of exp * exp
  | MUL of exp * exp
  | DIV of exp * exp
  | EQUAL of exp * exp
  | LESS of exp * exp
  | NOT of exp
  | SEQ of exp * exp                  (* sequence *)
  | IF of exp * exp * exp             (* if-then-else *)
  | WHILE of exp * exp                (* while loop *)
  | LETV of id * exp * exp            (* variable binding *)
  | LETF of id * id list * exp * exp  (* procedure binding *)
  | CALLV of id * exp list            (* call by value *)
  | CALLR of id * id list             (* call by referenece *)
  | RECORD of (id * exp) list         (* record construction *)
  | FIELD of exp * id                 (* access record field *)
  | ASSIGN of id * exp                (* assgin to variable *)
  | ASSIGNF of exp * id * exp         (* assign to record field *)
```

4

```
  | WRITE of exp
and id = string

type loc = int
type value =
| Num of int
| Bool of bool
| Unit
| Record of record
and record = (id * loc) list
type memory = (loc * value) list
type env = binding list
and binding = LocBind of id * loc | ProcBind of id * proc
and proc = id list * exp * env
```

Implemente the function `runb`:

$$\text{runb} : \text{ exp } \text{->} \text{ value}$$

which takes a program expression and computes its value. Whenever the semantics is undefined, raise the exception `UndefinedSemantics`.

Examples:

- The program

```
let ret = 1 in
let n = 5 in
while (0 < n) {
    ret := ret * n;
    n := n - 1;
};
ret
```

is represented by

```
LETV ("ret", NUM 1,
    LETV ("n", NUM 5,
        SEQ (
            WHILE (LESS (NUM 0, VAR "n"),
                SEQ (
                    ASSIGN ("ret", MUL (VAR "ret", VAR "n")),
                    ASSIGN ("n", SUB (VAR "n", NUM 1))
                )
            ),
            VAR "ret")))
```

and produces 120.

- The program

```
let proc f (x1, x2) =
    x1 := 3;
    x2 := 3;
in
let x1 = 1 in
let x2 = 1 in
f <x1, x2>;
x1 + x2
```

  is represented by

```
LETF ("f", ["x1"; "x2"],
    SEQ (
        ASSIGN ("x1", NUM 3),
        ASSIGN ("x2", NUM 3)
    ),
    LETV("x1", NUM 1,
        LETV("x2", NUM 1,
            SEQ(
                CALLR ("f", ["x1"; "x2"]),
                ADD(VAR "x1", VAR "x2")))))
```

  and produces 6.

- The program

```
let f = {x := 10, y := 13} in
let proc swap (a, b) =
    let temp = a in
    a := b;
    b := temp
in
swap (f.x, f.y);
f.x
```

  is represented by

```
LETV ("f", RECORD ([("x", NUM 10); ("y", NUM 13)]),
    LETF ("swap", ["a"; "b"],
        LETV ("temp", VAR "a",
            SEQ (
                ASSIGN ("a", VAR "b"),
                ASSIGN ("b", VAR "temp"))),
        SEQ (
```

```
            CALLV("swap", [FIELD (VAR "f", "x"); FIELD (VAR "f", "y")]),
            FIELD (VAR "f", "x")
        )
      )
)
```

and produces 10.