MP 2 – Extending a Simple Imperative Programming Language in K with Declarations and References
CS 422 – Spring 2016
Revision 1.1

Assigned March 18, 2016
Due April 3, 2016, 8:00pm
Extension 24 hours (10% penalty)

1 Change Log

1.1 Cleaned up the use of $l$ and $l'$ in the rules for the semantics expressions. Added $I = lvalue(M)$ to rule for Assignment in IMP3.

1.0 Initial Release.

2 Turn-In Procedure

Put your code as plain text for this MP in a file named $mp2.k$, and submit your plain text file $mp2.k$ by first adding it to your svn repository directory assignments/mp2, which may be done using the command (svn add mp2.k) and then committing it using (svn commit -m "submitting mp2" mp2.k). Your file should contain your name, and netid in a comment at the top, and it should contain your solution. It should be named $mp2.k$ and committed in your assignments/mp2 directory.

3 Objectives

The purpose of this MP is to familiarize you with specifying how references work in an imperative programming language, both when they occur (as r-variables, a.k.a. $rvars$) in expressions whose value we are computing, and when they occur (as l-variables, a.k.a $lvars$), and how these interact with declarations.

4 Background

In MP1, you implemented the core of a simple imperative programming language, IMP1, which can be seen as a fragment of the C programming language. In this assignment, you will be asked to extend the K specification of this language to include variable declarations and references. This problem is similar to the problem we have worked in class of adding references to SIMP. We will begin by giving the syntax and Natural Semantics related to SIMP with references as done in class. You may find the K file for this language at

https://courses.engr.illinois.edu/cs422/sp2016/lectures/simp3.k
4.1 Syntax of SIMP3

In class, we worked with specifying in K the language SIMP3 whose syntax is given by the BNF Grammar below:

\[
\begin{align*}
I & \quad \in \quad \text{Identifiers} \\
N & \quad \in \quad \text{Numerals} \\
M & \quad ::= \quad I \mid * M \\
E & \quad ::= \quad N \mid \text{nil} \mid M \mid (E) \mid E + E \mid E \ast E \mid E - E \\
B & \quad ::= \quad \text{true} \mid \text{false} \mid (B) \mid B \& B \mid B \text{ or } B \mid \text{not } B \mid E < E \mid E = E \\
C & \quad ::= \quad \text{skip} \mid (C) \mid C ; C \mid \{C\} \mid M := E \mid \text{if } B \text{ then } C \text{ else } C \text{ fi} \mid \text{while } B \text{ do } C \text{ od}
\end{align*}
\]

4.2 Natural Semantics for SIMP3

Let $L$ be a set of Locations (where we can always find a fresh one, i.e. one that is not in use in our current configuration). Let $l, l': \text{Identifiers} \rightarrow \text{Locations}$. We assume a set of Values of final results of expressions (in this case you can assume integers, nil, and references to locations, written $\text{ref}(i)$ for $i$ a Location). Let $m, m': \text{Locations} \rightarrow \text{Values}$. Parentheses are a construct for parsing only. In all syntactic categories, parenthesized entities have the semantics as the entity inside the parentheses. The Natural Semantics corresponding to the K semantics we gave in class for the SIMP3 is as follows:

**Expressions:**

- **Constants:**
  - Numerals are values: $(N, l, m) \Downarrow N$
  - Nil is a value: $(\text{nil}, l, m) \Downarrow \text{nil}$

- **Mutables:**
  - Identifiers: $(I, l, m) \Downarrow m(l(I))$ if $l(I)$ and $m(l(I))$ exist
  - References: $(M, l, m) \Downarrow \text{ref}(i)$ if $m(i)$ exist

- **Arithmetic Expressions:**
  \[
  (E, l, m) \Downarrow U \quad (E', l, m) \Downarrow V \quad U + V = N \quad \text{where } \oplus \in \{+,-,\ast\} \text{ and } U, V \in \text{Values} \]

- **Booleans:**
  - Boolean Constants: $(\text{true}, l, m) \Downarrow \text{true} \quad (\text{false}, l, m) \Downarrow \text{false}$

- **Arithmetic Relations:**
  \[
  (E, l, m) \Downarrow U \quad (E', l, m) \Downarrow V \quad U \sim V = b \quad \text{where } \sim \in \{==,<\} \]

\[\]

2
### Boolean Expressions:

<table>
<thead>
<tr>
<th>Boolean Expression</th>
<th>Evaluation</th>
</tr>
</thead>
<tbody>
<tr>
<td>((B, l, m) \downarrow \text{false})</td>
<td>((B, l, m) \downarrow \text{false})</td>
</tr>
<tr>
<td>((B &amp; B', l, m) \downarrow \text{false})</td>
<td>((B \text{ or } B', l, m) \downarrow \text{false})</td>
</tr>
<tr>
<td>((B, l, m) \downarrow \text{true})</td>
<td>((B', l, m) \downarrow \text{false})</td>
</tr>
</tbody>
</table>

### L-values:

\[ l\text{value}(I, l, m) = \begin{cases} 
\text{ref}(l(I), l, m) & \text{if } l(I) \text{ exists} \\
\text{ref}(j), l[I \leftarrow j], m & \text{if } l(I) \text{ doesn't exist, } j \text{ fresh for } l \text{ and } m 
\end{cases} \]

\[ l\text{value}(M, l, m) = (\text{ref}(i), l', m') \quad \text{if } m'(i) = \text{ref}(j) \]

\[ l\text{value}(*M, l, m) = (\text{ref}(j), l', m') \]

\[ l\text{value}(*M, l, m) = (\text{ref}(i), l', m') \quad \text{if } m'(i) \text{ does not exist, } j \text{ fresh for } l' \text{ and } m' \]

### Commands:

**Assignment:**

\[ (E, l, m) \downarrow V \quad l\text{value}(M, l, m) = (\text{ref}(i), l', m') \]

\[ (M := E, l, m) \downarrow (l', m'[i \leftarrow V]) \]

**Skip:**

\[ (\text{skip}, l, m) \downarrow (l, m) \]

**Sequencing:**

\[ (C, l, m) \downarrow (l', l', m') \quad (C', l', m') \downarrow (l'', m'') \]

\[ (C; C', l, m) \downarrow (l'', m'') \]

**Block:**

\[ (C), l, m) \downarrow (l', m') \]

**If-true:**

\[ (B, l, m) \downarrow \text{true} \quad (C, l, m) \downarrow (l', m') \]

\[ (\text{if } B \text{ then } C \text{ else } C', l, m) \downarrow (l', m') \]

**If-false:**

\[ (B, l, m) \downarrow \text{false} \quad (C', l, m) \downarrow (l', m') \]

\[ (\text{if } B \text{ then } C \text{ else } C', l, m) \downarrow (l', m') \]

**While-false:**

\[ (B, l, m) \downarrow \text{false} \]

\[ (\text{while } B \text{ do } C \text{ od}, l, m) \downarrow (l, m) \]

**While-true:**

\[ (B, l, m) \downarrow \text{true} \quad (C, l, m) \downarrow (l', m') \quad (\text{while } B \text{ do } C \text{ od}, l', m') \downarrow (l'', m'') \]

\[ (\text{while } B \text{ do } C \text{ od}, l, m) \downarrow (l'', m'') \]
4.3 Syntax of IMP3

The following is a BNF grammar for the language you are to specify in K. It is an extension of the language you specified in MP2.

\[
\begin{align*}
I & \in \text{Identifiers} \\
N & \in \text{Numerals} \\
V & ::= N \mid \text{nil} \\
M & ::= I \mid * M \\
E & ::= V \mid M \mid (E) \mid E * E \mid E + E \mid E - E \\
& \quad \mid E \cdot E \mid E == E \mid E \&\& E \mid E \| E \mid ! E \mid M = E \\
\text{Blk} & ::= \{ \} \mid \{ \text{StmtList} \} \\
\text{StmtList} & ::= \text{Stmt} \mid \text{StmtList} \\
\text{Stmt} & ::= E \mid \text{Blk} \mid \text{if} (E) \text{Stmt} \text{else} \text{Stmt} \mid \text{while} (E) \text{Stmt} \\
\text{Decl} & ::= \text{int} M = V ; \mid \text{int} M ; \\
\text{Prog} & ::= \text{Stmt} \mid \text{Decl Prog}
\end{align*}
\]

The unary operator * binds more tightly than the unary operator !, which, in turn, binds more tightly than any of the binary operators. The operator * binds the most tightly of the binary operations, with + and - having the same precedence as each and binding next most tightly. Below them are the relation operators, with less than < binding more tightly than equality ==. Logical and && binds next most tightly, followed by logical or ||, which, in turn, binds more tightly than assignment (=). Assignment associates to the right; all other binary operators given associate to the left.

4.4 Natural Semantics of IMP3

As with SIMPL3, let \( L \) be a set of Locations (where we can always find a fresh one, i.e. one that is not in use in our current configuration). Let \( l, l' : \text{Identifiers} \rightarrow \text{Locations} \). We assume a set of Values of final results of expressions (in this case you can assume integers, nil, and references to locations, written \( \text{ref}(i) \) for \( i \) a Location). Let \( m, m' : \text{Locations} \rightarrow \text{Values} \). The Natural Semantics for the IMP3 as follows:

Constants: Numerals are values: \( (N, l, m) \Downarrow (N, l, m) \) Nil is a value: \( (\text{nil}, l, m) \Downarrow (\text{nil}, l, m) \)

Mutables:

Identifiers: \( (I, l, m) \Downarrow (m(l(I)), l, m) \) if \( l(I) \) and \( m(l(I)) \) exist

References:

\[
\begin{align*}
(M, l, m) \Downarrow (\text{ref}(i), l, m) \\
(*M, l, m) \Downarrow (m(i), l, m)
\end{align*}
\]

if \( m(i) \) exist

Parentheses:

\[
\begin{align*}
((E), l, m) \Downarrow (V, l, m') \\
(E, l, m) \Downarrow (V, l, m')
\end{align*}
\]

Arithmetic Expressions:

\[
\begin{align*}
(E, l, m) \Downarrow (U, l, m') \\
(E', l, m') \Downarrow (V, l, m'') \\
U \oplus V = N \\
(E \oplus E', l, m) \Downarrow (N', l, m'')
\end{align*}
\]

where \( \oplus \in \{+, *, -\} \) and \( U, V \in \text{Values} \)
Arithmetic Relations:

\[
(E,l,m) \Downarrow (U,l,m') \quad (E',l,m') \Downarrow (V,l,m'') \quad U \sim V = \text{true} \\
(E \sim E',l,m) \Downarrow (1,l,m'') \\
(E,l,m) \Downarrow (U,l,m') \quad (E',l,m') \Downarrow (V,l,m'') \quad U \sim V = \text{false} \\
(E \sim E',l,m) \Downarrow (0,l,m'')
\]

where \( \sim \in \{=, <\} \)

Boolean Expressions:

\[
\begin{align*}
(E,l,m) & \Downarrow (0,l,m') \\
(E \& \& E',l,m) & \Downarrow (0,l,m') \\
(E,l,m) & \Downarrow (U,l,m') \\
(E,l,m) & \Downarrow (V,l,m') \quad V \neq 0 \\
(E \| E',l,m) & \Downarrow (1,l,m') \\
(E,l,m) & \Downarrow (0,l,m') \quad (E',l,m') \Downarrow (V,l,m'') \quad V \neq 0 \\
(E \| E',l,m) & \Downarrow (1,l,m') \\
(E,l,m) & \Downarrow (0,l,m') \quad (E',l,m') \Downarrow (0,l,m'') \\
(E \| E',l,m) & \Downarrow (0,l,m') \\
(E,l,m) & \Downarrow (0,l,m') \\
(E,l,m) & \Downarrow (V,l,m') \quad V \neq 0 \\
(E \neg, l,m) & \Downarrow (1,l,m') \\
(E,l,m) & \Downarrow (0,l,m') \\
(E \neg, l,m) & \Downarrow (0,l,m')
\end{align*}
\]

L-values:

\[
lvalue(I,l,m) = \text{ref}(I(I)) \quad \text{if } l(I) \text{ exists}
\]

\[
lvalue(M,l,m) = \text{ref}(i) \quad m(i) = \text{ref}(j)
\]

\[
lvalue(* M,l,m) = \text{ref}(j)
\]

Assignment:

\[
\begin{align*}
\text{lvalue } (* M, l, m) & \Rightarrow M \text{ requires exists } j. \quad M \Downarrow \text{ref } j \\
(E,l,m) & \Downarrow (V,l,m') \quad \text{lvalue}(M,l,m) = \text{ref}(i) \\
(M = E,l,m) & \Downarrow (V,l,m'[i \leftarrow V]) \quad \text{if } \text{lvalue}(M,l,m) \text{ exists}
\end{align*}
\]

Blocks:

Empty Block: \( \{ \}, l,m \Downarrow (l,m) \)

Block Sequence: \( (\text{StmtList},l,m) \Downarrow (l,m') \)

Statement Sequences:

Single Statement: \( (\text{Stmt},l,m) \Downarrow (l,m') \)

Statement Sequence: \( (\text{Stmt},l,m) \Downarrow (l,m') \quad (\text{StmtList},l,m') \Downarrow (l,m'') \\
(\text{Stmt StmtList},l,m) \Downarrow (l,m'') \)
5 Problems

1. In the file mp2.k define a module MP2–SYNTAX giving the syntax for IMP3 (borrowing heavily form IMP2), and a module MP2 giving the semantics for IMP3, consistent with the syntax and semantics given above. You may copy as much of the syntax and semantics of SIMP3 and IMP2 as you find useful.