MP 3 – Extending a Simple Imperative Programming Language in K with Blocks with Local Scope and Functions
CS 422 – Spring 2016
Revision 1.1

Assigned April 8, 2016
Due April 17, 2016, 8:00pm
Extension 24 hours (10% penalty)

1 Change Log

1.1 Changed the date and added a clause for Es.
1.0 Initial Release.

2 Turn-In Procedure

Put your code as plain text for this MP in a file named mp3.k, and submit your plain text file mp3.k by first adding it to your svn repository directory assignments/mp3, which may be done using the command (svn add mp3.k) and then committing it using (svn commit -m "submitting mp3" mp3.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 mp3.k and committed in your assignments/mp3 directory.

3 Objectives

The purpose of this MP is to familiarize you with specifying how functions in work in an imperative programming language, both in terms of how they provide a local context for evaluation and how their initialization and completion require switching evaluation contexts.

4 Background

In MP2, you extended your implementation of the core of a simple imperative programming language, IMP2, 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 function declarations and function application. This problem is similar to the problem we have worked in class of adding references to SIMP. You may find the K file for this language at

https://courses.engr.illinois.edu/cs422/sp2016/lectures/simp5.k
4.1 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.

\[ I \in \text{Identifiers} \]
\[ Is ::= \cdot | I, Is \]
\[ N \in \text{Numerals} \]
\[ V ::= N | \text{nil} \]
\[ M ::= I | * M \]
\[ E ::= V | M | (E) | E * E | E + E | E - E \]
\[ | E < E | E == E | E & & E | E || E | ! E | M = E | I(Es) \]

\[ Es ::= \cdot | E, ES \]
\[ Blk ::= \{ \} | \{ \text{Prog} \} \]
\[ StmtList ::= Stmnt | Stmt StmntList \]
\[ Stmnt ::= E; | \text{Blk} | \text{if (E) Stmnt else Stmnt} | \text{while (E) Stmnt} | \text{return E} \]
\[ Decl ::= \text{int M} = V; | \text{int M;} | \text{int I(Is) Blk;} \]
\[ Prog ::= StmtList | Decl Prog \]

The symbol \( \cdot \) is representing the empty sequence. 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.

4.2 Natural Semantics of IMP3

As we did in MP2, let \( I \) be a set of identifiers, let \( L \) be an infinite set of locations and fresh : \( \mathcal{P}(L) \rightarrow L \) a function selecting a location not in the input set of locations, if one exists. Let \( l, l' : I \rightarrow L \). We assume a set of values \( V_i \) of final results of expressions (in this case you can assume integers, nil, references to locations, written \( \text{ref}(i) \) for \( i \) a location, \( \bullet \) for the “value” of a completed computation, \( \text{rtn}(V_i) \) for the “value” of a computation returning a value and closures, written \( \text{closure}(Is, Blk, l) \) containing a parameter list \( Is \), a program block \( Blk \) and location map \( l \). Let \( m, m' : L \rightarrow V_i \).

As a part of a function call, we need to be able to take a list of parameters, a list of values, a location mapping and a memory, and generate an updated location mapping and updated memory adding bindings for each of the parameters to each corresponding value. This is accomplished by the function match:

\[
\begin{align*}
\text{match}(\cdot, \cdot, l,m) &= (l,m) \\
\text{match}((I, Is), (V_i, Vs), l,m) &= \text{match}(Is, Vs, l[I \leftarrow L], m[l \leftarrow V_i]) \\
\text{where } L &= \text{fresh}(\text{dom}(m) \cup \text{rng}(L))
\end{align*}
\]

Natural Semantics for the IMP3 as follows:

Expressions \((E)\):

- 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: \((M, l,m) \downarrow (\text{ref}(i), l,m')\) if \( m(i) \) exist

References: \((+M, l,m) \downarrow (m(i), l,m')\) if \( m(i) \) exist
Parentheses: \( ((E), l, m) \Downarrow (V), l, m' \) \( (E), l, m \Downarrow (V), l, m' \)

Arithmetic Expressions: \( (E), l, m \Downarrow (U), l, m' \) \( (E'), l, m' \Downarrow (V), l, m'' \) \( U \oplus V = W \) where \( \oplus \in \{ +, *, - \} \) and \( U, V \in N \)

Arithmetic Relations: \( (E), l, m \Downarrow (U), l, m' \) \( (E'), l, m' \Downarrow (V), l, m'' \) \( U \sim V = \text{true} \)

\( (E), l, m \Downarrow (U), l, m' \) \( (E'), l, m' \Downarrow (V), l, m'' \) \( U \sim V = \text{false} \)

Boolean Expressions: \( \text{where } \sim \in \{ =, < \} \)

\( (E), l, m \Downarrow (0), l, m' \)
\( (E \& \& E'), l, m \Downarrow (0), l, m' \)
\( (E), l, m \Downarrow (V), l, m' \) \( V \neq 0 \)
\( (E'), l, m' \Downarrow (0), l, m'' \)
\( (E \| E'), l, m \Downarrow (1), l, m' \)
\( (E), l, m \Downarrow (0), l, m' \)
\( (E'), l, m' \Downarrow (0), l, m'' \)

L-values:

\( \text{lvalue}(I), l, m = \text{ref}(l(I)) \) if \( l(I) \) exists
\( \text{lvalue}(M), l, m = \text{ref}(i) \) \( m(i) = \text{ref}(j) \)
\( \text{lvalue}(*M), l, m = \text{ref}(j) \)

Assignment:

\( (E), l, m \Downarrow (V), l, m' \) \( \text{lvalue}(M), l, m = \text{ref}(i) \)
\( M = E, l, m \Downarrow (V), l, m'[i \leftarrow V] \) if \( \text{lvalue}(M), l, m \) exists

Function Application:

\( m(I) = \text{closure}(Is, Blk, l') \)
\( \forall i = 1, \ldots, n. (E_{i+1}, l, m_i) \Downarrow (V_{i+1}, l, m_{i+1}) \)
\( (l', m') = \text{match}(Is, (V_1, \ldots, V_n), l, m_n) \)
\( Blk, l', m \Downarrow Blk (V_i, l', m') \)
\( (I(E_1, \ldots, E_n), l, m_0) \Downarrow (Vi, l, m) \)
Blocks (Blk):

Empty Block: \((\{\}, l, m) \downarrow (\bullet, l, m)\)

Block Sequence: 
\[
\frac{(Prog, l, m) \downarrow (\text{rtn}(Vi), l', m')}{(\{Prog\}, l, m) \downarrow (Vi, l, m')} \quad \frac{(Prog, l, m) \downarrow (\bullet, l', m')}{(\{Prog\}, l, m) \downarrow (\bullet, l, m')}
\]

Statement Sequences (StmtList):

Single Statement: 
\[
\frac{(Stmt, l, m) \downarrow_{Stmt} (Vi, l, m')}{(Stmt, l, m) \downarrow_{StmtList} (Vi, l, m')}
\]

Statement Sequence: 
\[
\frac{(Stmt, l, m) \downarrow (\bullet, l, m') \quad (StmtList, l, m') \downarrow (Vi, l, m'')}{(Stmt \downarrow_{StmtList, l, m} (Vi, l, m'')}
\]

\[
\frac{(Stmt, l, m) \downarrow (\text{rtn}(Vi), l, m')}{(Stmt \downarrow_{StmtList, l, m} \downarrow (\text{rtn}(Vi), l, m')}
\]

Statements (Stmt):

Expressions: 
\[
\frac{(E, l, m) \downarrow (V, l, m')}{(E, l, m) \downarrow (\bullet, l, m')}
\]

Block: 
\[
\frac{(Blk, l, m) \downarrow_{Blk} (Vi, l, m')}{(Blk, l, m) \downarrow_{Stmt} (Vi, l, m')}
\]

If-true: 
\[
\frac{(E, l, m) \downarrow (V, l, m') \quad V \neq 0 \quad (Stmt, l, m') \downarrow (Vi, l, m'')}{(if \ E \ then \ Stmt \ else \ Stmt' \ fi, l, m') \downarrow (Vi, l, m'')}
\]

If-false: 
\[
\frac{(E, l, m) \downarrow (0, l, m') \quad (Stmt', l, m') \downarrow (Vi, l, m'')}{(if \ E \ then \ Stmt \ else \ Stmt' \ fi, l, m') \downarrow (Vi, l, m'')}
\]

While-false: 
\[
(E, l, m) \downarrow (0, l, m')
\]

While-true: 
\[
\frac{(E, l, m) \downarrow (V, l, m') \quad V \neq 0 \quad (Stmt, l, m') \downarrow (\bullet, l, m'') \quad (while \ E \ do \ Stmt \ od \ , l, m) \downarrow (Vi, l, m''')}{(while \ E \ do \ Stmt \ od \ , l, m) \downarrow (Vi, l, m''')}
\]

\[
\frac{(E, l, m) \downarrow (V, l, m') \quad V \neq 0 \quad (Stmt, l, m') \downarrow (\text{rtn}(Vi), l, m'') \quad (while \ E \ do \ Stmt \ od \ , l, m) \downarrow (\text{rtn}(Vi), l, m'')}{(while \ E \ do \ Stmt \ od \ , l, m) \downarrow (\text{rtn}(Vi), l, m'')}
\]

Return: 
\[
\frac{(E, l, m) \downarrow (V, l, m')}{(\text{return} \ E, l, m) \downarrow (\text{rtn}(Vi), l, m')}
\]

Declarations (Decl):

(int \( I, l, m) \downarrow (I[i \leftarrow i], m) \quad i \text{ fresh for } (l, m)\)

(int \( I = N, l, m) \downarrow (I[i \leftarrow i], m[i \leftarrow N]) \quad i \text{ fresh for } (l, m)\)
\[
\begin{align*}
\text{(int } M, l, m) & \triangleright (l, m') \quad \text{value}(M, l, m') = \text{ref}(i) \\
\text{ (int } * M, l, m) & \triangleright (l, m'[i \leftarrow \text{ref}(j)]) \quad j \text{ fresh for } (l, m') \\
\text{ (int } M, l, m) & \triangleright (l, m') \quad \text{value}(M, l, m') = \text{ref}(i) \\
\text{ (int } * M = V, l, m) & \triangleright (l, m'[i \leftarrow \text{ref}(j); j \leftarrow V]) \quad j \text{ fresh for } (l, m') \\
\text{ (int } I (Is) \text{ Blk}; l, m) & \triangleright (l[I \leftarrow i], m[i \leftarrow \text{closure}(Is, \text{Blk}, l)]) \quad i \text{ fresh for } (l, m)
\end{align*}
\]

Programs:

Statement: \[
\frac{(StmtList, l, m) \triangleright StmtList (Vi, l, m')}{(StmtList, l, m) \triangleright (Vi, l, m')}
\]

Declaration and Program: \[
\frac{(Decl, l, m) \triangleright (l', m') \quad (Prog, l', m') \triangleright (Vi, l'', m'')}{(Decl \text{ Prog}, l, m) \triangleright (Vi, l'', m'')}
\]

5 Problems

1. In the file mp3.k define a module MP3-SYNTAX giving the syntax for IMP3 (borrowing heavily from previous assignments), and a module MP3 giving the semantics for IMP3, consistent with the syntax and semantics given above, in that it must associate the same values to variables. You may copy as much of the syntax and semantics of files found on the website as you find useful.