MP 1 – Implementing a Simple Imperative Programming Language in K

CS 422 – Spring 2016
Revision 1.0

Assigned February 22, 2016
Due February 29, 2016, 8:00pm
Extension 24 hours (10% penalty)

1 Change Log

1.0 Initial Release.

2 Turn-In Procedure

Put you code as plain text for this MP in a file named `mp1.k`, and submit your plain text file `mp1.k` by first adding it to your svn repository directory `assignments/mp1`, which may be done using the command
(`svn add mp1.k`) and then committing it using
(`svn commit -m "submitting mp1" mp1.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 `mp1.k` and committed in your `assignments/mp1` directory.

3 Objectives

The purpose of this MP is to familiarize you with using K to specify a simple imperative programming language.

4 Background

In class, we have looked at a simple imperative programming language SIMPL1 and how to specify it is several semantic frameworks, including the K specification language. In this assignment you will be asked to specify a similar imperative programming language in K. We will begin with a review of the syntax and Natural Semantics of SIMPL1, followed by it specification in K. Then you will be given the syntax and Natural Semantics for a nearly comparable fragment of C and asked to specify it in K.

4.1 Syntax of SIMPL1

In class, we worked with specifying in K the language SIMPL1 whose syntax is given by the BNF Grammar below:

$$
\begin{array}{lcl}
I & \in & \textit{Identifiers} \\
N & \in & \textit{Numerals} \\
E & ::= & N \mid I \mid E + E \mid E * E \mid E - E \\
B & ::= & \mathsf{true} \mid \mathsf{false} \mid B \& B \mid B \ \mathsf{or}\ B \mid \mathsf{not}\ B \mid E < E \mid E = E \\
C & ::= & \mathsf{skip} \mid C; C \mid \{C\} \mid I := E \mid \mathsf{if}\ B\ \mathsf{then}\ C\ \mathsf{else}\ C\ \mathsf{fi} \mid \mathsf{while}\ B\ \mathsf{do}\ C\ \mathsf{od}
\end{array}
$$

4.2 Natural Semantics for SIMPL1

Assuming a set Values of final results of expressions (in this case you can assume integers), and $m,\ m'$: Identifiers $\rightharpoonup$ Values, recall the Natural Semantics we gave for the SIMPL1 as follows:

Constants:

Identifiers: $(I, m) \Downarrow m(I)$ if $m(I)$ exists Numerals are values: $(N, m) \Downarrow N$

Booleans: $(\mathsf{true}, m) \Downarrow \mathsf{true}$ $(\mathsf{false}, m) \Downarrow \mathsf{false}$

Arithmetic Expressions:

$$\frac{(E, m) \Downarrow U \quad (E', m) \Downarrow V \quad U \oplus V = N}{(E \oplus E', m) \Downarrow N} \text{ where } \oplus \in \{+, *, -\} \text{ and } U, V \in \texttt{Values}$$

Arithmetic Relations:

$$\frac{(E, m) \Downarrow U \quad (E', m) \Downarrow V \quad U \sim V = b}{(E \sim E', m) \Downarrow b} \text{ where } \sim \in \{=, <\}$$

Boolean Expressions:

$$\frac{(B, m) \Downarrow \mathsf{false}}{(B \& B', m) \Downarrow \mathsf{false}} \qquad \frac{(B, m) \Downarrow \mathsf{false} \quad (B', m) \Downarrow b}{(B \text{ or } B', m) \Downarrow b} \qquad \frac{(B, m) \Downarrow \mathsf{false}}{(\mathsf{not}\ B, m) \Downarrow \mathsf{true}}$$

$$\frac{(B, m) \Downarrow \mathsf{true} \quad (B', m) \Downarrow b}{(B \& B', m) \Downarrow b} \qquad \frac{(B, m) \Downarrow \mathsf{true}}{(B \text{ or } B', m) \Downarrow \mathsf{true}} \qquad \frac{(B, m) \Downarrow \mathsf{true}}{(\mathsf{not}\ B, m) \Downarrow \mathsf{false}}$$

Commands:

Assignment: $\dfrac{(E, m) \Downarrow V}{(I := E, m) \Downarrow m[I \leftarrow V]}$ where $m[I \leftarrow V](J) = \begin{cases} V & \text{if } J = I \\ m(J) & \text{otherwise} \end{cases}$

Skip: $(\mathsf{skip}, m) \Downarrow m$ Sequencing: $\dfrac{(C, m) \Downarrow m' \quad (C', m') \Downarrow m''}{(C; C', m) \Downarrow m''}$ Block: $\dfrac{(C, m) \Downarrow m'}{(\{C\}, m) \Downarrow m'}$

If-true: $\dfrac{(B, m) \Downarrow \mathsf{true} \quad (C, m) \Downarrow m'}{(\mathsf{if}\ B\ \mathsf{then}\ C\ \mathsf{else}\ C'\ \mathsf{fi}, m) \Downarrow m'}$ If-false: $\dfrac{(B, m) \Downarrow \mathsf{false} \quad (C', m) \Downarrow m'}{(\mathsf{if}\ B\ \mathsf{then}\ C\ \mathsf{else}\ C'\ \mathsf{fi}, m) \Downarrow m'}$

While-false: $\dfrac{(B, m) \Downarrow \mathsf{false}}{(\mathsf{while}\ B\ \mathsf{do}\ C\ \mathsf{od}\ , m) \Downarrow m}$

While-true: $\dfrac{(B, m) \Downarrow \mathsf{true} \ \ (C, m) \Downarrow m' \ \ (\mathsf{while}\ B\ \mathsf{do}\ C\ \mathsf{od}\ , m') \Downarrow m''}{(\mathsf{while}\ B\ \mathsf{do}\ C\ \mathsf{od}\ , m) \Downarrow m''}$

4.3 Syntax of IMP1

The following is a BNF grammar for the language you are to specify in K:

$$
\begin{array}{rcl}
I & \in & \textit{Identifiers} \\
N & \in & \textit{Numerals} \\
E & ::= & N \mid I \mid (E) \mid E * E \mid E + E \mid E - E \\
 & & E< E \mid E{==}E \mid E \,\&\&\, E \mid E \,\|\, E \mid \,!\, E \mid I = E \\
Blk & ::= & \{\,\} \mid \{\, StmtList \,\} \\
StmtList & ::= & Stmt \mid Stmt\ Stmtlist \\
Stmt & ::= & E; \mid Blk \mid \textsf{if (}\ E\ \textsf{)}\ Stmt\ \textsf{else}\ Stmt \mid \textsf{while (}\ E\ \textsf{)}\ Stmt \\
Decl & ::= & \textsf{int}\ I\ \textsf{=}\ N\textsf{;} \\
DeclList & ::= & Decl \mid Decl\ DeclList \\
Prog & ::= & Stmt \mid DeclList\ Stmt
\end{array}
$$

The unary operator ! 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 IMP1

Assuming a set Values of final results of expressions (in this case you can assume integers), and $m,\ m'$: Identifiers $\rightharpoonup$ Values, the Natural Semantics for the IMP1 as follows:

Constants: Identifiers: $(I, m) \Downarrow (m(I), m)$ if $m(I)$ exists Numerals are values: $(N, m) \Downarrow (N, m)$

Parentheses: $$\frac{((E), m) \Downarrow (V, m')}{(E, m) \Downarrow (V, m')}$$

Arithmetic Expressions:

$$\frac{(E, m) \Downarrow (U, m') \quad (E', m') \Downarrow (V, m'') \quad U \oplus V = N}{(E \oplus E', m) \Downarrow (N, M'')}$$ where $\oplus \in \{+, *, -\}$ and $U, V \in \texttt{Values}$

Arithmetic Relations:

$$\frac{(E, m) \Downarrow (U, M') \quad (E', m') \Downarrow (V, m'') \quad U \sim V = \textsf{true}}{(E \sim E', m) \Downarrow (1, m'')} \qquad \frac{(E, m) \Downarrow (U, m') \quad (E', m') \Downarrow (V, m'') \quad U \sim V = \textsf{false}}{(E \sim E', m) \Downarrow (0, m'')}$$

where $\sim \in \{=, <\}$

Boolean Expressions:

$$\frac{(E, m) \Downarrow (0, m')}{(E \&\& E', m) \Downarrow (0, m')} \qquad \frac{(E, m) \Downarrow (V, m') \quad V \neq 0 \quad (E', m') \Downarrow (0, m'')}{(E \&\& E', m) \Downarrow (0, m'')}$$

$$\frac{(E, m) \Downarrow (U, m') \quad U \neq 0 \quad (E', m') \Downarrow (V, m'') \quad V \neq 0}{(E \&\& E', m) \Downarrow (1, m'')}$$

$$\frac{(E, m) \Downarrow (V, m') \quad V \neq 0}{(E \,\|\, E', m) \Downarrow (1, m')} \qquad \frac{(E, m) \Downarrow (0, m') \quad (E', m') \Downarrow (V, m'') \quad V \neq 0}{(E \,\|\, E', m) \Downarrow (1, m'')} \qquad \frac{(E, m) \Downarrow (0, m') \quad (E', m') \Downarrow (0, m'')}{(E \,\|\, E', m) \Downarrow (0, m'')}$$

$$\frac{(E,m) \Downarrow (0,m')}{(!E,m) \Downarrow (1,m')} \qquad \frac{(E,m) \Downarrow (V,m') \quad V \neq 0}{(!E,m) \Downarrow (0,m')}$$

Assignment:

$$\frac{m(I) = U \quad (E,m) \Downarrow (V,m')}{(I := E,m) \Downarrow (V, m'[I \leftarrow V])} \text{ where } m'[I \leftarrow V](J) = \begin{cases} V & \text{if } J = I \\ m'(J) & \text{otherwise} \end{cases}$$

Blocks:

$$\text{Empty Block:} \quad (\{\ \},m) \Downarrow m \qquad \text{Block Sequence:} \quad \frac{(StmtList,m) \Downarrow m'}{(\{StmtList\},m) \Downarrow m'}$$

Statement Sequences:

$$\text{Single Statement:} \quad \frac{(Stmt,m) \Downarrow m'}{(Stmt,m) \Downarrow m'} \qquad \text{Statement Sequence:} \quad \frac{(Stmt,m) \Downarrow m' \quad (StmtList,m') \Downarrow m''}{(Stmt\ StmtList,m) \Downarrow m''}$$

Statements:

$$\text{Expressions:} \quad \frac{(E,m) \Downarrow (V,m')}{(E,m) \Downarrow m'} \qquad \text{Block:} \quad \frac{(Blk,m) \Downarrow m'}{(Blk,m) \Downarrow m'}$$

$$\text{If-true:} \quad \frac{(E,m) \Downarrow (V,m') \quad V \neq 0 \quad (Stmt,m') \Downarrow m''}{(\textsf{if } E \textsf{ then } Stmt \textsf{ else } Stmt' \textsf{ fi}, m') \Downarrow m'} \qquad \text{If-false:} \quad \frac{(E,m) \Downarrow (0,m') \quad (Stmt',m') \Downarrow m''}{(\textsf{if } E \textsf{ then } Stmt \textsf{ else } Stmt' \textsf{ fi}, m) \Downarrow m''}$$

$$\text{While-false:} \quad \frac{(E,m) \Downarrow (0,m')}{(\textsf{while } E \textsf{ do } Stmt \textsf{ od }, m) \Downarrow m'}$$

$$\text{While-true:} \quad \frac{(E,m) \Downarrow (V,m') \quad V \neq 0 \quad (Stmt,m') \Downarrow m'' \quad (\textsf{while } E \textsf{ do } Stmt \textsf{ od }, m'') \Downarrow m'''}{(\textsf{while } E \textsf{ do } Stmt \textsf{ od }, m) \Downarrow m'''}$$

Declarations: : $(\textsf{int } I = N, m) \Downarrow m[I \leftarrow N]$

Declaration Lists:

$$\text{Single Declaration:} \quad \frac{(Decl,m) \Downarrow m'}{(Decl,m) \Downarrow m'} \qquad \text{Declaration List:} \quad \frac{(Decl,m) \Downarrow m' \quad (DeclList,m') \Downarrow m''}{(Decl\ DeclList,m) \Downarrow m''}$$

Programs:

$$\text{Statement:} \quad \frac{(Stmt,\{\ \}) \Downarrow m}{(Stmt,\{\ \}) \Downarrow m} \qquad \text{Declaration List and Statement:} \quad \frac{(DeclList,\{\ \}) \Downarrow m \quad (Stmt,m) \Downarrow m'}{(DeclList\ Stmt,\{\ \}) \Downarrow m'}$$

5 Problems

1. In the file `mp1.k` define a module `MP1-SYNTAX` giving the syntax for IMP, and a module `MP1` giving the semantics for IMP, consistent with the syntax and semantics given above. You may copy as much of the syntax and semantics of SIMPL1 as you find useful.

   To test your specification, first build an interpreter from it using

```
kompile mp1.k
```

You can then try the interpreter on test program found in `fact.mp1` by

```
krun fact.mp1
```

The result should be something like

```
<T> <k> . </k> <mem> x |-> 5 r |-> 120 i |-> 5 </mem> </T>
```

6 Words of Advise

You will need to make use of `requires` and various boolean conditions using `==Int`, `<Int`, and `notBool` for specifying the "Boolean" expressions in IMP1. Remember that they are actually computing integers.

You should do you development in stages. When you have a stage compiling, add in a module to test it and make some test files for yourself. After you are satisfied with a given stage comment out an code that was specific to the testing. K4.0 seems to have some bugs with importing that causes it to pick up more than it should. Also, if you `kompile` other .k files in the mp1 directory, after testing them out, delete (`rm -rf`) the `-kompiled` directory it created. You can only have one `-kompiled` in the directory in which you run `krun` on an example.

You will probably need to make some transitions where part or all of the entity transitions to . (period). This is the empty list. It is a list and can not be put into a HOLE that is not of list type. The `strict` attributes generate transitions creating and filling HOLEs of the types of the nonterminals in the syntax production annotated with them.

As you are correcting code that either did not compile or computed examples incorrectly, if recompilation does not seem to reflect the changes you have made, delete (`rm -fr`) `mp1-kompiled`. K4.0 (and maybe 3.6) work a little too hard at reusing already compiled code fragments.

Don't waste time going round in circles trying to debug your specifications. Get help after a few attempts or when you are clearly confused. You can also look at the examples in the `k/tutorial/1_k/2_imp` directory in the K distribution. It should be up-to-date for the current version of K. The language done there is similar to, but not the same as, the language I have given you here. Be sure you don't accidentally give me that version instead of the version I have requested.

K is a very useful formalism for expressing programming language features, but K4.0 is not a production quality tool. If you are having difficulty with behavior you can not understand, please ask. You may need extra clarification of the meaning of some K constructs, or you may have encountered a bug in the system. So far, the bugs have been somewhat annoying, but also easily enough avoided. I have tried to distill the wisdom of my experiences above, but if you are having a problem, please ask.