CSCI 5535, ECEN 5533: Homework 2

Due Thursday, September 8, 2011

Exercise 1: Bookkeeping. Indicate in a sentence or two how much time
you spent on this homework, how difficult you found it subjectively, and
what you found to be the hardest part. Additionally, if your opinions have
changed since the last homework, indicate one thing you like about the class
so far and one thing you would change about it.

Exercise 2: Structural Small-Step Operational Semantics. In class,
we defined a small-step operational semantics for IMP. We review the reduc-
tion rules in the following, first reviewing the rules for arithmetic expressions.

\[ a \rightarrow_{\sigma} a' \]

\[ x \rightarrow_{\sigma} \sigma(x) \]

\[ n_1 + n_2 \rightarrow_{\sigma} n_1 + n_2 \]

\[ a_1 \rightarrow_{\sigma} a_1' \]

\[ a_1 + a_2 \rightarrow_{\sigma} a_1' + a_2 \]

\[ a_2 \rightarrow_{\sigma} a_2' \]

\[ n_1 + a_2 \rightarrow_{\sigma} n_1 + a_2' \]

The rules for subtraction and multiplication are similar. Next, we review the
rules for Boolean expressions.

\[ b \rightarrow_{\sigma} b' \]

\[ n = n \rightarrow_{\sigma} \text{true} \]

\[ n_1 \neq n_2 \rightarrow_{\sigma} \text{false} \]

\[ a_1 \rightarrow_{\sigma} a_1' \]

\[ a_1 = a_2 \rightarrow_{\sigma} a_1' = a_2 \]

\[ a_2 \rightarrow_{\sigma} a_2' \]

\[ n_1 = a_2 \rightarrow_{\sigma} n_1 = a_2' \]

\[ \text{not true} \rightarrow_{\sigma} \text{false} \]

\[ \text{not false} \rightarrow_{\sigma} \text{true} \]

\[ b \rightarrow_{\sigma} b' \]

\[ \text{not } b \rightarrow_{\sigma} \text{not } b' \]
The reduction rules for logical \texttt{and} and \texttt{or} are up to you to define. More about this later.

Next we have the reduction rules for commands.

\[
\begin{align*}
\langle c, \sigma \rangle & \rightarrow \langle c', \sigma' \rangle \\
\langle x := n, \sigma \rangle & \rightarrow \langle \text{skip}, \sigma[x := n] \rangle \\
\langle \text{skip}; c, \sigma \rangle & \rightarrow \langle c, \sigma \rangle \\
\langle \text{if} \ true \ \text{then} \ c_1 \ \text{else} \ c_2, \sigma \rangle & \rightarrow \langle c_1, \sigma \rangle \\
\langle \text{if} \ false \ \text{then} \ c_1 \ \text{else} \ c_2, \sigma \rangle & \rightarrow \langle c_2, \sigma \rangle \\
\langle \text{while} \ b \ \text{do} \ c, \sigma \rangle & \rightarrow \langle \text{if} \ b \ \text{then} \ c; \ \text{while} \ b \ \text{do} \ c \ \text{else} \ \text{skip}, \sigma \rangle \\
\langle x := e, \sigma \rangle & \rightarrow \langle x := e', \sigma \rangle \\
\langle c_1; c_2, \sigma \rangle & \rightarrow \langle c'_1; c_2', \sigma' \rangle \\
\langle c_1, \sigma \rangle & \rightarrow \langle c'_1, \sigma' \rangle \\
\langle b, \sigma \rangle & \rightarrow \langle b', \sigma \rangle \\
\langle \text{if} \ b \ \text{then} \ c_1 \ \text{else} \ c_2, \sigma \rangle & \rightarrow \langle \text{if} \ b' \ \text{then} \ c_1 \ \text{else} \ c_2, \sigma \rangle
\end{align*}
\]

- Extend IMP with the \texttt{let} command (as we did in homework 1) and with commands that perform input, output, parallel composition, and mutex locking and unlocking.

\[
c ::= \ldots \mid \text{let} \ x = a \ \text{in} \ c \mid \text{print} \ a \mid \text{input} \ x \mid c || c \mid \text{lock} \ x \mid \text{unlock} \ x
\]

Define small-step reduction rules for these new constructs and, if necessary, give updated rules for the original constructs of IMP. The input command reads an integer whereas the print command outputs an integer. The parallel composition command executes the two commands in parallel, in a sequentially consistent manor in which all “leaf” commands (commands that do not include sub-commands) are executed atomically. The lock command attempts to acquire the lock of the given name. If that lock is already taken, then the command blocks. If the lock is not already taken, then the lock is acquired and execution proceeds to the next command. The unlock command releases the lock, if it had been acquired, and proceeds to the next command.

- Define the small-step reduction rules for logical \texttt{and} and \texttt{or}. Does it matter whether the reduction rules perform short-circuiting or not? If
it matters, give an example that demonstrates that it matters, and define the short-circuiting version of the rules.

- Download the Homework 2 code pack from the course web page. The README.txt describes the code pack, like in Homework 1. Modify hw2.ml so that it implements a complete interpreter for IMP with the extensions described above. The Makefile includes a “make test” target that you should use (at least) to test your work.

- Modify the file example.imp so that it contains a “tricky” IMP command (presumably involving exceptions) that can be parsed by our IMP test harness (e.g., “imp < example.imp” should not yield a parse error).

- Rename hw2.ml to your_identkey-hw2.ml and rename example.imp to your_identkey-example.imp for submission. Do not modify any other files. Your submission’s grade will be based on how many of the submitted example.imps it interprets correctly (in a manner just like the “make test” trials). If your submitted example.imp breaks the greatest number of interpreters (and more than 0!), you will receive extra credit. If there is a tie, all those in the tie will receive the extra credit.

- Make sure your code compiles. Code that does not compile will not be graded. If there is some case you cannot get to work, simply comment it out.