Practice Questions for Quiz 3

Scope.
- big-step operational semantics
- small-step operational semantics
- states, configurations, stuck configurations, terminal configurations, divergent configurations
- transition systems

Themes of the quiz.
- Can you follow the rules?
- Can you interpret a new (big-step/small-step) rule?
- Can you formulate a new (big-step/small-step) rule for a new programming construct?

Question 1 (4 points).

The rules for the small-step semantics of LC are in Appendix 3 on page 5.

YOUR PROBLEM: Give a derivation of the small-step transition:

\[
(9 + (\ell \times 11), s) \rightarrow (9 + ((3 - 1) \times 11), s)
\]

where \( s \) is the state \( \{ \ell \rightarrow 3 \} \). Label each application of a rule.

Question 2 (5 points).

The small-step semantic rules for LC are given in Appendix 3 on page 3.

Notation: \( s^{m,n} \) denotes the state \( \{ \ell \rightarrow m, \ell' \rightarrow n \} \). E.g., \( s^{5,12} = \{ \ell \rightarrow 5, \ell' \rightarrow 12 \} \).

YOUR PROBLEM: Give a small-step semantic derivation of

\[
\left\{ \ell : = 10; \text{ if } (\ell > \ell' \text{ then } \ell' : = 100 \text{ else } \ell' : = 200 \right\}, s^{0,20} \rightarrow \ldots \rightarrow \langle \text{skip}, s^{10,200} \rangle
\]

Do not worry about giving a small-step derivation of each transition.

Question 3 (5 points).

The big-step semantic rules for LC are given in Appendix 4 on page 3.

Notation: \( s^m \) denotes the state \( \{ \ell \rightarrow m \} \). E.g., \( s^5 = \{ \ell \rightarrow 5 \} \).

YOUR PROBLEM: Give a big-step semantic derivation of

\[
\left\{ \ell : = 1; \text{ if } (\ell < 0 \text{ then skip else } \ell : = \ell + 2 \right\}, s^{10} \downarrow \langle \text{skip}, s^3 \rangle
\]

Label each rule application.

Question 4 (6 points).

Background: The programming language Marvin has the grammar:

\[
\begin{align*}
C & ::= \text{skip} \mid ++R \mid --R \mid \text{iter}(R : C) \mid C;C \\
R & ::= M \mid N
\end{align*}
\]

Marvin-programs run on a machine with a two integer registers, named M and N. The informal semantics of Marvin-commands are:

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>skip</td>
<td>No-op: the registers are unchanged</td>
</tr>
<tr>
<td>++R</td>
<td>Increment: add 1 to register R (with no change to the other register)</td>
</tr>
<tr>
<td>--R</td>
<td>Decrement: subtract 1 from register R (with no change to the other register)</td>
</tr>
<tr>
<td>iter(R : C)</td>
<td>Iterate: equivalent to “while(R \neq 0) do C”</td>
</tr>
<tr>
<td>C&lt;sub&gt;1&lt;/sub&gt;; C&lt;sub&gt;2&lt;/sub&gt;</td>
<td>Sequencing: change the counters as directed by C&lt;sub&gt;1&lt;/sub&gt; and then change those values as directed by C&lt;sub&gt;2&lt;/sub&gt;</td>
</tr>
</tbody>
</table>

Here is a sample Marvin program:

\[ \text{iter}(M : --M; ++N); ++M \]

We want to consider giving a big-step operational semantics for Marvin. A state is a pair of integers \((m, n)\), where \(m\) is the value of register M and \(n\) is the value of register N. The rule format is:

\[
\text{Name: } \langle C, (m, n) \rangle \downarrow \langle \text{skip}, (m', n') \rangle \quad \text{(side conditions, if any)}
\]

Example: The ++ command has the rule:

\[
\begin{align*}
\text{Increment: } & \quad \langle ++R, (m, n) \rangle \downarrow \langle \text{skip}, (m', n') \rangle \\
& \quad \begin{cases} \text{if } R = M, & \text{then } (m', n') = (m + 1, n) \\
& \text{if } R = N, & \text{then } (m', n') = (m, n + 1) \end{cases}
\end{align*}
\]

YOUR PROBLEMS: Give (correct) big-step operational semantics rules for:

(a) (2 points) the -- command
(b) (4 points) the iter command
Appendix 1: A grammar for LC

<table>
<thead>
<tr>
<th>Phases</th>
<th>P ::=</th>
<th>C</th>
<th>E</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>Commands</td>
<td>C ::=</td>
<td>skip</td>
<td>ℓ := E</td>
<td>C</td>
</tr>
<tr>
<td></td>
<td></td>
<td>if B then C else C</td>
<td>while B do C</td>
<td></td>
</tr>
<tr>
<td>Integer Expressions</td>
<td>E ::=</td>
<td>n</td>
<td>ℓ</td>
<td>E</td>
</tr>
<tr>
<td>Boolean Expressions</td>
<td>B ::=</td>
<td>b</td>
<td>E</td>
<td>E</td>
</tr>
<tr>
<td>Integers</td>
<td>n ∈ Z = { . . . , −3, −2, −1, 0, 1, 2, 3, . . .}</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Booleans</td>
<td>b ∈ B = {true, false}</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Locations</td>
<td>ℓ ∈ L = {ℓ₀, ℓ₁, ℓ₂, . . .}</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

If ℓ is the integer currently stored in ℓ

Appendix 2: Notation for modifying states

s[ℓ ↦ k] is a modification of state s such that:

(s[ℓ ↦ k])(ℓ′) = \begin{cases} k, & \text{if } ℓ = ℓ' \\ s(ℓ'), & \text{if } ℓ ≠ ℓ' \end{cases}

E.g., for s = { ℓ → 12, ℓ' → 3, ℓ'' → 9 }: s[ℓ' → 20] = { ℓ → 12, ℓ' → 20, ℓ'' → 9 }

Appendix 3: A small-step semantics for LC

→-op₁: ⟨E₁, s⟩ → ⟨E₁', s'⟩
→-op₂: ⟨E₂, s⟩ → ⟨E₂', s'⟩
→-op₃: ⟨n₁ ⊕ n₂, s⟩ → ⟨c, s⟩ (c = n₁ ⊕ n₂)
→-loc: ⟨M, s⟩ → ⟨s(ℓ), s⟩ (ℓ ∈ dom(s))
→-set₁: ⟨ℓ := E, s⟩ → ⟨ℓ := E', s'⟩
→-set₂: ⟨ℓ := n, s⟩ → ⟨skip, s[ℓ → n]⟩
→-seq₁: ⟨C₁, s⟩ → ⟨C₁', s'⟩
→-seq₂: ⟨skip; C, s⟩ → ⟨C, s⟩
→-while: ⟨while B do C, s⟩ → ⟨if B then {C; while B do C} else skip, s⟩
→-if₁: ⟨B, s⟩ → ⟨B', s'⟩
→-if₂: ⟨if B then C₁ else C₂, s⟩ → ⟨if B' then C₁ else C₂, s⟩
→-if₃: ⟨if false then C₁ else C₂, s⟩ → ⟨C₁, s⟩

Appendix 4: A big-step semantics for LC

⇓-Con: ⟨c, s⟩ ⇓ ⟨c, s⟩ (c ∈ Z ∪ B)
⇓-⊕: ⟨E₁, s⟩ ⇓ ⟨n₁, s'⟩, ⟨E₂, s⟩ ⇓ ⟨n₂, s''⟩ (c = n₁ ⊕ n₂)
⇓-Set: ⟨ℓ := E, s⟩ ⇓ ⟨ℓ := E', s'[ℓ → ℓ]⟩
⇓-Seq: ⟨C₁, s⟩ ⇓ ⟨skip, s'⟩, ⟨C₂, s⟩ ⇓ ⟨skip, s'⟩, ⟨C₁; C₂, s⟩ ⇓ ⟨skip, s''⟩
⇓-If₁: ⟨B, s⟩ ⇓ ⟨false, s'⟩, ⟨C₁, s⟩ ⇓ ⟨skip, s''⟩, ⟨if B then C₁ else C₂, s⟩ ⇓ ⟨skip, s''⟩
⇓-If₂: ⟨B, s⟩ ⇓ ⟨false, s'⟩, ⟨C₂, s⟩ ⇓ ⟨skip, s''⟩, ⟨if B then C₁ else C₂, s⟩ ⇓ ⟨skip, s''⟩
⇓-While₁: ⟨B, s⟩ ⇓ ⟨false, s'⟩, ⟨C₁, s⟩ ⇓ ⟨skip, s''⟩, ⟨while B do C, s''⟩ ⇓ ⟨skip, s'''⟩
⇓-While₂: ⟨B, s⟩ ⇓ ⟨false, s'⟩, ⟨while B do C, s⟩ ⇓ ⟨skip, s''⟩
Practice Questions for Quiz 3

An Answer for 1

\[
\begin{align*}
\text{loc} & \frac{1}{1} \langle !\ell, s \rangle \rightarrow \langle 3, s \rangle \\
*_{1} & \frac{1}{(\ell - 1) \times 11, s} \rightarrow \langle 3 - 1, s \rangle \\
+_{2} & \frac{1}{9 + ((\ell - 1) \times 11), s} \rightarrow \langle 9 + ((3 - 1) \times 11), s \rangle
\end{align*}
\]

An Answer for 2

\[
\begin{align*}
\langle \ell : = 10; \text{if} (\ell > \ell') \text{ then } \ell' : = 100 \text{ else } \ell' : = 200 \rangle, s^{0,20} \\
\rightarrow \langle \text{skip; if} (\ell > \ell') \text{ then } \ell' : = 100 \text{ else } \ell' : = 200 \rangle, s^{10,20} \\
\rightarrow \langle \text{if} (\ell > \ell') \text{ then } \ell' : = 100 \text{ else } \ell' : = 200 \rangle, s^{10,20} \\
\rightarrow \langle \text{if} (10 > \ell') \text{ then } \ell' : = 100 \text{ else } \ell' : = 200 \rangle, s^{10,20} \\
\rightarrow \langle \text{if} (10 > 20) \text{ then } \ell' : = 100 \text{ else } \ell' : = 200 \rangle, s^{10,20} \\
\rightarrow \langle \text{if} (\text{false}) \text{ then } \ell' : = 100 \text{ else } \ell' : = 200 \rangle, s^{10,20} \\
\rightarrow \langle \ell' : = 200, s^{10,20} \rangle \\
\rightarrow \langle \text{skip, } s^{10,200} \rangle
\end{align*}
\]

Aside. Problem 4 is based on a famous theorem by Marvin Minsky that the two-counter machine model is Turing complete. This is a handy thing to know since any machine model that is at least as powerful as two-counter machines (a fairly modest requirement) is also going to be Turing complete. For details, see:

http://en.wikipedia.org/wiki/Counter_machine

An Answer for 4:

Decrement: \( \langle \neg (R), (m, n) \rangle \downarrow \langle \text{skip}, (m', n') \rangle \) 

\( \text{Iterate}_0 : \langle \text{iter}(R : C), (m, n) \rangle \downarrow \langle \text{skip}, (m, n) \rangle \) \( ^{(*)} \)

\( \text{Iterate}_1 : \langle C, (m, n) \rangle \downarrow \langle \text{skip}, (m_1, n_1) \rangle \) \( \langle \text{iter}(R : C), (m_1, n_1) \rangle \downarrow \langle \text{skip}, (m_2, n_2) \rangle \) \( ^{(\ddagger)} \)

\( ^{(*)} \text{ if } R = M, \text{ then } (m', n') = (m - 1, n); \text{ if } R = N, \text{ then } (m', n') = (m, n - 1) \)

\( ^{(\ddagger)} R = M \text{ and } m = 0 \text{ or else } R = N, \text{ then } n = 0. \)

\( ^{(\S)} R = M \text{ and } m \neq 0 \text{ or else } R = N \text{ and } n \neq 0. \)

An Answer for 3

\[
\begin{align*}
\text{Con} & \begin{array}{c} \langle 1, s^{10} \rangle \downarrow \langle 1, s^{1} \rangle \\
\text{Set} & \langle \ell : = 1, s^{10} \rangle \downarrow \langle \text{skip}, s^{1} \rangle \\
\text{Seq} & \end{array} \\
\text{Loc} & \langle !\ell, s^{1} \rangle \downarrow \langle 1, s^{1} \rangle \\
\text{Con} & \langle 0, s^{1} \rangle \downarrow \langle 0, s^{1} \rangle \\
\text{Loc} & \langle !\ell, s^{1} \rangle \downarrow \langle 1, s^{1} \rangle \\
\text{Con} & \langle 2, s^{1} \rangle \downarrow \langle 2, s^{1} \rangle \\
\text{Set} & \langle !\ell + 2, s^{1} \rangle \downarrow \langle 3, s^{1} \rangle \\
\text{Seq} & \langle \ell : = !\ell + 2, s^{1} \rangle \downarrow \langle \text{skip}, s^{3} \rangle \\
\end{align*}
\]

\[
\begin{align*}
\langle \ell : = 1; \text{if} (\ell < 0) \text{ then skip else } \ell : = !\ell + 2 \rangle, s^{10} & \downarrow \langle \text{skip}, s^{3} \rangle
\end{align*}
\]