COMP3161/COMP9164

Abstract Machines Exercises

Liam O'Connor

October 20, 2019

1. **Decision Machines**: Suppose we have a language of nested brackets N (where ε is the empty string):

$$\frac{}{} \qquad \frac{e\ N}{} \qquad \frac{e\ N}{} \qquad \frac{e\ N}{}$$

Note that ()() is *not* a string in

We developed a simple abstract machine to check if strings are in this language. We set the states for the machine to be simply strings. Initial states are all non-empty strings, and the final state is the empty string. Then, our state transition relation is:

$$\frac{}{(e) \mapsto e}M_1 \qquad \frac{}{[e] \mapsto e}M_2 \qquad \frac{}{\langle e\rangle \mapsto e}M_3$$

(a) A machine *recognises* a language if any machine in the state corresponding to a string S will eventually reach a final state if an

i. [$\star$] Show that the s … hine reaches a final state given the s

Solution: The string is in the language, as shown:

$$\cfrac{\cfrac{\cfrac{\overline{\varepsilon\ N}\,N}{\langle\rangle\ N}}{[\langle\rangle]\ N}N_4}{([\langle\rangle])\ N}N_2$$

The machine derivation is simply:

	$([\langle\rangle])$	
$\mapsto$	$[\langle\rangle]$	(M_1)
$\mapsto$	$\langle\rangle$	(M_2)
$\mapsto$	ε	(M_3)

ii. [$\star\star$] Show that the string $[]()[]$ is not in the language N, and show that our machine reaches a non-final state with no outgoing transitions given the same string, i.e, there exists some stuck state s such that $[]()[] \overset{\star}{\mapsto} s$

Solution: If we attempt to derive $[]()[]\ N$:

$$\cfrac{\cfrac{???}{]()[\ N}}{[]()[]\ N}N_4$$

We get the subgoal $]()[\ N$, which is false, as all strings in N are either ε or begin with an opening bracket. Hence, as the rules are unambiguous, there is no other way to derive $[]()[]\ N$ and hence it is not in N. Similarly, our machine derivation:

	$[]()[]$	
$\mapsto$	$]()[$	(M_2)
$\mapsto$	???	

We end up in the state]()[, which is a *stuck* state, as there are no transitions from a state that begins with a closing bracket.

iii. Prove that the machine recognises the language N, that is:

α) [★★★] $s\ N \implies s \overset{\star}{\mapsto} \varepsilon$. The relation $\overset{\star}{\mapsto}$ of course being the reflexive transitive closure of $\mapsto$, that is:

$$\frac{}{s \overset{\star}{\mapsto} s}\text{REFL*} \qquad \frac{s_1 \mapsto s_2 \quad s_2 \overset{\star}{\mapsto} s_3}{s_1 \overset{\star}{\mapsto} s_3}\text{TRANS*}$$

Solution:

Base case: Where $s = \varepsilon$, we must show $\varepsilon \overset{\star}{\mapsto} \varepsilon$. We can show this using the reflexivity rule:

$$\frac{}{\overset{\star}{\mapsto}}\text{REFL*}$$

Inductive case ... is that $s' \overset{\star}{\mapsto} \varepsilon$, we must show that ...

$$\frac{\dfrac{}{(s') \mapsto s'}M_1 \quad \dfrac{}{s' \overset{\star}{\mapsto} \varepsilon}\text{I.H}}{(s') \overset{\star}{\mapsto} \varepsilon}\text{TRANS*}$$

The other inductive cases are extremely similar. □

β) [★★★] $s \overset{\star}{\mapsto} \varepsilon \implies s\ N$

Solu

Proo ... ion, which is the sa

Base case: Where the length of the execution is zero - i.e, we are already in a final state. The only final state is ε, and hence our proof go ... ch is already known from rule N_1.

Inductive case: Where our state s exec ... s'), and $s' \overset{\star}{\mapsto} \varepsilon$ $(*)$. From $(*)$ we have the inductive hypothesis $s'\ N$. We must show that $s\ N$. We proceed by case distinction on s. Seeing as $s \mapsto s'$, s must be one of (s') (by rule M_1), $[s']$ (by rule M_2), or $\langle s' \rangle$ (by rule M_3). All three cases are nearly identical, so we will deal with just the first case, where $s = (s')$.

$$\frac{\dfrac{}{s'\ N}\text{I.H}}{(s')\ N}N_2$$

□

(b) Suppose that we were unable to efficiently read from both the beginning and end of the string simultaneously (For example, if a tape or a linked list is used to represent the string). This makes our original machine highly inefficient, as each state transition must examine the end of a string for a closing bracket.

We develop a new, stack-based machine that attempts to solve this problem. Our stack consists of three symbols, P, A, and B, one for each type of bracket. The states of the machine are of the form $s \mid e$, where s is a stack and e is a string. Our initial states are all states with an empty stack and a non-empty string, i.e: $\circ \mid e$, our final state is $\circ \mid \varepsilon$, and our state transitions are as follows:

$$\frac{}{s \mid (e \mapsto \mathtt{P} \rhd s \mid e}S_1 \qquad \frac{}{s \mid \langle e \mapsto \mathtt{A} \rhd s \mid e}S_2 \qquad \frac{}{s \mid [e \mapsto \mathtt{B} \rhd s \mid e}S_3$$

$$\frac{}{\mathtt{P} \rhd s \mid)e \mapsto s \mid e}S_4 \qquad \frac{}{\mathtt{A} \rhd s \mid \rangle e \mapsto s \mid e}S_5 \qquad \frac{}{\mathtt{B} \rhd s \mid]e \mapsto s \mid e}S_6$$

i. [★] Show the execution of the new stack machine given the start state $\circ \mid [(\langle\rangle)]$.

Solution: The machine execution proceeds as follows:

$$
\begin{array}{lll}
 & \circ \mid [(\langle\rangle)] & \\
\mapsto & \mathtt{B} \rhd \circ \mid (\langle\rangle)] & (S_3) \\
\mapsto & \mathtt{P} \rhd \mathtt{B} \rhd \circ \mid \langle\rangle)] & (S_1) \\
\mapsto & \mathtt{A} \rhd \mathtt{P} \rhd \mathtt{B} \rhd \circ \mid \rangle)] & (S_2) \\
\mapsto & \mathtt{P} \rhd \mathtt{B} \rhd \circ \mid)] & (S_5) \\
\mapsto & B \rhd \circ \mid] & (S_4) \\
\mapsto & \circ \mid \varepsilon & (S_6)
\end{array}
$$

ii. Does the new machine recognise N?

α) [$\star\star\star\star$] Prove or disprove that $s\ N \implies \circ \mid s \overset{\star}{\mapsto} \circ \mid \varepsilon$ for all strings s. *Hint:* You may find it useful to prove the following lemma:

$$\frac{s_1 \overset{\star}{\mapsto} s_2 \quad s_2 \mapsto s_3}{\overset{\star}{\mapsto}}\ \textsc{Lemma}$$

Also, you may need

Solution:

Proof of Lemma. We will prove the lemma provided above first, as it will come in handy. We proceed by induction on the size of the execution $s_1 \overset{\star}{\mapsto} s_2$, and must show that, given $s_2 \mapsto s_3$ (†), that $s_1 \overset{\star}{\mapsto} s_3$.

Base case: $s_1 \overset{0}{\mapsto} s_2$, i.e $s_1 = s_2$. We must therefore show that $s_2 \overset{\star}{\mapsto} s_3$:

$$\frac{\dfrac{}{s_2 \mapsto s_3}(\dagger) \quad \dfrac{}{s_3 \overset{\star}{\mapsto} s_3}\textsc{Refl}^*}{\overset{\star}{\mapsto}}\ ^*$$

Induc... e inductive hypothesis from (

$$\frac{s_2 \mapsto s_3}{s_1' \overset{\star}{\mapsto}}$$

Then, we simply derive the proof goal

$$\frac{\dfrac{}{s_1 \mapsto s_1'}(*) \quad \dfrac{\dfrac{}{s_2 \mapsto s_3}(\dagger)}{s_1' \overset{\star}{\mapsto} s_3}\text{I.H}}{s_1 \overset{\star}{\mapsto} s_3}\ \textsc{Trans}^*$$

□

Proof of main theorem. Now that we have proven the lemma, we must now prove that $s\ N \implies \circ \mid s \overset{\star}{\mapsto} \circ \mid \varepsilon$. We will *generalise this proof goal* to make the stronger claim that $s\ N \implies t \mid sr \overset{\star}{\mapsto} t \mid r$ for any stack t and remainder string r. Note that this trivially implies our original proof goal by setting t to $\circ$ and r to ε.

Base case: Where $s = \varepsilon$, we must therefore show that $t \mid r \overset{\star}{\mapsto} t \mid r$, trivially shown by rule Refl*.

Inductive case: $s = (s')$, where $s'\ N(*)$. From $(*)$, we have the inductive hypothesis: $t' \mid s'r' \overset{\star}{\mapsto} t' \mid r'$, for any t' and r'. We must show that $t \mid (s')r \overset{\star}{\mapsto} t \mid r$ for all t, r.

$$\frac{\dfrac{}{t \mid (s')r \mapsto \mathtt{P} \rhd t \mid s')r}S_1 \quad \dfrac{\dfrac{}{\mathtt{P} \rhd t \mid s')r \overset{\star}{\mapsto} \mathtt{P} \rhd t \mid)r}\text{I.H}^1 \quad \dfrac{}{\mathtt{P} \rhd t \mid)r \mapsto t \mid r}S_4}{\mathtt{P} \rhd t \mid s')r \overset{\star}{\mapsto} t \mid r}\textsc{Lemma}}{t \mid (s')r \overset{\star}{\mapsto} t \mid r}\ \textsc{Trans}^*$$

The other inductive cases are extremely similar. □

[1]: The application of the I.H rule here sets t' to be $\mathtt{P} \rhd t$ and r' to be $)r$.

β) [$\star\star$] Prove or disprove that $\circ \mid s \overset{\star}{\mapsto} \circ \mid \varepsilon \implies s\ N$

Solution:

Counterexample. We will disprove this by way of a counterexample. It is already established that $()()$ is not in N. We will show that $\circ \mid ()() \overset{\star}{\mapsto} \circ \mid \varepsilon$ and thus there is no way that $\circ \mid s \overset{\star}{\mapsto} \circ \mid \varepsilon$ could imply $s\ N$.

The machine execution is as follows:

$$\begin{array}{lll} & \circ \mid ()() & \\ \mapsto & \mathsf{P} \triangleright \circ \mid)() & (S_1) \\ \mapsto & \circ \mid () & (S_4) \\ \mapsto & \mathsf{P} \triangleright \circ \mid) & (S_1) \\ \mapsto & \circ \mid \varepsilon & (S_4) \end{array}$$

□

iii. [⋆⋆] If your answer to the previous question was *no*, amend the structure of the stack machine so that it does recognise N (efficiently). Explain your answer.

Solution: The pr s
in N placed next to e rackets,
followed by a sequenc
machine, it should not
there are two modes, *pushing* ($\succ$), and *popping* ($\prec$). The machine starts in *pushing* mode, i.e: $\circ \succ s$ for some string s, and now we have two terminating states: $\circ \succ \varepsilon$ and $\circ \prec \varepsilon$. Our transition rules are updated as follows:

$$\frac{}{s \succ (e \mapsto \mathsf{P} \triangleright s \succ e} \qquad \frac{}{s \succ \langle e \mapsto \mathsf{A} \triangleright s \succ e} \qquad \frac{}{s \succ [e \mapsto \mathsf{B} \triangleright s \succ e}$$

$$\frac{}{\mathsf{P} \triangleright s} \qquad \frac{}{} \qquad \frac{}{s \succ\ ]e \mapsto \mathsf{B} \triangleright s \prec\ ]e}$$

$$\prec\ ]e \mapsto s \prec e$$

As there are no rules to go from *popping* to *p* not push a symbol after one has been popped, and hence the machine recogni

2. **Computing Machines**: Abstract machines are not just used for d ers), they can also be used to compute results. Can you think of a machine to compute binary addition?

   (a) [⋆⋆⋆] Formalise such a machine.

   Hint: Think about the algorithm you would use when adding up large binary numbers on paper.

Solution: The machine's states are of the form:

$$\left.\begin{array}{l} n_1 \\ n_2 \end{array}\right\} s \ \langle\!\langle \mathsf{c} \rangle\!\rangle$$

Where s, n_1 and n_2 are strings of binary digits, and c is a single carry bit. n_1 and n_2 are also padded with zeros so as to be the same length.

Initial states are all states where s is empty and the carry bit is *zero*:

$$\left.\begin{array}{l} n_1 \\ n_2 \end{array}\right\} \varepsilon \ \langle\!\langle \mathsf{0} \rangle\!\rangle$$

Final states are all states where n_1 and n_2 are empty and the carry bit is *zero*:

$$\left.\begin{array}{l} \varepsilon \\ \varepsilon \end{array}\right\} s \ \langle\!\langle \mathsf{0} \rangle\!\rangle$$

The transition rules work as follows:

$$\frac{}{\left.\begin{array}{l} n_1\mathsf{0} \\ n_2\mathsf{0} \end{array}\right\} s \ \langle\!\langle \mathsf{0} \rangle\!\rangle \mapsto \left.\begin{array}{l} n_1 \\ n_2 \end{array}\right\} \mathsf{0}s \ \langle\!\langle \mathsf{0} \rangle\!\rangle} B_1 \qquad \frac{}{\left.\begin{array}{l} n_1\mathsf{0} \\ n_2\mathsf{1} \end{array}\right\} s \ \langle\!\langle \mathsf{0} \rangle\!\rangle \mapsto \left.\begin{array}{l} n_1 \\ n_2 \end{array}\right\} \mathsf{1}s \ \langle\!\langle \mathsf{0} \rangle\!\rangle} B_2$$

$$\frac{}{\left.\begin{matrix} n_1\mathtt{1} \\ n_2\mathtt{0} \end{matrix}\right\} s \;\langle\!\langle \mathtt{0} \rangle\!\rangle \mapsto \left.\begin{matrix} n_1 \\ n_2 \end{matrix}\right\} \mathtt{1}s \;\langle\!\langle \mathtt{0} \rangle\!\rangle}\,B_3 \qquad \frac{}{\left.\begin{matrix} n_1\mathtt{1} \\ n_2\mathtt{1} \end{matrix}\right\} s \;\langle\!\langle \mathtt{0} \rangle\!\rangle \mapsto \left.\begin{matrix} n_1 \\ n_2 \end{matrix}\right\} \mathtt{0}s \;\langle\!\langle \mathtt{1} \rangle\!\rangle}\,B_4$$

$$\frac{}{\left.\begin{matrix} n_1\mathtt{0} \\ n_2\mathtt{0} \end{matrix}\right\} s \;\langle\!\langle \mathtt{1} \rangle\!\rangle \mapsto \left.\begin{matrix} n_1 \\ n_2 \end{matrix}\right\} \mathtt{1}s \;\langle\!\langle \mathtt{0} \rangle\!\rangle}\,B_1c \qquad \frac{}{\left.\begin{matrix} n_1\mathtt{0} \\ n_2\mathtt{1} \end{matrix}\right\} s \;\langle\!\langle \mathtt{1} \rangle\!\rangle \mapsto \left.\begin{matrix} n_1 \\ n_2 \end{matrix}\right\} \mathtt{0}s \;\langle\!\langle \mathtt{1} \rangle\!\rangle}\,B_2c$$

$$\frac{}{\left.\begin{matrix} n_1\mathtt{1} \\ n_2\mathtt{0} \end{matrix}\right\} s \;\langle\!\langle \mathtt{1} \rangle\!\rangle \mapsto \left.\begin{matrix} n_1 \\ n_2 \end{matrix}\right\} \mathtt{0}s \;\langle\!\langle \mathtt{1} \rangle\!\rangle}\,B_3c \qquad \frac{}{\left.\begin{matrix} n_1\mathtt{1} \\ n_2\mathtt{1} \end{matrix}\right\} s \;\langle\!\langle \mathtt{1} \rangle\!\rangle \mapsto \left.\begin{matrix} n_1 \\ n_2 \end{matrix}\right\} \mathtt{1}s \;\langle\!\langle \mathtt{1} \rangle\!\rangle}\,B_4c$$

$$\frac{}{\left.\begin{matrix} \varepsilon \\ \varepsilon \end{matrix}\right\} s \;\langle\!\langle \mathtt{1} \rangle\!\rangle \mapsto \left.\begin{matrix} \varepsilon \\ \varepsilon \end{matrix}\right\} \mathtt{1}s \;\langle\!\langle \mathtt{0} \rangle\!\rangle}\,B_{\text{overflow}}$$

(b) [⋆] Compute the result

Solution:

The result is 10000, as shown below:

$$\begin{array}{lll}
& \left.\begin{matrix} \mathtt{0110} \\ \mathtt{1010} \end{matrix}\right\} \varepsilon \;\langle\!\langle \mathtt{0} \rangle\!\rangle & \\
\mapsto & \left.\begin{matrix} \mathtt{011} \\ \mathtt{101} \end{matrix}\right\} 0 \;\langle\!\langle \mathtt{0} \rangle\!\rangle & (B_1) \\
\mapsto & \left.\begin{matrix} \mathtt{01} \\ \mathtt{10} \end{matrix}\right\} 00 \quad \mathtt{1} & (B_4) \\
\mapsto & \left.\begin{matrix} \mathtt{0} \\ \mathtt{1} \end{matrix}\right\} 000 & \\
\mapsto & \left.\begin{matrix} \varepsilon \\ \varepsilon \end{matrix}\right\} 0000 & {}_2 \\
\mapsto & \left.\begin{matrix} \varepsilon \\ \varepsilon \end{matrix}\right\} 10000 \;\langle\!\langle \mathtt{0} \rangle\!\rangle & (B_{\text{overflow}})
\end{array}$$

3. **Evaluation Machines**: Because machines can express computation, we can also use them to express the operational semantics of a programming language. Imagine an extremely simple functional language with the following big-step semantics:

$$\frac{}{\mathtt{lam}(x, y) \Downarrow \langle\!\langle x.y \rangle\!\rangle}\,\textsc{Lambda} \qquad \frac{e_1 \Downarrow \langle\!\langle x.y \rangle\!\rangle \quad e_2 \Downarrow e_2' \quad y[x := e_2'] \Downarrow r}{\mathtt{apply}(e1, e2) \Downarrow r}\,\textsc{Apply}$$

(a) [⋆⋆] Develop a structural operational ("small step") semantics for this language.

i. Include three rules for function application. Assume the function expression is evaluated *before* the argument expression. Note that this language does *not* include explicit recursion.

Solution:

$$\frac{t_1 \mapsto t_1'}{\mathtt{apply}(t_1, t_2) \mapsto \mathtt{apply}(t_1', t_2)}\,\textsc{Apply}_1 \qquad \frac{t_2 \mapsto t_2'}{\mathtt{apply}(\mathtt{lam}(x.y), t_2) \mapsto \mathtt{apply}(\mathtt{lam}(x.y), t_2')}\,\textsc{Apply}_2$$

$$\frac{}{\mathtt{apply}(\mathtt{lam}(x.y), \mathtt{lam}(a.b)) \mapsto y[x := \mathtt{lam}(a.b)]}\,\textsc{Apply}_3$$

(b) Now define an abstract machine which eliminates recursion from the meta-level of the semantics to include an explicit stack, *a la* the *C Machine*.

i. [⋆] Define a suitable stack formalism.

Solution:

$$\frac{}{\circ\ \mathit{Stack}} \qquad \frac{x\ \mathit{Frame} \quad s\ \mathit{Stack}}{x \triangleright s\ \mathit{Stack}}$$

Where a Frame is simply either $\texttt{apply}(\square, x)$ or $\texttt{apply}(x, \square)$ for some x.

ii. [⋆⋆] Define the set of states Σ, the set of initial states $I \subseteq \Sigma$, and the set of final states $F \subseteq \Sigma$.

Solution: The set of states consists of an expression, and a stack:

$$\frac{s\ \mathit{Stack} \quad e\ \mathit{Expr}}{s \mid e \in \Sigma} \tag{1}$$

Initial states are an expression with an empty stack:

$$\frac{e\ \mathit{Expr}}{e \quad I}$$

Final states are a funct

$$\frac{}{\circ \mid \texttt{lam}(x.e) \in F}$$

iii. [⋆⋆] Include three rules for function application, using capture-avoiding substitution as a built-in machine operation.

Solution:

$$\frac{}{\mapsto}$$

$$\frac{}{\square) \triangleright s \mid e_2}$$

$$\frac{}{\texttt{apply}(\texttt{lam}(x.y), \square) \triangleright s \mid \texttt{lam}(a.b) \mapsto s \quad y[x := \texttt{lam}(a.b)]}$$

(c) Now suppose that we want to eliminate substitution from our machine. E include environments, *a la* the *E Machine*. Recall than ned as:

$$\frac{}{\bullet\ \mathit{Env}} \qquad \frac{x\ \mathit{Ident} \quad y\ \mathit{Expr} \quad \Gamma\ \mathit{Env}}{x \leftarrow y; \Gamma\ \mathit{Env}}$$

i. [⋆⋆] Revise your definition of the state sets Σ, I and F, and of the stack.

Solution: Our stack can now also include *environments*:

$$\frac{s\ \mathit{Stack} \quad \Gamma\ \mathit{Env}}{\Gamma \triangleright s\ \mathit{Stack}}$$

Our state now also includes a *current environment*, of the form $s \mid \Gamma \mid e$, where s is a Stack, Γ is an environment and e is an expression.
I and F are unchanged except that they include the empty environment.

ii. [⋆⋆⋆] Add a transition rule for function literals. Note that these function literals should produce *closures* which capture the environment at their definition.

Solution:

$$\frac{}{s \mid \Gamma \mid \texttt{lam}(x, y) \mapsto s \mid \Gamma \mid \langle\!\langle \Gamma, x.y \rangle\!\rangle}$$

iii. [⋆⋆⋆] Revise your rules for function application.

Solution:

$$\frac{}{s \mid \texttt{apply}(e_1, e_2) \mapsto \texttt{apply}(\square, e_2) \triangleright s \mid e_1}$$

$$\frac{}{\texttt{apply}(\Box, e_2) \triangleright s \mid \Gamma \mid \langle\langle \Delta, x.y \rangle\rangle \mapsto \texttt{apply}(\langle\langle \Delta, x.y \rangle\rangle, \Box) \triangleright s \mid \Delta \mid e_2}$$

$$\frac{}{\texttt{apply}(\langle\langle \Gamma, x.y \rangle\rangle, \Box) \triangleright s \mid \Delta \mid \langle\langle E, a.b \rangle\rangle \mapsto \Delta \triangleright s \mid x \leftarrow \langle\langle E, a.b \rangle\rangle; \Gamma \mid y}$$

iv. [⋆⋆⋆] Include any additional rules necessary to complete the definition, such as variable lookup.

Solution: Variable Lookup:

$$\frac{}{s \mid x \leftarrow y; \Gamma \mid x \mapsto s \mid x \leftarrow y; \Gamma \mid y}$$

Popping environments from the stack, back into the current environment:

$$\frac{}{\Gamma \triangleright s \mid \Delta \mid \langle\langle E, x.y \rangle\rangle \mapsto s \mid \Gamma \mid \langle\langle E, x.y \rangle\rangle}$$

v. [⋆⋆] Give an example of an e... es in order to evaluate correctly. Explain you...

Solution: A simple example is:

$$\texttt{apply}(\texttt{apply}(\texttt{lam}(x, \texttt{lam}(y, \texttt{apply}(x, y))), \texttt{lam}(a, a)), \texttt{lam}(b, b))$$

Evaluating the outer application will cause the inner application to be evaluated first, where x is bound to $\langle\langle a.a \rangle\rangle$. Without closures, the inner application will return the function $\langle\langle y.\texttt{apply}$ ... lting in a free variable inside the fu ... p encountering x fr ... ronment containin ... x will not be found free.

4. **Stack Machines**: In this question, we will examine a machine that is qu... used in *virtual machines*, such as the JVM, called a stack machine. Ima... language with the following big step semantics:

$$\frac{x \in \mathbb{Z}}{\texttt{num}(x) \Downarrow x}\textsc{Num} \qquad \frac{x \Downarrow x' \quad y \Downarrow y'}{\texttt{plus}(x, y) \Downarrow x' + y'}\textsc{Plus} \qquad \frac{x \Downarrow x' \quad y \Downarrow y'}{\texttt{times}(x, y) \Downarrow x' \times y'}\textsc{Times}$$

We have a machine, called the *J Machine*, that's capable of performing these operations, however it works by using a stack to store operands and accumulate results. For example, 4 * (2 + 3) would be the following program in the *J Machine*'s bytecode: `val(4);val(2);val(3);add;times`. Each `val` instruction pushes a value to the stack, and each operation instruction pops two values off, and pushes the result of the operation.

Formally, the *J Machine* is specified as follows: The machine consists of three *instructions*:

$$\frac{x \in \mathbb{Z}}{\texttt{val}(x)\ \mathit{Inst}} \qquad \frac{}{\texttt{plus}\ \mathit{Inst}} \qquad \frac{}{\texttt{times}\ \mathit{Inst}}$$

The state of the machine consists of a list of instructions, called a *Program*, and a stack of integers:

$$\frac{}{\texttt{halt}\ \mathit{Program}} \qquad \frac{i\ \mathit{Inst} \quad p\ \mathit{Program}}{i; p\ \mathit{Program}}$$

$$\frac{}{\circ\ \mathit{Stack}} \qquad \frac{x \in \mathbb{Z} \quad s\ \mathit{Stack}}{x \triangleright s\ \mathit{Stack}}$$

They are presented in the form $s \mid p$ where s is a stack and p is program. The initial state consists of the empty stack and any nonempty program p i.e, $\circ \mid p$. The final state consists of a stack with merely one element r (the result of the computation), and the empty program, i.e, $r \triangleright \circ \mid \texttt{halt}$.

The state transition rules are as follows:

$$\frac{}{s \mid \texttt{val}(x);p \mapsto x \triangleright s \mid p}J_1 \qquad \frac{}{y \triangleright x \triangleright s \mid \texttt{add};p \mapsto x+y \triangleright s \mid p}J_2 \qquad \frac{}{y \triangleright x \triangleright s \mid \texttt{times};p \mapsto x \times y \triangleright s \mid p}J_3$$

(a) [★★] Translate the expression `plus(times(num(-1),num(7)),num(7))` into a *J Machine* program, and write down each step the *J Machine* would take to execute this program.

Solution: The program is: `val(-1);val(7);times;val(7);plus;halt`.

Execution is as follows:

	$\circ \mid$ `val(-1);val(7);times;val(7);plus;halt`	
$\mapsto$	-1 $\triangleright \circ \mid$ `val(7);times;val(7);plus;halt`	(J_1)
$\mapsto$	7 $\triangleright$ -1 $\triangleright \circ \mid$ `times;val(7);plus;halt`	(J_1)
$\mapsto$	-7 $\triangleright \circ \mid$ `val(7);plus;halt`	(J_3)
$\mapsto$	7 $\triangleright$ -7 $\triangleright \circ \mid$ `plus;halt`	(J_1)
$\mapsto$	0 $\triangleright \circ \mid$ `halt`	(J_2)

(b) [★★★] Formalise (using infere … am which translates expressions in the arithmetic l … bytecode. You may assume that the semicolon op …

Solution:

$$\frac{}{\texttt{num}(n) \curlyvee \texttt{val}(n);\texttt{halt}}\text{NUM}_J$$

$$\frac{n \curlyvee n';\texttt{halt} \quad m \curlyvee m';\texttt{halt}}{\texttt{plus}(n,m) \quad n';m';\texttt{plus};\texttt{halt}}\text{PLUS}_J \qquad \frac{n \curlyvee n';\texttt{halt} \quad m \curlyvee m';\texttt{halt}}{\texttt{times}(n,m) \quad n';m';\texttt{times};\texttt{halt}}\text{TIMES}_J$$

(c) [★★★★] Suppose we w … etic language, using environments as sh …

$$\frac{x \in \mathbb{Z}}{\Gamma \vdash \texttt{num}(x) \Downarrow x}\text{NUM} \qquad \frac{\Gamma \vdash x \Downarrow x' \quad \Gamma \vdash y \Downarrow y'}{\Gamma \vdash \texttt{plus}(x,y) \Downarrow x'} \qquad \frac{\Gamma \quad x \quad x' \quad \Gamma \vdash y \Downarrow y'}{) \Downarrow x' \times y'}\text{TIMES}$$

$$\frac{\Gamma \vdash e_1 \Downarrow v_1 \quad \Gamma \cup \{x \leftarrow v_1\} \vdash e_2}{\Gamma \vdash \texttt{let}(x,e_1,e_2) \Downarrow v_2} \qquad \frac{}{\Gamma \vdash \texttt{var}(x) \Downarrow v}\text{VAR}$$

Extend the *J Machine* to support this construct, and expand your $\curlyvee$ relation to include the correct translation. Don't forget to deal with name shadowing by exploiting stacks.

Solution: We extend the state definition of the states in the machine to include an *additional* stack of environments, called *scopes*, notated as $z \mid s \mid p$, where z is the integer stack and s is the scope stack. The initial states now look like this:

$$\frac{}{\circ \mid \{\} \mid p}$$

That is, they start with the empty environment sitting at the bottom of the scope stack. Similarly, final states also have the empty environment only on their scope stack.

We introduce three new instructions, `scope`, `descope`, and `var`, which have the following semantics:

`scope(x)` pushes a new environment to the scope stack. The new environment is the same as the old environment except it includes a new binding[1] from the name x to the value on the top of the value stack. The value stack is also popped.

`descope(x)` simply pops the scope stack. `var(x)` pushes the value of a variable to the value stack. The value is determined by looking in the topmost scope environment.

$$\frac{}{v \triangleright s \mid \Gamma \triangleright \zeta \mid \texttt{scope}(x);p \mapsto s \mid \Gamma \cup \{x \leftarrow v\} \triangleright \zeta \mid p}J_4 \qquad \frac{}{s \mid \Gamma \triangleright \zeta \mid \texttt{descope};p \mapsto s \mid \zeta \mid p}J_5$$

$$\frac{}{s \mid \{x \leftarrow v\} \cup \Gamma \triangleright \zeta \mid \texttt{var}(x);p \mapsto v \triangleright s \mid \{x \leftarrow v\} \cup \Gamma \triangleright \zeta \mid p}J_6$$

[1]: Because each environment is a superset of the last, pointer magic could be used here to make this efficient in practice.

As for the compilation relation, we translate `let` as follows:

$$\frac{e_1 \curlyvee e_1'; \texttt{halt} \quad e_2 \curlyvee e_2'; \texttt{halt}}{\texttt{let}(x, e_1, e_2) \curlyvee e_1'; \texttt{scope}(x); e_2'; \texttt{descope}; \texttt{halt}}\,\textsc{Let}_J$$

And, for variable lookup, it's quite simple:

$$\frac{}{\texttt{var}(x) \curlyvee \texttt{var}(x)}\,\textsc{Var}_J$$

Note: This is basically how the JVM bytecode works (modulo some OO features).