COMP3161/COMP9164
Semantics Exercises

Liam O’Connor
September 29, 2019

1. [⋆⋆] Jankenbon:
Below is a language specified to compute the results of rock paper scissors tournaments:

\[ L ::= \varnothing \mid \diamond \mid \square \mid (\text{Play } L \ L) \]

We assume the existence of an operation \texttt{match} which computes the result of an individual game.

\[
\begin{align*}
\text{match}(\varnothing, \varnothing) &= \varnothing \\
\text{match}(\varnothing, \diamond) &= \diamond \\
\text{match}(\varnothing, \square) &= \square \\
\text{match}(\diamond, \varnothing) &= \varnothing \\
\text{match}(\diamond, \diamond) &= \varnothing \\
\text{match}(\diamond, \square) &= \square \\
\text{match}(\square, \varnothing) &= \diamond \\
\text{match}(\square, \diamond) &= \varnothing \\
\text{match}(\square, \square) &= \square
\end{align*}
\]

Here are some small step semantics to compute the tournament result:

\[
\frac{e_1 \mapsto e_1'}{(\text{Play } e_1 \ e_2) \mapsto (\text{Play } e_1' \ e_2')}
\]
\[
\frac{v_1 \in \{\varnothing, \diamond, \square\} \quad e_2 \mapsto e_2'}{(\text{Play } v_1 \ e_2) \mapsto (\text{Play } v_1' \ e_2')}
\]
\[
\frac{v_1 \in \{\varnothing, \diamond, \square\} \quad v_2 \in \{\varnothing, \diamond, \square\}}{(\text{Play } v_1 \ v_2) \mapsto \text{match}(v_1, v_2)}
\]

Determine a suitable big-step equivalent semantics for this language.

\begin{center}
\begin{tabular}{|c|}
\hline
Solution: The set of evaluable expressions is just \( L \). The set of values is \{\varnothing, \diamond, \square\}. We have three axioms:
\hline
\end{tabular}
\end{center}

\[
\begin{array}{c}
\varnothing \downarrow \varnothing \\
\diamond \downarrow \diamond \\
\square \downarrow \square
\end{array}
\]

And one other rule:

\[
\frac{e_1 \downarrow v_1 \quad e_2 \downarrow v_2}{(\text{Play } e_1 \ e_2) \downarrow \text{match}(v_1, v_2)}
\]

2. Logical Formulae: Imagine we have a simple propositional expression language\(^1\):

\[
\begin{array}{cccc}
\top & \downarrow \text{PROP} & \bot & \downarrow \text{PROP} \\
\text{PROP} & \text{PROP} & \text{PROP} & \text{PROP} \\
x \land y & \text{PROP} & \text{PROP} & \text{PROP} \\
x \rightarrow y & \text{PROP} & \text{PROP} & \text{PROP}
\end{array}
\]

\(^1\)Yes, the grammar is ambiguous, but assume it’s just a symbolic representation of abstract syntax.
(a) Here are the big-step evaluation semantics for this language:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Truth Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \top )</td>
<td>True</td>
</tr>
<tr>
<td>( \bot )</td>
<td>False</td>
</tr>
<tr>
<td>( x )</td>
<td>True</td>
</tr>
<tr>
<td>( \neg x )</td>
<td>False</td>
</tr>
<tr>
<td>( x \land y )</td>
<td>False</td>
</tr>
<tr>
<td>( x \lor y )</td>
<td>False</td>
</tr>
</tbody>
</table>

Determine a small step (SOS) semantics for the language \( \text{Prop} \).

i. \([\star]\) Identify the set of states \( \Sigma \), the set of initial states \( I \), and the set of final states \( F \).

**Solution:** The set of states \( \Sigma \) is simply the set of all expressions \( \text{Prop} \). \( F = \{ \top, \bot \} \). \( I = \Sigma \setminus F \).

ii. \([\star\star]\) Provide inference rules for a relation \( \mapsto \subseteq \Sigma \times \Sigma \), which performs one step only of the expression evaluation.

\[
\begin{align*}
\text{SOS-\text{Not}_1} & : \quad a \mapsto a' & \neg a \mapsto \neg a' \\
\text{SOS-\text{Not}_2} & : \quad x \mapsto y & \neg x \mapsto y \\
\text{SOS-\text{And}_1} & : \quad x \mapsto y & \neg x \mapsto \neg y \\
\text{SOS-\text{And}_2} & : \quad x \mapsto y & \neg x \mapsto \neg y
\end{align*}
\]

(b) We shall now prove that the reflexive, transitive closure of \( \mapsto \), \( \mapsto^* \), is implied by the big-step semantics above. \( \mapsto^* \) is defined by the following rules:

\[
\begin{align*}
\text{Reflexivity} & : \quad x \mapsto x \\
\text{Transitivity} & : \quad x \mapsto y \quad y \mapsto z \quad x \mapsto z
\end{align*}
\]

i. \([\star\star]\) First prove the following transitivity lemma:

\[
\frac{p \mapsto q \quad q \mapsto r}{p \mapsto r}
\]

**Solution:** We will proceed by rule induction on the first premise, that \( p \mapsto^* q \). Therefore, for each case, we must show:

\[
\begin{align*}
q \mapsto r & : \quad p \mapsto r \\
\text{Base case (where } p = q \text{ from rule Refl*)}. \quad \text{For this case, as } p = q, \text{ our goal is just} \\
q \mapsto r & : \quad q \mapsto r
\end{align*}
\]

which is trivial.

**Inductive case (from rule Trans*)**. We know that \( p \mapsto p' \) (\( \ast \)) and \( p' \mapsto q \) (\( \ast\ast \)). We must show that \( q \mapsto^* r \) (\( \ast\ast\ast \)) implies \( p \mapsto^* r \), given the inductive hypothesis from (\( \ast\ast \)) that:

\[
\begin{align*}
q \mapsto^* r & : \quad p' \mapsto^* r \\
\text{IH} & : \quad q \mapsto^* r
\end{align*}
\]

We can now show our goal:

\[
\begin{align*}
p \mapsto_{(\ast)} & : \quad p \mapsto p' \\
q \mapsto^* & : \quad q \mapsto^* r \\
\text{IH} & : \quad p' \mapsto^* r \\
\text{Trans*} & : \quad p \mapsto^* r
\end{align*}
\]
ii. [⋆⋆] Now prove the following two lemmas about Not:

\[
\begin{align*}
    x \mapsto \top & \quad \text{LEMMA-NOT}_1 \\
    x \mapsto \bot & \quad \text{LEMMA-NOT}_2
\end{align*}
\]

Solution:

Proof of Lemma-Not\(_1\). We proceed by rule induction on the premise, that \(x \mapsto \top\), with the goal of proving that \(\neg x \mapsto \bot\).

Base Case (from \(\text{Refl}^*\)), where \(\top \mapsto \top\), we must show that \(\neg \top \mapsto \bot\):

\[
\begin{align*}
    \neg \top & \mapsto \bot \\
    \bot & \mapsto \bot \quad \text{SOS-Not}_2
\end{align*}
\]

Inductive Case (from \(\text{Trans}^*\)), where \(x \mapsto x'\) (⋆) and \(x' \mapsto \top\) (**), we must show that \(\neg x \mapsto \bot\), with the inductive hypothesis from (**) that \(\neg x' \mapsto \bot\).

\[
\begin{align*}
    x \mapsto x' & \quad \text{SOS-Not}_1 \\
    \neg x' & \mapsto \bot \quad \text{I.H}
\end{align*}
\]

Proof of Lemma-Not\(_2\) is very similar and is omitted.

iii. [⋆⋆] Now prove the following lemmas about And:

\[
\begin{align*}
    x \mapsto \bot & \quad \text{LEMMA-AND}_1 \\
    x \mapsto \top & \quad \text{LEMMA-AND}_2
\end{align*}
\]

Solution:

Proof of Lemma-And\(_1\). We proceed by rule induction on the premises that \(x \mapsto \bot\), with the goal of proving that \(x \land y \mapsto \bot\).

Base Case (from \(\text{Refl}^*\)), where \(\bot \mapsto \bot\), we must show that \(\bot \land y \mapsto \bot\):

\[
\begin{align*}
    \bot \land y & \mapsto \bot \\
    \bot & \mapsto \bot \quad \text{SOS-AND}_3 \\
    y & \mapsto \bot
\end{align*}
\]

Inductive Case (from \(\text{Trans}^*\)), where \(x \mapsto x'\) (⋆) and \(x' \mapsto \top\) (**), we must show that \(x \land y \mapsto \bot\), with the inductive hypothesis from (**) that \(x' \land y \mapsto \bot\).

\[
\begin{align*}
    x \land y & \mapsto \bot \\
    x' \land y & \mapsto \bot \quad \text{SOS-AND}_1 \\
    x' & \mapsto \bot \quad \text{I.H}
\end{align*}
\]

Proof of Lemma-And\(_2\) is very similar, and is therefore omitted.

iv. [⋆⋆⋆] Using these lemmas or otherwise, show that \(E \Downarrow V\) implies \(\sigma_E \mapsto \sigma_V\), where \(\sigma_E\) is the state corresponding to the expression \(E\) and \(\sigma_V\) is the final state corresponding to the value \(V\).

Solution: We define \(\sigma_{\text{true}} = \top\), \(\sigma_{\text{false}} = \bot\), and \(\sigma_E = E\) for all expressions \(E\).

We proceed by rule induction on the rules of \(\Downarrow\).

Base case (from rule True). We must show that \(\top \mapsto \sigma_{\text{true}}\), i.e. \(\top \mapsto \top\) which is true by rule \(\text{Refl}^*\).

Base case (from rule False) We must show that \(\bot \mapsto \sigma_{\text{false}}\), i.e. \(\bot \mapsto \bot\) which is true by rule \(\text{Refl}^*\).

Not cases:

Inductive case (from rule Not\(_1\)). We must show, assuming \(E \Downarrow \text{True}\) (⋆), that \(\neg x \mapsto \bot\).
We have the I.H. from (*) that \( x \vdash \sigma_{\text{True}} \), i.e. \( x \vdash \top \). By LEMMA-NOT₁, we can conclude that \( \neg x \vdash \bot \) as required.

**Inductive case (from rule NOT₂).** We must show, assuming \( x \downarrow \text{False} (*) \), that \( \neg x \vdash \top \).

We have the I.H. from (*) that \( x \vdash \sigma_{\text{False}} \), i.e. \( x \vdash \bot \). By LEMMA-NOT₂, we can conclude that \( \neg x \vdash \top \) as required.

**Inductive case (from rule AND₁).** We must show, assuming \( x \downarrow \text{False} (*) \), that \( x \land y \vdash \bot \).

We have the I.H. from (*) that \( x \vdash \bot \). Thus, we have our goal from LEMMA-AND₁ with the I.H.

**Inductive case (from rule AND₃).** We must show, assuming \( x \downarrow \text{True} (*) \) and \( y \downarrow v (***) \), that \( x \land y \vdash \sigma_v \).

We have two inductive hypotheses:

1. \( x \vdash \top \) from (*)
2. \( y \vdash \sigma_v \) from (***)

We can show our goal as follows:

\[
\begin{array}{c}
\frac{IH_1}{x \vdash \top} & \text{LEMMA-AND}_2 & \frac{IH_2}{y \vdash \sigma_v} \\
\frac{x \land y \vdash y}{x \land y \vdash \sigma_v} & \text{TRANSITIVE} \\
\end{array}
\]

Thus, by induction, we have shown that \( E \downarrow V \implies E \vdash \sigma_V \)

(c) Now we will prove the other direction of the equivalence.

i. [****] Prove the following lemma by rule induction on the small-step premise:

\[
\frac{e \vdash e'}{e \downarrow v} \frac{e' \downarrow v}{e \downarrow v}
\]

**Hint:** It might help to keep \( v \) arbitrary in the induction, and thus prove for each case:

\[
\forall v. \frac{e' \downarrow v}{e \downarrow v}
\]

**Solution:** We begin rule induction on the cases for \( e \vdash e' \).

**Base case (from rule SOS-NOT₁).** Where \( e = \neg \top \) and \( e' = \bot \). We must show that \( \bot \downarrow v \) implies \( \neg \top \downarrow v \). The only way for \( \bot \downarrow v \) to hold is if \( v = \text{False} \). Thus we can show \( \neg \top \downarrow v \) using rules NOT₁ and TRUE.

**Base case (from rule SOS-NOT₃).** Where \( e = \neg \bot \) and \( e' = \top \). We must show that \( \top \downarrow v \) implies \( \neg \bot \downarrow v \). The only way for \( \top \downarrow v \) to hold is if \( v = \text{True} \). Therefore we must show \( \neg \bot \downarrow \text{True} \), trivial from rules NOT₃ and FALSE.

**Base case (from rule SOS-AND₂).** Where \( e = \top \land e' \). We must show that \( e' \downarrow v \) implies \( \top \land e' \downarrow v \). This is trivially shown by rule AND₂ and TRUE.

**Base case (from rule SOS-AND₃).** Where \( e = \bot \land e' \). We must show that, assuming \( \bot \land e' \downarrow v (* \*) \), that \( \bot \downarrow v \). If \( v \) were True, the assumption (*) would indicate that \( \bot \downarrow \text{True} \), which is impossible. Therefore, \( v \) must be False, and our goal can be trivially shown by the rule FALSE.

**Inductive case (from rule SOS-NOT₁).** Where \( e = \neg a \) and \( e' = \neg a' \) and \( a \vdash a' \). We get the inductive hypothesis that:

\[
\forall v. \frac{a' \downarrow v}{a \downarrow v}
\]

And we must show that, assuming \( \neg a' \downarrow v \), that \( a \downarrow v \). If \( a' \downarrow v \), this must indicate that either \( a' \downarrow \text{True} \), in which case \( v = \text{False} \) or \( a' \downarrow \text{False} \), in which case \( v = \text{True} \). Either way,
we can use our I.H. to conclude that $a \downarrow \text{True}$ or $a \downarrow \text{False}$ respectively. Then we may show our goal using either $\text{NOT}_1$ or $\text{NOT}_2$.

Inductive case (from rule SOS-And$_1$) Where $e = a \land b$ and $e' = a' \land b$ and $a \Rightarrow a'$. We get the inductive hypothesis that:

$$\forall v. \ a' \downarrow v \Rightarrow a \downarrow v$$

And we must show that, assuming $a' \land b \downarrow v$, that $a \land b \downarrow v$. If $a' \land b \downarrow v$, this must indicate that either $a' \downarrow \text{True}$, in which case $b \downarrow v$, or $a' \downarrow \text{False}$, in which case $v = \text{False}$. Either way, we can use our I.H. to conclude that $a \downarrow \text{True}$ or $a \downarrow \text{False}$ respectively. Then we may show our goal using either $\text{AND}_1$ or $\text{AND}_2$.

ii. [$\star$] Now prove that every completed small step trace has a corresponding big step evaluation:

$$e \Rightarrow e' \Rightarrow \sigma_v$$

**Solution:** We begin rule induction on the cases for $e \Rightarrow e' \Rightarrow \sigma_v$.

**Base case (from rule Refl$^*$).** Where $e = \sigma_v$. The term $e$ can either be $\top$ or $\bot$, and in either case we have that $e \downarrow v$.

**Inductive case (from rule Trans$^*$) Where $e \Rightarrow e'$ and $e' \Rightarrow \sigma_v$.** We get the inductive hypothesis that $e' \downarrow v$. We must show that $e \downarrow v$, which can be done easily using the lemma proven in the previous part.

(d) [$\star\star$] Suppose we wanted to add quantifiers and variables to our logic language:

$$
\frac{x \text{ Prop}}{\exists v. \ x \text{ Prop}} \quad \frac{x \text{ Prop}}{\forall v. \ x \text{ Prop}} \quad \frac{v \text{ is a variable name}}{v \text{ Prop}}
$$

Write a static semantics judgement for this language, written $\vdash e \text{ Ok}$ (with whatever context you like before the $\vdash$), that ensures that there are no free variables in a given logical formula. Remember that a free variable is a variable that is not bound by a quantifier or lambda.

**Solution:** The context shall be a set of variable names, denoted $\Gamma$.

$$
\frac{v \in \Gamma}{\Gamma \vdash v \text{ Ok}} \quad \frac{\Gamma \cup \{v\} \vdash x \text{ Ok}}{\Gamma \vdash \exists v. \ x \text{ Ok}} \quad \frac{\Gamma \vdash x \text{ Ok}}{\Gamma \vdash \forall v. \ x \text{ Ok}} \quad \frac{\Gamma \vdash x \text{ Ok}}{\Gamma \vdash \neg x \text{ Ok}} \quad \frac{\Gamma \vdash y \text{ Ok}}{\Gamma \vdash x \land y \text{ Ok}} \quad \frac{\Gamma \vdash y \text{ Ok}}{\Gamma \vdash x \lor y \text{ Ok}}
$$

This is more or less analogous to the scope-checking rules presented in lectures.

3. **Bizarro-Poland:** Imagine we have a reverse Polish notation calculator language. Reverse Polish notation is an old calculator format that does not require the use of parenthetical expressions. To achieve this, it moves all operators to post-fix, rather than in-fix order. E.g. $1 + 2$ becomes $1 \ 2 +$, or $1 - (3 + 2)$ becomes $1 \ 3 \ 2 + -$. These calculators evaluated these expressions by pushing symbols onto a stack until an operator was encountered, when two symbols would be popped off and the result of the operation pushed on. The grammar is easily defined:

$$
\frac{x \in N}{x \text{ Symbol}} \quad \frac{x \in \{+, -, /, \ast\}}{x \text{ Symbol}}
$$

$$
\frac{\epsilon \text{ RPN}}{x \text{ Symbol}} \quad \frac{x \text{ RPN}}{x \text{ RPN}} \quad \frac{x \ast \text{ RPN}}{x \ast \text{ RPN}}
$$

(a) The issue is that this grammar allows for invalid programs (such as $1 + 2$ or $- + \ast$).

i. [$\star\star$] Write some static semantics inference rules for a judgement $\vdash e \text{ Ok}$ to ensure that programs are well formed.
Solution: We will equip our rules with context that includes the number of values that are available on the stack.

The empty program is only valid if there is exactly one value left on the stack - this means the program will evaluate to one result:

\[
\frac{}{1 \vdash \epsilon \text{ Ok}} \quad \text{EMPTY}
\]

Prepending a number to a program means that one less value is required on the stack:

\[
\frac{x \in \mathbb{N} \quad (n+1) \vdash xs \text{ Ok} \quad \text{Num}}{n \vdash xs \text{ Ok} \quad \text{Num}} \quad \text{Num}
\]

Prepending a symbol will require two values on the stack, and produce one value. The structure of the rule we write reflects this:

\[
\frac{x \in \{+, -, *, /\} \quad (n+1) \vdash xs \text{ Ok}}{(n+2) \vdash xs \text{ Ok} \quad \text{Op}} \quad \text{Op}
\]

ii. \(\star\) Show that \(\vdash 1 \ 3 \ 2 \ + \ - \ \text{Ok}\).

Solution:

\[
\frac{}{1 \vdash \epsilon \text{ Ok} \quad \text{EMPTY}} \quad \frac{}{2 \vdash - \text{ Ok} \quad \text{Op}} \quad \frac{}{3 \vdash ++ \text{ Ok} \quad \text{Op}} \quad \frac{}{2 \vdash 2 + - \text{ Ok} \quad \text{Num}} \quad \frac{}{1 \vdash 3 \ 2 + - \text{ Ok} \quad \text{Num}} \quad \frac{}{0 \vdash 1 \ 3 \ 2 + - \text{ Ok} \quad \text{Num}}.
\]

(b) We will now define some big-step evaluation semantics for this calculator. It may be helpful to read the program from right-to-left rather than left-to-right.

i. \(\star\) Identify the set of evaluable expressions \(E\), and the set of result values \(V\). It may be helpful to use a stack, defined as follows:

\[
\frac{}{\circ \text{ Stack} \quad x \in \mathbb{N} \quad xs \text{ Stack} \quad x \triangleright xs \text{ Stack}}
\]

Solution: The set \(E\) is simply the set of all \(e\) such that \(e \ RPN\). \(V\) is the set of all stacks.

ii. \(\star\star\) Define a relation \(\Downarrow \subseteq E \times V\) which evaluates RPN programs.

Solution: The empty program evaluates to the empty stack:

\[
\epsilon \Downarrow \circ \quad \text{BS-EMPTY}
\]

The non-empty program ending in an operator performs the operation on the stack resulting from the previous symbols:

\[
\frac{xs \Downarrow a \triangleright b \triangleright s \quad x \in \{+, -, *, /\} \quad \text{BS-Op}}{xs x \Downarrow (a \ x \ b) \triangleright s}
\]

The non-empty program ending in a number simply pushes a number onto the stack from the previous symbols:

\[
\frac{xs \Downarrow s \quad x \in \mathbb{N} \quad \text{BS-Num}}{xs x \Downarrow x \triangleright s}
\]

(c) \(\star\star\) Now we will try small-step semantics.

Our states \(\Sigma\) will be of the form \(s \vdash p\) where \(s\) is a stack of natural numbers and \(p\) is an RPN program. Initial states are all states of the form \(\circ \vdash p\).

Final states are all states of the form \(n \triangleright \circ \vdash \epsilon\).
Define a relation $\Rightarrow \subseteq \Sigma \times \Sigma$, which evaluates one step of the calculation.

**Solution:** Numbers simply push to the stack:

\[
\begin{align*}
x \in \mathbb{N} & \quad s \vdash x \ x s \Rightarrow x \ x s \vdash x s \\
\text{SS-NUM} & \quad \text{Operations simply pop two elements from the stack and operate on them.}
\end{align*}
\]

\[
\begin{align*}
x \in \{+, -, *, /\} & \quad a \ b \ b \ s \vdash x \ x s \Rightarrow a \ x b \ b \ s \vdash x s \\
\text{SS-OP} & \quad \text{It is worth noting that the lemma Transitive from Question 1 applies here also.}
\end{align*}
\]

(d) \[\text{[****]}\] Now we will prove the easier direction of the equivalence proof. Show using rule induction on $\downarrow$ that, for all expressions $e$, if $e \downarrow v$ then $e \Rightarrow \sigma_e \Rightarrow \sigma_v$ (where $\sigma_e$ and $\sigma_v$ are initial and final states respectively corresponding to $e$ and $v$).

**Hint:** You may find it helpful to assume the following lemma (A proof of it is provided in the solutions):

\[
\begin{align*}
\circ \vdash x s & \Rightarrow v \vdash \epsilon \text{ APPEND} \\
\circ \vdash x s & \Rightarrow v \vdash x \\
\end{align*}
\]

It is worth noting that the lemma Transitive from Question 1 applies here also.

**Solution:** We shall define $\sigma_e$ as $\circ \vdash e$, and $\sigma_v$ as $v \vdash \epsilon$.

**Base case (from rule BS-EMPTY).** Where $e = \epsilon$ and $v = \circ$. We must show $\circ \vdash \epsilon \Rightarrow \circ \vdash \epsilon$, trivially true by rule Refl*.

**Inductive case (from rule BS-Num).** Here $e = x s$ for some $x \in \mathbb{N}$, and $v = x \Rightarrow v'$ for some stack $v'$. We know that $x s \downarrow v'$ which gives rise to the I.H. that $\circ \vdash x s \Rightarrow v' \vdash \epsilon$. Using APPEND with that I.H., we can conclude that $\circ \vdash x s x \Rightarrow v' \vdash x$ (**). As we know that $\Rightarrow$ is Transitive, we can write a chain of reasoning here rather than a derivation tree.

\[
\begin{align*}
\circ \vdash x s x \Rightarrow v' \vdash x & \quad (\ast) \\
\Rightarrow x \Rightarrow v' \vdash \epsilon & \quad \text{(SS-Num)}
\end{align*}
\]

**Inductive case (from rule BS-OP).** Here $e = x s$ for some operator $x$ and $v = (a x b) \Rightarrow v'$ for some stack $v'$. We know that $x s \downarrow v'$ which gives rise to the inductive hypothesis that $\circ \vdash x s \Rightarrow a \Rightarrow v' \vdash \epsilon$, we must show that $\circ \vdash x s x \Rightarrow (a x b) \Rightarrow v' \vdash \epsilon$. From APPEND, we can therefore conclude that $\circ \vdash x s x \Rightarrow a \Rightarrow b \Rightarrow v' \vdash x$ (**),

\[
\begin{align*}
\circ \vdash x s x \Rightarrow a \Rightarrow b \Rightarrow v' \vdash x & \quad (\ast) \\
\Rightarrow (a x b) \Rightarrow v' \vdash \epsilon & \quad \text{(SS-OP)}
\end{align*}
\]

Thus we have shown by induction that the big step semantics map to the small step semantics.

We still need to prove APPEND. We proceed rule induction on the premise, however we will strengthen our proof goal to the more general rule below:

\[
\begin{align*}
\vdash x s & \Rightarrow v \vdash \epsilon \text{ APPEND'} \\
\end{align*}
\]

This trivially implies APPEND by setting $s = \circ$.

**Base case (when $xs = \epsilon$ and $s = v$, from rule Refl*).** We must show that $v \vdash x \Rightarrow v \vdash x$, trivial by rule Refl*.

**Inductive case (when $s \vdash x s \Rightarrow s' \vdash x s'$ (**)) for some $s'$ and $x s'$, and $s' \vdash x s' \Rightarrow v \vdash \epsilon$ (**)). We get the inductive hypothesis from (**), that $s' \vdash x s' \Rightarrow v \vdash x$.

Looking at the possible cases for (**), we can always prove that $s \vdash x s \Rightarrow s' \vdash x s' x$ (**), as the rules for the small-step semantics only ever examine the left hand side of a non-empty sequence. Therefore, we may now show our goal:

\[
\begin{align*}
\vdash x s x \Rightarrow s' \vdash x s' x & \quad (***) \\
\Rightarrow v \vdash x & \quad \text{(I.H.)}
\end{align*}
\]
(e) [****] Show that your static semantics defined in (a) ensure that the program will not reach a stuck state. That is, show that \( \vdash e \text{ Ok} \implies \circ \vdash e \mapsto s \vdash \epsilon \), for some \( s \). You may find it helpful to generalise your proof goal before beginning induction.

**Solution:** We shall start by generalising our goal to the more flexible \( n \vdash e \text{ Ok} \implies v_1 \triangleright v_2 \triangleright \cdots \triangleright v_n \circ \vdash e \mapsto s \vdash \epsilon \) for some \( s \) and all \( v_1 \) through \( v_n \). This trivially implies our first goal by setting \( n = 0 \).

**Base case (where \( e = \epsilon \) and \( n = 1 \), from rule \text{ EMPTY}).** We must show that \( v_1 \triangleright \vdash \epsilon \mapsto s \vdash \epsilon \), for some \( s \). This is trivially true by rule \text{ REFL*}, where \( s = v_1 \circ \).

**Inductive case (where \( e = x \ xs \) for some \( x \in \mathbb{N} \), and \( n + 1 \vdash xs \text{ Ok} \) (\( * \)).** Our inductive hypothesis from (\( * \)) is that \( \mathcal{v}_1' \triangleright \cdots \triangleright \mathcal{v}_n' \triangleright \circ \vdash xs \mapsto s' \vdash \epsilon \) for some \( s' \) and all \( \mathcal{v}_i' \) through \( \mathcal{v}_{n+1}' \).

We can show our goal by setting our goal’s \( s = s' \), the IH’s \( \mathcal{v}_1' = x \) and \( \mathcal{v}_i' = \mathcal{v}_{i-1} \) for all subsequent \( i \).

\[
\begin{array}{c}
v_1 \triangleright \cdots \triangleright v_n \circ \vdash xs \mapsto x \triangleright v_1 \triangleright \cdots \triangleright v_n \circ \vdash xs \\
\hline
\text{SS-Num} \\
\hline
\mathcal{v}_1 \triangleright \mathcal{v}_2 \triangleright \cdots \triangleright \mathcal{v}_n \circ \vdash xs \mapsto s' \vdash \epsilon \end{array}
\]

\[
\begin{array}{c}
\vdash e \text{ Ok} \\
\hline
\text{IH} \\
\hline
\mathcal{v}_1 \triangleright \mathcal{v}_2 \triangleright \cdots \triangleright \mathcal{v}_n \circ \vdash xs \mapsto s' \vdash \epsilon \end{array}
\]

The case where \( x \) is an operator is very similar, just different stacks. \( \square \)