Concepts of Programmig Language Design

Semantics Exercises

Liam O'Connor-Davis, Gabriele Keller

December 5, 2024

1. **Logical Formulae:** Imagine we have a simple propositional expression language[1]:

$$\frac{}{\top\ \textbf{Prop}} \qquad \frac{}{\bot\ \textbf{Prop}} \qquad \frac{e_1\ \textbf{Prop} \quad e_2\ \textbf{Prop}}{e_1 \wedge e_2\ \textbf{Prop}} \qquad \frac{e_1\ \textbf{Prop}}{\neg e_1\ \textbf{Prop}}$$

   (a) The big-step semantics is given as

   - The set of evaluable expressions: $E = \{e \mid e\ \textbf{Prop}\}$
   - The set of values: $V = \{\texttt{True}, \texttt{False}\}$

   and the $\Downarrow$-relation, defined by the following rules:

$$\frac{}{\top \Downarrow \texttt{True}}\textsc{True} \qquad \frac{}{\bot \Downarrow \texttt{False}}\textsc{False}$$

$$\frac{e_1 \Downarrow \texttt{True}}{\neg e_1 \Downarrow \texttt{False}}\textsc{Not}_1 \qquad \frac{e_1 \Downarrow \texttt{False}}{\neg e_1 \Downarrow \texttt{True}}\textsc{Not}_2 \qquad \frac{e_1 \Downarrow \texttt{False}}{e_1 \wedge e_2 \Downarrow \texttt{False}}\textsc{And}_1 \qquad \frac{e_1 \Downarrow \texttt{True} \quad e_2 \Downarrow v}{e_1 \wedge e_2 \Downarrow v}\textsc{And}_2$$

   Determine a small step (SOS) semantics for the language **Prop**.

   i. Identify the set of states Q, the set of initial states I, and the set of final states F.

   > **Solution:** The set of states Q is simply the set of all expressions Prop. $F = \{\top, \bot\}$. $I = Q$.

   ii. Provide inference rules for a relation $\mapsto\ : Q \times Q$, which performs one step only of the expression evaluation.

   > **Solution:**
>
> $$\frac{a \mapsto a'}{\neg a \mapsto \neg a'}\text{SOS-Not}_1 \qquad \frac{}{\neg\top \mapsto \bot}\text{SOS-Not}_2 \qquad \frac{}{\neg\bot \mapsto \top}\text{SOS-Not}_3$$
>
> $$\frac{a \mapsto a'}{a \wedge b \mapsto a' \wedge b}\text{SOS-And}_1 \qquad \frac{}{\top \wedge b \mapsto b}\text{SOS-And}_2 \qquad \frac{}{\bot \wedge b \mapsto \bot}\text{SOS-And}_3$$

   (b) **Long, but not difficult:** We shall now prove that the reflexive, transitive closure of $\mapsto$, $\overset{\star}{\mapsto}$, is implied by the big-step semantics above. $\overset{\star}{\mapsto}$ is defined by the following rules:

$$\frac{}{e_1 \overset{\star}{\mapsto} e_1}\textsc{Refl}^* \qquad \frac{e_1 \mapsto e_2 \quad e_2 \overset{\star}{\mapsto} z}{e_1 \overset{\star}{\mapsto} z}\textsc{Trans}^*$$

[1] Yes, the grammar is ambiguous, but assume it's just a symbolic representation of abstract syntax.

i. First prove the following transitivity lemma:

$$\frac{p \overset{\star}{\mapsto} q \quad q \overset{\star}{\mapsto} r}{p \overset{\star}{\mapsto} r}\textsc{Transitive}$$

Solution: We will proceed by rule induction on the first premise, that $p \overset{\star}{\mapsto} q$.

Base case (from rule Refl**).* Substituting $p = q$ (from rule Refl*) into our proof goal, we get:

$$\frac{q \overset{\star}{\mapsto} q \quad q \overset{\star}{\mapsto} r}{q \overset{\star}{\mapsto} r}\textsc{Goal}$$

Which is trivially true.
Inductive case (from rule Trans*). We know $p \mapsto p'$ $(*)$, $p' \overset{\star}{\mapsto} q$ $(**)$, and $q \overset{\star}{\mapsto} r$ $(***)$. We must show that $p \overset{\star}{\mapsto} r$, given the inductive hypothesis:

$$\frac{p' \overset{\star}{\mapsto} q \quad q \overset{\star}{\mapsto} r}{p' \overset{\star}{\mapsto} r}IH$$

We can now show our goal:

$$\frac{\dfrac{}{p \mapsto p'}(*) \quad \dfrac{\dfrac{}{p' \overset{\star}{\mapsto} q}(**) \quad \dfrac{}{q \overset{\star}{\mapsto} r}(***)}{p' \mapsto r}IH}{p \overset{\star}{\mapsto} r}\textsc{Trans}^*$$

□

ii. Now prove the following two lemmas about Not:

$$\frac{e_1 \overset{\star}{\mapsto} \top}{\neg e_1 \overset{\star}{\mapsto} \bot}\textsc{Lemma-Not}_1 \qquad \frac{e_1 \overset{\star}{\mapsto} \bot}{\neg e_1 \overset{\star}{\mapsto} \top}\textsc{Lemma-Not}_2$$

Solution:

Proof of Lemma-Not$_1$. We proceed by rule induction on the premise, that $x \overset{\star}{\mapsto} \top$, with the goal of proving that $\neg x \overset{\star}{\mapsto} \bot$.
Base Case (from Refl*), where $\top \overset{\star}{\mapsto} \top$, we must show that $\neg\top \overset{\star}{\mapsto} \bot$:

$$\frac{\dfrac{}{\neg\top \mapsto \bot}\textsc{SOS-Not}_2 \quad \dfrac{}{\bot \overset{\star}{\mapsto} \bot}\textsc{Refl}^*}{\neg\top \overset{\star}{\mapsto} \bot}\textsc{Trans}^*$$

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

$$\frac{\dfrac{\dfrac{}{x \mapsto x'}(*)}{\neg x \mapsto \neg x'}\textsc{SOS-Not}_1 \quad \dfrac{\dfrac{}{x' \overset{\star}{\mapsto} \top}(**)}{\neg x' \overset{\star}{\mapsto} \bot}I.H}{\neg x \overset{\star}{\mapsto} \bot}\textsc{Trans}^*$$

Proof of Lemma-Not$_2$ is very similar and is omitted. □

iii. Now prove the following lemmas about And:

$$\frac{e_1 \overset{\star}{\mapsto} \bot}{e_1 \wedge e_2 \overset{\star}{\mapsto} \bot}\textsc{Lemma-And}_1 \qquad \frac{e_1 \overset{\star}{\mapsto} \top}{e_1 \wedge e_2 \overset{\star}{\mapsto} e_2}\textsc{Lemma-And}_2$$

Solution:

Proof of Lemma-And$_1$. We proceed by rule induction on the premises that $x \overset{\star}{\mapsto} \bot$, with the goal of proving that $x \wedge y \overset{\star}{\mapsto} \bot$.
Base Case (from Refl*), where $\bot \overset{\star}{\mapsto} \bot$, we must show that $\bot \wedge y \overset{\star}{\mapsto} \bot$:

$$\dfrac{\dfrac{}{\bot \wedge y \mapsto \bot}\text{SOS-And}_3 \quad \dfrac{}{\bot \overset{\star}{\mapsto} \bot}\text{Refl}^*}{\bot \wedge y \overset{\star}{\mapsto} \bot}\text{Trans}^*$$

Inductive Case (from Trans*), where $x \mapsto x'$ $(*)$ and $x' \overset{\star}{\mapsto} \bot (**)$, we must show that $x \wedge y \overset{\star}{\mapsto} \bot$, with the inductive hypothesis that $x' \overset{\star}{\mapsto} \bot \implies x' \wedge y \overset{\star}{\mapsto} \bot$.

$$\dfrac{\dfrac{\dfrac{}{x \mapsto x'}(*)}{x \wedge y \mapsto x' \wedge y}\text{SOS-And}_1 \quad \dfrac{\dfrac{}{x' \overset{\star}{\mapsto} \bot}(**)}{x' \wedge y \overset{\star}{\mapsto} \bot}I.H}{x \wedge y \overset{\star}{\mapsto} \bot}\text{Trans}^*$$

Proof of Lemma-And$_2$ is very similar, and is therefore omitted. □

iv. Using these lemmas or otherwise, show that $E \Downarrow V$ implies $Q_E \overset{\star}{\mapsto} Q_V$, where Q_E is the state corresponding to the expression E and Q_V is the final state corresponding to the value V.

Solution: We define $Q_{\texttt{True}} = \top$, $Q_{\texttt{False}} = \bot$, and $Q_E = E$ for all expressions E. We proceed by rule induction on the rules of $\Downarrow$.

Base case (from rule True*).* We must show that $\top \overset{\star}{\mapsto} Q_{\texttt{True}}$, i.e $\top \overset{\star}{\mapsto} \top$ which is true by rule Refl*.
Base case (from rule False) We must show that $\bot \overset{\star}{\mapsto} Q_{\texttt{False}}$, i.e $\bot \overset{\star}{\mapsto} \bot$ which is true by rule Refl*.
Not cases:
Inductive case (from rule Not$_1$). We must show, assuming $x \Downarrow \texttt{True}$ $(*)$ and $\neg x \Downarrow \texttt{False}$, that $\neg x \overset{\star}{\mapsto} \bot$.
We have the I.H $x \Downarrow V \implies x \overset{\star}{\mapsto} Q_V$ for any V. Unifying with $(*)$ we can conclude $x \overset{\star}{\mapsto} \top$. By Lemma-Not$_1$, we can conclude that $\neg x \overset{\star}{\mapsto} \bot$ as required.

Inductive case (from rule Not$_2$). We must show, assuming $x \Downarrow \texttt{False}$ $(*)$ and $\neg x \Downarrow \texttt{True}$, that $\neg x \overset{\star}{\mapsto} \top$.
We have the I.H $x \Downarrow V \implies x \overset{\star}{\mapsto} Q_V$ for any V. Unifying with $(*)$ we can conclude $x \overset{\star}{\mapsto} \bot$. By Lemma-Not$_2$, we can conclude that $\neg x \overset{\star}{\mapsto} \top$ as required.

And Cases:
There are two inductive cases for And. One where $x \wedge y \Downarrow \texttt{False}$ and one for $x \wedge y \Downarrow \texttt{True}$. Our inductive hypotheses are always:

$$\dfrac{x \Downarrow V}{x \overset{\star}{\mapsto} Q_V}IH_1 \quad \dfrac{y \Downarrow V}{y \overset{\star}{\mapsto} Q_V}IH_2$$

Inductive Case 1. We know that $x \wedge y \Downarrow \texttt{False}$. We must show that $x \wedge y \overset{\star}{\mapsto} \bot$. There are two ways that $x \wedge y \Downarrow \texttt{False}$ (our assumption) could be true. One is via rule And$_1$ where $x \Downarrow \texttt{False}$ $(*)$, and the other is via rule And$_2$, where $x \Downarrow \texttt{True}$ $(**)$ and $y \Downarrow \texttt{False}$ $(***)$.
The first case is easily proven:

$$\dfrac{\dfrac{\dfrac{}{x \Downarrow \texttt{False}}(*)}{x \overset{\star}{\mapsto} \bot}\,IH_1}{x \wedge y \overset{\star}{\mapsto} \bot}\,\textsc{Lemma-And}_1$$

The second case requires a little more fiddling:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{x \Downarrow \texttt{True}}(**)}{x \overset{\star}{\mapsto} \top}\,IH_1}{x \wedge y \overset{\star}{\mapsto} y}\,\textsc{Lemma-And}_2 \qquad \dfrac{\dfrac{}{y \Downarrow \texttt{False}}(***)}{y \overset{\star}{\mapsto} \bot}\,IH_2}{x \wedge y \overset{\star}{\mapsto} \bot}\,\textsc{Transitive}$$

Inductive Case 2

The only way for $x \wedge y \Downarrow \texttt{True}$ to hold is if both $x \Downarrow \texttt{True}$ $(*)$ and $y \Downarrow \texttt{True}$ $(**)$ (from rule $\textsc{And}_2$).

We have to show that $x \wedge y \overset{\star}{\mapsto} \top$, as shown:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{x \Downarrow \texttt{True}}(*)}{x \overset{\star}{\mapsto} \top}\,IH_1}{x \wedge y \overset{\star}{\mapsto} y}\,\textsc{Lemma-And}_2 \qquad \dfrac{\dfrac{}{y \Downarrow \texttt{True}}(**)}{y \overset{\star}{\mapsto} \top}\,IH_2}{x \wedge y \overset{\star}{\mapsto} \top}\,\textsc{Transitive}$$

Thus, by induction, we have shown that $E \Downarrow V \implies E \mapsto Q_V$ □

(c) Suppose we wanted to add quantifiers and variables to our logic language:

$$\frac{e\ \textbf{Prop}}{\exists(x.e)\ \textbf{Prop}} \qquad \frac{e\ \textbf{Prop}}{\forall(x.e)\ \textbf{Prop}} \qquad \frac{x\ \textbf{\textit{is a variable name}}}{x\ \textbf{Prop}}$$

It is no longer as easy to write static semantics for this language, as the formula may not be decidable, however we can still write static checkers that perform analysis on a more superficial level. Write a static semantics judgement for this language, written $\vdash e$ ***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 (environment) of variable names, denoted Γ.

$$\frac{v \in \Gamma}{\Gamma \vdash v\ \textbf{\textit{Ok}}}\textsc{LookupEnv} \qquad \frac{\Gamma \cup \{v\} \vdash x\ \textbf{\textit{Ok}}}{\Gamma \vdash \exists v.\ x\ \textbf{\textit{Ok}}}\textsc{Exists} \qquad \frac{\Gamma \cup \{v\} \vdash x\ \textbf{\textit{Ok}}}{\Gamma \vdash \forall v.\ x\ \textbf{\textit{Ok}}}\textsc{Forall}$$

$$\frac{\Gamma \vdash x\ \textbf{\textit{Ok}}}{\Gamma \vdash \neg x\ \textbf{\textit{Ok}}}\textsc{Not} \qquad \frac{\Gamma \vdash x\ \textbf{\textit{Ok}} \quad \Gamma \vdash y\ \textbf{\textit{Ok}}}{\Gamma \vdash x \wedge y\ \textbf{\textit{Ok}}}\textsc{And}$$

2. **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:

$$\frac{x \in \mathbb{N}}{x\ \textbf{\textit{Symbol}}} \qquad \frac{x \in \{+, -, /, *\}}{x\ \textbf{\textit{Symbol}}}$$

$$\frac{}{\epsilon\ \textit{RPN}} \qquad \frac{x\ \textit{Symbol} \quad xs\ \textit{RPN}}{x\ xs\ \textit{RPN}}$$

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

i. Write some static semantics inference rules for a judgement $\vdash e\ \textit{Ok}$ 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:

$$\frac{}{1 \vdash \epsilon\ \textit{Ok}}\textsc{Empty}$$

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

$$\frac{x \in \mathbb{N} \quad (n+1) \vdash xs\ \textit{Ok}}{n \vdash x\ xs\ \textit{Ok}}\textsc{Num}$$

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

$$\frac{x \in \{+, -, *, /\} \quad (n+1) \vdash xs\ \textit{Ok}}{(n+2) \vdash x\ xs\ \textit{Ok}}\textsc{Op}$$

ii. Show that $\vdash 1\ 3\ 2\ +\ -\ \textit{Ok}$.

Solution:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{1 \vdash \epsilon\ \textit{Ok}}\textsc{Empty}}{2 \vdash -\ \textit{Ok}}\textsc{Op}}{3 \vdash +-\ \textit{Ok}}\textsc{Op}}{2 \vdash 2\ +\ -\ \textit{Ok}}\textsc{Num}}{1 \vdash 3\ 2\ +\ -\ \textit{Ok}}\textsc{Num}}{0 \vdash 1\ 3\ 2\ +\ -\ \textit{Ok}}\textsc{Num}.$$

(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. Identify the set of evaluable expressions E, and the set of result values V.

Solution: The set E is simply the set of all e such that $e\ \textit{RPN}$. V is the set of all stacks, defined as follows:

$$\frac{}{\circ\ \textit{Stack}} \qquad \frac{x \in \mathbb{N} \quad xs\ \textit{Stack}}{x \triangleright xs\ \textit{Stack}}$$

ii. Define a relation $\Downarrow : E \times V$ which evaluates RPN programs.

Solution: The empty program evaluates to the empty stack:

$$\frac{}{\epsilon \Downarrow \circ}\textsc{BS-Empty}$$

The non-empty program ending in an operator performs the operation on the

stack resulting from the previous symbols:

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

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

$$\frac{xs \Downarrow s \quad x \in \mathbb{Z}}{xs\ x \Downarrow x \triangleright s}\text{BS-Num}$$

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

Our states Q 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 $n \triangleright \circ \vdash \epsilon$.

i. Define a relation $\mapsto : Q \times Q$, which evaluates one step of the calculation.

Solution: Numbers simply push to the stack:

$$\frac{x \in \mathbb{N}}{s \vdash x\ xs \mapsto x \triangleright s \vdash xs}\text{SS-Num}$$

Operations simply pop two elements from the stack and operate on them.

$$\frac{x \in \{+, -, *, /\}}{a \triangleright b \triangleright s \vdash x\ xs \mapsto a\ x\ b \triangleright s \vdash xs}\text{SS-Op}$$

(d) Now we will prove small step and big step equivalent.

i. Show using rule induction on $\Downarrow$ that, for all expressions e, if $e \Downarrow v$ then $Q_e \overset{!}{\mapsto} Q_v$ (where Q_e and Q_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):

$$\frac{\circ \vdash xs \overset{\star}{\mapsto} v \vdash \epsilon}{\circ \vdash xs\ x \overset{\star}{\mapsto} v \vdash x}\text{Append}$$

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

Solution: We shall define Q_e as $\circ \vdash e$, and Q_v as $v \vdash \epsilon$.

Base case (from rule BS-Empty*).* Where $e = \epsilon$ and $v = \circ$. We must show $\circ \vdash \epsilon \overset{\star}{\mapsto} \circ \vdash \epsilon$, trivially true by rule Refl*.
Inductive case (from rule BS-Num). Knowing the following:

- $v = x \triangleright v'$
- $e = xs\ x$
- $x \in \mathbb{N}$
- $xs \Downarrow v' \quad (*)$

And equipped with the inductive hypothesis that $xs \Downarrow v' \implies \circ \vdash xs \overset{\star}{\mapsto} v' \vdash \epsilon$, we must show that $\circ \vdash xs\ x \overset{\star}{\mapsto} x \triangleright v' \vdash \epsilon$.

$$\dfrac{\dfrac{\dfrac{\dfrac{}{xs \Downarrow v'}(*)}{\circ \vdash xs \overset{\star}{\mapsto} v' \vdash \epsilon}IH}{\circ \vdash xs\ x \overset{\star}{\mapsto} v' \vdash x}\textsc{Append} \quad \dfrac{}{v' \vdash x \mapsto x \triangleright v' \vdash \epsilon}\textsc{SS-Num}}{\circ \vdash xs\ x \overset{\star}{\mapsto} x \triangleright v' \vdash \epsilon}\textsc{Trans}^*$$

Inductive case (from rule BS-Op). Knowing the following:

- $v = a\ x\ b \triangleright v'$
- $e = xs\ x$
- $x \in \{+, -, *, /\}$
- $xs \Downarrow a \triangleright b \triangleright v' \quad (*)$

And armed with the inductive hypothesis that $xs \Downarrow a \triangleright b \triangleright v' \implies \circ \vdash xs \overset{\star}{\mapsto} a \triangleright b \triangleright v' \vdash \epsilon$, we must show that $\circ \vdash xs\ x \overset{\star}{\mapsto} (a\ x\ b) \triangleright v' \vdash \epsilon$.

$$\dfrac{\dfrac{\dfrac{\dfrac{}{xs \Downarrow a \triangleright b \triangleright v'}(*)}{\circ \vdash xs \overset{\star}{\mapsto} a \triangleright b \triangleright v' \vdash \epsilon}IH}{\circ \vdash xs\ x \overset{\star}{\mapsto} a \triangleright b \triangleright v' \vdash x}\textsc{Append} \quad \dfrac{}{a \triangleright b \triangleright v' \vdash x \mapsto (a\ x\ b) \triangleright v' \vdash \epsilon}\textsc{SS-Num}}{\circ \vdash xs\ x \overset{\star}{\mapsto} (a\ x\ b) \triangleright v' \vdash \epsilon}\textsc{Trans}^*$$

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 by structural induction on xs, however we will strengthen our proof goal to the more general rule below:

$$\dfrac{s \vdash xs \overset{\star}{\mapsto} v \vdash \epsilon}{s \vdash xs\ x \overset{\star}{\mapsto} v \vdash x}\textsc{Append'}$$

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

Base case (when $xs = \epsilon$). We must show, assuming $s \vdash \epsilon \overset{\star}{\mapsto} v \vdash \epsilon$, then $s \vdash x \overset{\star}{\mapsto} v \vdash x$. Seeing as there is no state q such that $s \vdash \epsilon \mapsto q$, the only way $s \vdash \epsilon \overset{\star}{\mapsto} v \vdash \epsilon$ (our assumption) could be true is if it was so by rule Refl*, and hence $v = s$. Thus we can conclude that $s \vdash x \overset{\star}{\mapsto} s \vdash x$ by rule Refl*.
Inductive case (when $xs = a\ as$). We must show, assuming $s \vdash a\ as \overset{\star}{\mapsto} v \vdash \epsilon (*)$, that $s \vdash a\ as\ x \overset{\star}{\mapsto} v \vdash x$. Our inductive hypothesis is that $s' \vdash as \overset{\star}{\mapsto} v' \vdash \epsilon \implies s' \vdash as\ x \overset{\star}{\mapsto} v' \vdash x$. There are two cases. One where a is a number and one where a is an operator. We will deal with $a \in \mathbb{N}$ first.

$$\dfrac{\dfrac{}{s \vdash a\ as\ x \mapsto a \triangleright s \vdash as\ x}\textsc{SS-Num} \quad \dfrac{\dfrac{}{a \triangleright s \vdash as \overset{\star}{\mapsto} v \vdash \epsilon}(*)}{a \triangleright s \vdash as\ x \overset{\star}{\mapsto} v \vdash x}\textsc{IH}}{s \vdash a\ as\ x \overset{\star}{\mapsto} v \vdash x}\textsc{Trans}_2^*$$

The rule $\textsc{Trans}_2^*$ is trivially derivable from Transitive, Trans*, and Refl*. The case where a is an operator is very similar, just with different stacks. □

ii. Show using rule induction on $\overset{\star}{\mapsto}$ that, for all expressions e and values v, if $Q_e \overset{\star}{\mapsto} Q_v$

then $e \Downarrow v$. It may be useful to assume the inverse of APPEND:

$$\frac{\circ \vdash xs\ x \overset{\star}{\mapsto} v \vdash x}{\circ \vdash xs \overset{\star}{\mapsto} v \vdash \epsilon}\ \textsc{Append}^{-1}$$

Solution: We must show that $\circ \vdash e \overset{\star}{\mapsto} v \vdash \epsilon$ implies $e \Downarrow v$

REFL* *case (where $e = \epsilon$ and $v = \circ$).* We must show that $\epsilon \Downarrow \circ$, trivially by rule BS-EMPTY.
TRANS* case (where $e = xs\ x$), we must show that $\circ \vdash xs\ x \overset{\star}{\mapsto} v \vdash \epsilon$ $(*)$ implies $xs\ x \Downarrow v$. We have the inductive hypothesis that $\circ \vdash xs \overset{\star}{\mapsto} v' \vdash \epsilon$ implies $xs \Downarrow v'$. We proceed by case distinction on x.
In the case where $x \in \mathbb{N}$, we can rewrite our assumption $(*)$ as $\circ \vdash xs\ x \overset{\star}{\mapsto} x \triangleright v' \vdash \epsilon$. We can begin deducing facts from our assumption (this is only possible because the rules here are unambiguous - if there was another way of deriving this assumption, this approach would be invalid):

$$\frac{\dfrac{\vdots}{\circ \vdash xs\ x \overset{\star}{\mapsto} v' \vdash x}\ \textit{Deduction} \qquad \dfrac{}{v' \vdash x \mapsto x \triangleright v' \vdash \epsilon}\ \textsc{SS-Num}}{\circ \vdash xs\ x \overset{\star}{\mapsto} x \triangleright v' \vdash \epsilon}\ \textsc{Trans}*$$

Hence, we can conclude from "reverse-engineering" our assumption here that $\circ \vdash xs\ x \overset{\star}{\mapsto} v' \vdash x$.
Now we can try to prove our goal:

$$\frac{\dfrac{\dfrac{\dfrac{}{\circ \vdash xs\ x \overset{\star}{\mapsto} v' \vdash x}\ \textit{Deduction}}{\circ \vdash xs \overset{\star}{\mapsto} v' \vdash \epsilon}\ \textsc{Append}^{-1}}{xs \Downarrow v'}\ \textit{IH} \qquad \dfrac{}{x \in \mathbb{N}}}{xs\ x \Downarrow x \triangleright v'}$$

The case where x is an operator is very similar, only with different stacks. □

(e) Show that your static semantics defined in (a) ensure that the program will evaluate to a value. That is, show that $\vdash e\ \textbf{\textit{Ok}} \implies e \Downarrow s$, for some s. You may find it helpful to generalise your proof goal before beginning induction.

Solution: We shall show that $\vdash e\ \textbf{\textit{Ok}} \implies \circ \vdash e \overset{\star}{\mapsto} s \vdash \epsilon$ for some s, which is equivalent to our goal above, as we have shown the isomorphism between big step and small step semantics.
We shall start by generalising our goal to the more flexible $n \vdash e\ \textbf{\textit{Ok}} \implies v_1 \triangleright v_2 \triangleright \cdots \triangleright v_n \triangleright \circ \vdash e \overset{\star}{\mapsto} s \vdash \epsilon$ for some s and all v_1 through v_n. This trivially implies our first goal by setting $n = 0$.

Base case (where $e = \epsilon$). We know $n \vdash \epsilon\ \textbf{\textit{Ok}}$, hence we can deduce that $n = 1$ from rule EMPTY. Hence we must show that $v_1 \triangleright \circ \vdash \epsilon \overset{\star}{\mapsto} s \vdash \epsilon$, for some s. This is trivially true by rule REFL*, where $s = v_1 \triangleright \circ$.
Inductive case (where $e = x\ xs$ and $x \in \mathbb{N}$) We know that $n \vdash x\ xs\ \textbf{\textit{Ok}}$, and we can deduce by inversion of NUM that $(n+1) \vdash xs\ \textbf{\textit{Ok}}$ $(*)$. Our inductive hypothesis is that $(n+1) \vdash xs\ \textbf{\textit{Ok}} \implies v'_1 \triangleright \cdots \triangleright v'_n \triangleright v'_{n+1} \triangleright \circ \vdash xs \overset{\star}{\mapsto} s' \vdash \epsilon$ for some s' and all v'_0 through v'_n.

We can show our goal by setting our goal's $s = s'$, the IH's $v_1' = x$ and $v_i' = v_{i-1}$ for all subsequent i.

$$\dfrac{\dfrac{}{v_1 \triangleright \cdots \triangleright v_n \triangleright \circ \vdash x\ xs \mapsto x \triangleright v_1 \triangleright \cdots \triangleright v_n \triangleright \circ \vdash xs}\text{SS-NUM} \quad \dfrac{\dfrac{}{(n+1) \vdash xs\ \textbf{\textit{Ok}}}(*)}{x \triangleright v_1 \triangleright \cdots \triangleright v_n \triangleright \circ \vdash xs \overset{\star}{\mapsto} s' \vdash \epsilon}IH}{v_1 \triangleright v_2 \triangleright \cdots \triangleright v_n \triangleright \circ \vdash x\ xs \overset{\star}{\mapsto} s' \vdash \epsilon}\text{TRANS}_2^*$$

The case where x is an operator is very similar, just different stacks. □

3. **Dynamic semantics of a simple robot control language** A robot moves along a grid according to a simple program. The program consists of a possibly empty sequence of the commands move and turn, separated by semicolons. Initially, the robot faces east and starts at the grid coordinates (0,0). The command turn causes the robot to turn 90 degrees counter-clockwise, and move to move one unit in the direction it is facing.

   (a) **Small step semantics:** devise a set of small-step semantics rules for this language. This means determining answers to the following questions:
   - What is the set of states?
   - Which of those states are final states, and which are initial states?
   - What transitions exist between those states?

   (b) **Big step semantics:** what would a suitable a set of big-step evaluation rules for this language look like? In particular:
   - What is the set of evaluable expressions?
   - What is the set of values?
   - How do evaluable expressions evaluate to those values?

Solution:

- The state can be described as a triple of current position of the robot in the 2D plane, encoded as a vector in Z^2, the direction it is facing, which can also be encoded as a vector in Z^2, and the sequence of instructions not yet executed.
- Initially, the robot faces east $\begin{pmatrix}1\\0\end{pmatrix}$, and is at position $\begin{pmatrix}0\\0\end{pmatrix}$. The final states are those where the sequence of instructions is empty.
- Transitions:

$$\frac{}{(pos, dir, \texttt{move}; ins) \mapsto (pos + dir, dir, ins)}$$

$$\frac{}{(pos, dir, \texttt{turn}; ins) \mapsto (pos, \begin{pmatrix}0 & -1\\1 & 0\end{pmatrix} * dir, ins)}$$

For a big step semantics, we can model the set of evaluatable expressions in the same way as we did for the small step semantics. The values are just pairs of position and direction. In this case, the big step semantics is not simpler than the small step semantics. In fact, we have to add one rule, which explicitly maps the empty instruction sequence to a final state.

$$\frac{}{(pos, dir, \circ) \Downarrow (pos, dir)}(empty\ sequence)$$

$$\frac{(pos + dir, dir, ins) \Downarrow (pos', dir')}{(pos, dir, \texttt{move}; ins) \Downarrow (pos', dir')}(move)$$

$$\frac{(pos, \begin{pmatrix}0 & -1\\1 & 0\end{pmatrix} * dir, ins) \Downarrow (pos', dir')}{(pos, dir, \texttt{turn}; ins) \Downarrow (pos', dir')}(turn)$$

There are many other solutions to this problem. We could store the direction as *east*, *west* and so on, and provide rules how `turn` affects the direction (i.e., from east to south, south to west, and so on.

4. **TinyC** The semantics we defined for TinyC in the lecture is a call-by-value semantics. That is, we only pass the value to the function.

   Extend TinyC such that the type of a function parameter in a function declaration can be preceeded by the keyword `var` so that this variable is passed by reference. For example, evaluating the following TinyC program would result in the value 20:

```
int swap (var int a; var int b) {
  int tmp = a;
  a = b;
  b = tmp;
  return 0;
}

{
  int x = 10;
  int y = 20;
  swap (x, y);
  return (x);
}
```

   - Which (if any) adjustments to the static semantics are necessary?
   - How can you model this in the operational dynamic semantics?

   Solution: Function calls have to be checked to ensure that only variable names, not general expressions, can be at the position of call-by-reference parameters. If every reference passed into a function has to be distinct, aliasing can be statically excluded, and the dynamic semantics just needs to ensure that, upon returning from the function call, the value of the references variable is updated.

   However, if there is no such restriction, references have to be modelled in the dynamic semantics using a mapping from location to value.