Chapter 2

Semantics of IMP [Lecture 2–4]

Version of February 17, 2021

Summary We summarize the big-step semantics of IMP, and consider some simple examples of reasoning about semantic equivalences. We then show totality of (arithmetic and boolean) expression evaluation, and determinism of both expression evaluation and command execution. Finally, we formalize and show equivalence of the big-step semantics to a small-step semantics.

2.1 Big-step IMP semantics and simple reasoning

2.1.1 Recapitulation of the IMP language

We consider IMP more or less as defined in *FSoPL*, though with a short-circuit semantics of conjunction and disjunction, to make the corresponding proof cases a bit more interesting.

Syntax The abstract syntax of the language is as follows. Note that we distinguish syntactically between numbers $n \in \mathbf{Z}$ and numerals $\overline{n} \in \mathbf{AExp}$.

$$
\begin{array}{rcl}
n & \in & \mathbf{Z} = \{\ldots, -1, 0, 1, 2, \ldots\} \\
X & \in & \mathbf{Loc} = \{x, y, z, \mathit{sum}, \mathit{count}, \ldots\} \\
\\
a & ::= & \overline{n} \mid X \mid a_0 + a_1 \mid a_0 - a_1 \mid a_0 \times a_1 \\
t & ::= & \mathbf{true} \mid \mathbf{false} \\
b & ::= & t \mid a_0 = a_1 \mid a_0 \leq a_1 \mid \neg b_0 \mid b_0 \wedge b_1 \\
c & ::= & \mathbf{skip} \mid X := a \mid c_0; c_1 \mid \mathbf{if}\ b\ \mathbf{then}\ c_0\ \mathbf{else}\ c_1 \mid \mathbf{while}\ b\ \mathbf{do}\ c_0
\end{array}
$$

For the sake of illustration, we have omitted disjunction as a separate syntactic construct, opting instead to define it as syntactic sugar: $b_0 \vee b_1 \stackrel{\text{def}}{\equiv} \neg(\neg b_0 \wedge \neg b_1)$.

Semantics Like in *FSoPL*, we use $\sigma \in \Sigma = \mathbf{Loc} \to \mathbf{Z}$ to range over stores. The big-step semantics is then also essentially as in *FSoPL*, modulo notational details. In particular,

Judgment $\boxed{\langle a, \sigma\rangle \downarrow n}$:

$$\text{EA-Num} : \frac{}{\langle \overline{n}, \sigma\rangle \downarrow n} \qquad \text{EA-Loc} : \frac{}{\langle X, \sigma\rangle \downarrow \sigma(X)} \qquad \text{EA-Plus} : \frac{\langle a_0, \sigma\rangle \downarrow n_0 \qquad \langle a_1, \sigma\rangle \downarrow n_1}{\langle a_0 \ + \ a_1, \sigma\rangle \downarrow n_0 + n_1}$$

$$\text{EA-Minus} : \frac{\langle a_0, \sigma\rangle \downarrow n_0 \qquad \langle a_1, \sigma\rangle \downarrow n_1}{\langle a_0 \ - \ a_1, \sigma\rangle \downarrow n_0 - n_1} \qquad \text{EA-Times} : \frac{\langle a_0, \sigma\rangle \downarrow n_0 \qquad \langle a_1, \sigma\rangle \downarrow n_1}{\langle a_0 \ \times \ a_1, \sigma\rangle \downarrow n_0 \times n_1}$$

Judgment $\boxed{\langle b, \sigma\rangle \downarrow t}$:

$$\text{EB-Cst} : \frac{}{\langle t, \sigma\rangle \downarrow t}$$

$$\text{EB-EqT} : \frac{\langle a_0, \sigma\rangle \downarrow n \qquad \langle a_1, \sigma\rangle \downarrow n}{\langle a_0 = a_1, \sigma\rangle \downarrow \textbf{true}} \qquad \text{EB-EqF} : \frac{\langle a_0, \sigma\rangle \downarrow n_0 \qquad \langle a_1, \sigma\rangle \downarrow n_1}{\langle a_0 = a_1, \sigma\rangle \downarrow \textbf{false}} \, (n_0 \neq n_1)$$

$$\text{EB-LeqT} : \frac{\langle a_0, \sigma\rangle \downarrow n_0 \qquad \langle a_1, \sigma\rangle \downarrow n_1}{\langle a_0 \leq a_1, \sigma\rangle \downarrow \textbf{true}} \, (n_0 \leq n_1) \quad \text{EB-LeqF} : \frac{\langle a_0, \sigma\rangle \downarrow n_0 \qquad \langle a_1, \sigma\rangle \downarrow n_1}{\langle a_0 \leq a_1, \sigma\rangle \downarrow \textbf{false}} \, (n_0 > n_1)$$

$$\text{EB-NegT} : \frac{\langle b_0, \sigma\rangle \downarrow \textbf{true}}{\langle \neg b_0, \sigma\rangle \downarrow \textbf{false}} \qquad \text{EB-NegF} : \frac{\langle b_0, \sigma\rangle \downarrow \textbf{false}}{\langle \neg b_0, \sigma\rangle \downarrow \textbf{true}}$$

$$\text{EB-AndT} : \frac{\langle b_0, \sigma\rangle \downarrow \textbf{true} \qquad \langle b_1, \sigma\rangle \downarrow t}{\langle b_0 \wedge b_1, \sigma\rangle \downarrow t} \qquad \text{EB-AndF} : \frac{\langle b_0, \sigma\rangle \downarrow \textbf{false}}{\langle b_0 \wedge b_1, \sigma\rangle \downarrow \textbf{false}}$$

Judgment $\boxed{\langle c, \sigma\rangle \downarrow \sigma'}$:

$$\text{EC-Skip} : \frac{}{\langle \textbf{skip}, \sigma\rangle \downarrow \sigma} \qquad \text{EC-Assign} : \frac{\langle a, \sigma\rangle \downarrow n}{\langle X := a, \sigma\rangle \downarrow \sigma[X \mapsto n]}$$

$$\text{EC-Seq} : \frac{\langle c_0, \sigma\rangle \downarrow \sigma'' \qquad \langle c_1, \sigma''\rangle \downarrow \sigma'}{\langle c_0; c_1, \sigma\rangle \downarrow \sigma'}$$

$$\text{EC-IfT} : \frac{\langle b, \sigma\rangle \downarrow \textbf{true} \qquad \langle c_0, \sigma\rangle \downarrow \sigma'}{\langle \textbf{if } b \textbf{ then } c_0 \textbf{ else } c_1, \sigma\rangle \downarrow \sigma'} \qquad \text{EC-IfF} : \frac{\langle b, \sigma\rangle \downarrow \textbf{false} \qquad \langle c_1, \sigma\rangle \downarrow \sigma'}{\langle \textbf{if } b \textbf{ then } c_0 \textbf{ else } c_1, \sigma\rangle \downarrow \sigma'}$$

$$\text{EC-WhileF} : \frac{\langle b, \sigma\rangle \downarrow \textbf{false}}{\langle \textbf{while } b \textbf{ do } c_0, \sigma\rangle \downarrow \sigma}$$

$$\text{EC-WhileT} : \frac{\langle b, \sigma\rangle \downarrow \textbf{true} \qquad \langle c_0, \sigma\rangle \downarrow \sigma'' \qquad \langle \textbf{while } b \textbf{ do } c_0, \sigma''\rangle \downarrow \sigma'}{\langle \textbf{while } b \textbf{ do } c_0, \sigma\rangle \downarrow \sigma'}$$

Figure 2.1: Big-step semantics of IMP

we use $\downarrow$ instead of $\rightarrow$ for the evaluation judgments, and we write $\sigma[X \mapsto n]$, rather than $\sigma[n/X]$, for the updated store that maps X to n, and all other locations Y to $\sigma(Y)$. For easy reference, the judgments and rules are given in Figure 2.1.

2.1.2 Reasoning about semantic equivalence

With the formal semantics of expressions and commands settled, we can also define the notions of *semantic equivalence* of syntactic phrases. Following *FSoPL*, we introduce the following notation:

$$\begin{aligned} a \sim a' &\equiv \forall \sigma, n.\, \langle a, \sigma\rangle \downarrow n \Leftrightarrow \langle a', \sigma\rangle \downarrow n \\ b \sim b' &\equiv \forall \sigma, t.\, \langle b, \sigma\rangle \downarrow t \Leftrightarrow \langle b', \sigma\rangle \downarrow t \\ c \sim c' &\equiv \forall \sigma, \sigma'.\, \langle c, \sigma\rangle \downarrow \sigma' \Leftrightarrow \langle c', \sigma\rangle \downarrow \sigma' \end{aligned}$$

For instance, arithmetic expressions a and a' are considered semantically equivalent whenever, in all stores σ, if a can evaluate to an integer n, then so can a', and vice versa. This relationship is sometimes informally and *imprecisely* paraphrased as "a and a' always evaluate to the same result"; but such a paraphrase only makes sense if we have already established that, in every store, any expression must evaluate to exactly one number. While this (as we will show) is indeed the case for arithmetic expressions in IMP, it wouldn't necessarily hold in a larger language, and so it is important that we stick to the formal definition of semantic equivalence in discussion and proofs.

To show that two expressions (or commands) are equivalent, we consider how the left-hand-side of the $\Leftrightarrow$ could have been derived by the semantic rules, and check that the right-hand-side is then also derivable; and then we establish the same property from right to left. Sometimes this is straightforward: for example, to verify the expectable equivalence $a_0 + a_1 \sim a_1 + a_0$, we just observe that any derivation of $\langle a_0 + a_1, \sigma\rangle \downarrow n$ must have the shape

$$\textsc{ea-Plus}\ \frac{\begin{array}{cc}\vdots & \vdots \\ \langle a_0, \sigma\rangle \downarrow n_0 & \langle a_1, \sigma\rangle \downarrow n_1\end{array}}{\langle a_0 + a_1, \sigma\rangle \downarrow n_0 + n_1},$$

where $n = n_0 + n_1$. Putting together the two subderivations in the opposite order, and exploiting that, *for mathematical integers*, $n_0 + n_1 = n_1 + n_0$, we can easily construct a corresponding derivation of $\langle a_1 + a_0, \sigma\rangle \downarrow n$. The converse direction is essentially identical. However, when more than one rule may apply in a situation, we have to proceed more systematically. To illustrate, let us show the following simple but useful fact:

Theorem 2.1 *For all boolean expressions b, we have $b \wedge \mathbf{true} \sim b$.*

Proof. Let any b be given, and let σ and t be an arbitrary store and truth value, respectively; we must show that $\langle b \wedge \mathbf{true}, \sigma\rangle \downarrow t \Leftrightarrow \langle b, \sigma\rangle \downarrow t$. Recalling that $\Leftrightarrow$ is an abbreviation for two separate implications, we first show that, if $\langle b \wedge \mathbf{true}, \sigma\rangle \downarrow t$ by some derivation $\mathcal{E}$, then there exists a derivation $\mathcal{E}'$ of $\langle b, \sigma\rangle \downarrow t$.

Note that we know nothing about b, other than that it is a well formed boolean expression; thus, it might at first seem that we need to separately consider all the 5

possible shapes that b could have. Fortunately, this is not necessary: looking at the EB-*???* rules in the semantics, there are only two whose conclusion is an $\wedge$-expression (EB-AndT and EB-AndF), and hence there are only two overall shapes that the derivation $\mathcal{E}$ can take:

- Case $\mathcal{E} = \text{EB-True}\dfrac{\begin{array}{cc}\mathcal{E}_0 & \mathcal{E}_1\\ \langle b,\sigma\rangle \downarrow \mathbf{true} & \langle \mathbf{true},\sigma\rangle \downarrow t\end{array}}{\langle b \wedge \mathbf{true},\sigma\rangle \downarrow t}$, for some subderivations $\mathcal{E}_0$ and $\mathcal{E}_1$. In this case, again looking at the rules, we see that the subderivation $\mathcal{E}_1$ could only have used the rule EB-Cst (with t in the rule taken as **true**), so we must further have
$$\mathcal{E}_1 = \text{EB-Cst}\frac{}{\langle \mathbf{true},\sigma\rangle \downarrow \mathbf{true}}\,.$$
But this means that we must also have $t = \mathbf{true}$ in the original derivation $\mathcal{E}$; and thus we can simply use $\mathcal{E}_0$ as the required derivation $\mathcal{E}'$ of $\langle b,\sigma\rangle \downarrow t$.

- Case $\mathcal{E} = \text{EB-False}\dfrac{\begin{array}{c}\mathcal{E}_0\\ \langle b,\sigma\rangle \downarrow \mathbf{false}\end{array}}{\langle b \wedge \mathbf{true},\sigma\rangle \downarrow \mathbf{false}}$, for some subderivation $\mathcal{E}_0$. In this case, we evidently have $t = \mathbf{false}$, and so again, we can directly use $\mathcal{E}_0$ as $\mathcal{E}'$.

This concludes the $\Rightarrow$ direction of the proof, but we also need to show $\Leftarrow$, i.e., that if we have a derivation $\mathcal{E}'$ of $\langle b,\sigma\rangle \downarrow t$, then we can construct an $\mathcal{E}$ of $\langle b \wedge \mathbf{true},\sigma\rangle \downarrow t$. Here, because we know nothing about the shape of b, we can no longer usefully split into cases on how the derivation $\mathcal{E}'$ could look. Instead, however, we can exploit that there are only two possibilities for the result t:

- Case $t = \mathbf{true}$. Then $\mathcal{E}'$ is actually a derivation of $\langle b,\sigma\rangle \downarrow \mathbf{true}$, and so we can use EB-AndT and EB-Cst to construct $\mathcal{E}$:
$$\mathcal{E} = \frac{\begin{array}{cc}\mathcal{E}' & \\ \langle b,\sigma\rangle \downarrow \mathbf{true} & \dfrac{}{\langle \mathbf{true},\sigma\rangle \downarrow \mathbf{true}}\end{array}}{\langle b \wedge \mathbf{true},\sigma\rangle \downarrow \mathbf{true}}$$
(This also happens to be the *only* way we could construct $\mathcal{E}$, but that observation is *not* relevant to the proof; we just need to find *at least one* usable derivation.)

- Case $t = \mathbf{false}$. Here our derivation $\mathcal{E}'$ shows $\langle b,\sigma\rangle \downarrow \mathbf{false}$, and so we can use it with EB-AndF to build:
$$\mathcal{E} = \frac{\begin{array}{c}\mathcal{E}'\\ \langle b,\sigma\rangle \downarrow \mathbf{false}\end{array}}{\langle b \wedge \mathbf{true},\sigma\rangle \downarrow \mathbf{false}}$$

■

Note that, at no point in the proof, did we implicitly or explicitly rely on the given b evaluating to some unique boolean result; rather, the proof in both directions established (with somewhat different arguments) that *if* one of the boolean expressions $b \wedge \mathbf{true}$ and b *can* evaluate to some truth value t, then the other can also evaluate to that value – and emphatically *not* that "both $b \wedge \mathbf{true}$ and b must evaluate to the same t".

A similar, but slightly more involved, analysis–synthesis argument can be used to show the following semantic equivalence between commands, often called "loop unrolling":

Theorem 2.2 *For all boolean expressions b and commands c_0, we have the equivalence* $\underbrace{\textbf{while } b \textbf{ do } c_0}_{c} \sim \underbrace{\textbf{if } b \textbf{ then } (c_0; \textbf{while } b \textbf{ do } c_0) \textbf{ else skip}}_{c'}$.

Proof. (See also Proposition 2.8 in *FSoPL*, keeping in mind the differences in notation.) ($\Rightarrow$ direction): We assume that we have a derivation $\mathcal{E}$ of $\langle c, \sigma\rangle \downarrow \sigma'$, and we must construct an $\mathcal{E}'$ of $\langle c', \sigma\rangle \downarrow \sigma'$. Looking at the execution rules for **while**-commands, we see that $\mathcal{E}$ must have one of the following two shapes:

- Case $\mathcal{E} = \dfrac{\overset{\mathcal{E}_0}{\langle b, \sigma\rangle \downarrow \textbf{false}}}{\langle \textbf{while } b \textbf{ do } c_0, \sigma\rangle \downarrow \sigma}$, so $\sigma' = \sigma$. Then we can take:

$$\mathcal{E}' = \frac{\overset{\mathcal{E}_0}{\langle b, \sigma\rangle \downarrow \textbf{false}} \qquad \overline{\langle \textbf{skip}, \sigma\rangle \downarrow \sigma}}{\langle \textbf{if } b \textbf{ then } (c_0; c) \textbf{ else skip}, \sigma\rangle \downarrow \sigma}.$$

- Case $\mathcal{E} = \dfrac{\overset{\mathcal{E}_0}{\langle b, \sigma\rangle \downarrow \textbf{true}} \qquad \overset{\mathcal{E}_1}{\langle c_0, \sigma\rangle \downarrow \sigma''} \qquad \overset{\mathcal{E}_2}{\langle c, \sigma''\rangle \downarrow \sigma'}}{\langle \textbf{while } b \textbf{ do } c_0, \sigma\rangle \downarrow \sigma'}$ (for some σ'').

  Here we can build $\mathcal{E}'$ as follows:

$$\mathcal{E}' = \frac{\overset{\mathcal{E}_0}{\langle b, \sigma\rangle \downarrow \textbf{true}} \qquad \dfrac{\overset{\mathcal{E}_1}{\langle c_0, \sigma\rangle \downarrow \sigma''} \qquad \overset{\mathcal{E}_2}{\langle c, \sigma''\rangle \downarrow \sigma'}}{\langle c_0; c, \sigma\rangle \downarrow \sigma'}}{\langle \textbf{if } b \textbf{ then } (c_0; c) \textbf{ else skip}, \sigma\rangle \downarrow \sigma'}.$$

(Note that we are formally splitting into cases based on which *rule* was used for $\mathcal{E}$, not on what *truth value* b evaluated to; the two notions merely happen to coincide here.)

($\Leftarrow$ direction): We now assume given an $\mathcal{E}'$ of $\langle c', \sigma\rangle \downarrow \sigma'$, and we must construct an $\mathcal{E}$ of $\langle c, \sigma\rangle \downarrow \sigma'$. Since c' is an **if**-command, $\mathcal{E}'$ must have one of the following two shapes:

- Case $\mathcal{E}' = \dfrac{\overset{\mathcal{E}'_0}{\langle b, \sigma\rangle \downarrow \textbf{true}} \qquad \overset{\mathcal{E}'_1}{\langle c_0; c, \sigma\rangle \downarrow \sigma'}}{\langle \textbf{if } b \textbf{ then } (c_0; c) \textbf{ else skip}, \sigma\rangle \downarrow \sigma'}$.

  Then, since there is only one execution rule for sequence-commands, we know that $\mathcal{E}'_1$ must itself have the following shape:

$$\mathcal{E}'_1 = \frac{\overset{\mathcal{E}'_{10}}{\langle c_0, \sigma\rangle \downarrow \sigma''} \qquad \overset{\mathcal{E}'_{11}}{\langle c, \sigma''\rangle \downarrow \sigma'}}{\langle c_0; c, \sigma\rangle \downarrow \sigma'} \text{ (for some } \sigma'').$$

  But then we can put the various subderivations together using EC-WhileT:

$$\mathcal{E} = \frac{\overset{\mathcal{E}'_0}{\langle b, \sigma\rangle \downarrow \textbf{true}} \qquad \overset{\mathcal{E}'_{10}}{\langle c_0, \sigma\rangle \downarrow \sigma''} \qquad \overset{\mathcal{E}'_{11}}{\langle c, \sigma''\rangle \downarrow \sigma'}}{\langle \textbf{while } b \textbf{ do } c_0, \sigma\rangle \downarrow \sigma'}$$

- Case $\mathcal{E}' = \dfrac{\overset{\mathcal{E}'_0}{\langle b, \sigma\rangle \downarrow \textbf{false}} \qquad \overset{\mathcal{E}'_1}{\langle \textbf{skip}, \sigma\rangle \downarrow \sigma'}}{\langle \textbf{if } b \textbf{ then } (c_0; c) \textbf{ else skip}, \sigma\rangle \downarrow \sigma'}$.

  Here, since there is only one execution rule for **skip**, $\mathcal{E}'_1$ must have used it, and therefore have the following shape:

  $$\mathcal{E}'_1 = \frac{}{\langle \textbf{skip}, \sigma\rangle \downarrow \sigma}\,,$$

  so, in particular, we know that $\sigma' = \sigma$. And with that knowledge, we can again build the required $\mathcal{E}$, now using the EC-WhileF rule:

  $$\mathcal{E} = \frac{\overset{\mathcal{E}'_0}{\langle b, \sigma\rangle \downarrow \textbf{false}}}{\langle \textbf{while } b \textbf{ do } c_0, \sigma\rangle \downarrow \sigma}\,.$$

■

Again, the arguments in both directions, while superficially similar, are by no means identical, and there's no way to reliably prove both directions in $\sim$ at once – especially since there's no a priori guarantee that c_0 necessarily terminates, let alone that c and c' do. In particular, it would be plain incorrect to paraphrase the theorem as "c and c' in the same initial store always execute to the same final store."

2.2 Inductive reasoning about IMP semantics

In proofs about programming-language semantics we will frequently be reasoning about sorts whose elements are tree-like structures. Most obviously, sorts representing *program phrases*, such as arithmetic expressions, have abstract syntax trees as elements; but also *derivations* of various judgments, such as expression evaluation, are evidently trees, with rule instances as nodes (and axioms making up the leaf nodes). In this section, we will first briefly sketch some general principles for reasoning about trees, and then consider how these principles can be used to show some useful facts about the semantics.

2.2.1 Inductive reasoning principles

Unfinished. See *FSoPL* Sections 3.1, 3.2, and 3.5 for now.

2.2.2 Totality of expression evaluation

We start by showing totality of the evaluation judgment for arithmetic expressions:

Theorem 2.3 *For every arithmetic expression a and store σ, there exist an integer n and a derivation $\mathcal{E}$ of $\langle a, \sigma\rangle \downarrow n$.*

Proof. By induction on the *syntax* (or *structure*) of a:

- Case $a = \overline{n_0}$, for some n_0. Then we can simply take $n = n_0$, and use rule EA-Num to construct:

$$\mathcal{E} = \frac{}{\langle \overline{n_0}, \sigma \rangle \downarrow n_0} .$$

  This $\mathcal{E}$ is evidently a derivation of $\langle a, \sigma \rangle \downarrow n$, as required.

- Case $a = X$. Then we take $n = \sigma(X)$, and use rule EA-Loc:

$$\mathcal{E} = \frac{}{\langle X, \sigma \rangle \downarrow \sigma(X)} .$$

- Case $a = a_0 + a_1$. By IH on a_0, there exist an n_0 and a derivation $\mathcal{E}_0$ of $\langle a_0, \sigma \rangle \downarrow n_0$; likewise, by IH on a_1, there exist n_1 and $\mathcal{E}_1$ of $\langle a_1, \sigma \rangle \downarrow n_1$. But then we can take $n = n_0 + n_1$, and use EA-Plus to construct:

$$\mathcal{E} = \frac{\begin{array}{cc} \mathcal{E}_0 & \mathcal{E}_1 \\ \langle a_0, \sigma \rangle \downarrow n_0 & \langle a_1, \sigma \rangle \downarrow n_1 \end{array}}{\langle a_0 + a_1, \sigma \rangle \downarrow n_0 + n_1} .$$

- The cases for $a = a_0 - a_1$ and $a = a_0 \times a_1$ are completely analogous to $a = a_0 + a_1$.

■

Note that if we had integer division in the language, with syntax $a ::= \cdots \mid a_0 \,\mathbf{div}\, a_1$, and evaluation rule

$$\text{EA-DivNZ} : \frac{\langle a_0, \sigma \rangle \downarrow n_0 \qquad \langle a_1, \sigma \rangle \downarrow n_1}{\langle a_0 \,\mathbf{div}\, a_1, \sigma \rangle \downarrow \lfloor n_0 / n_1 \rfloor} \, (n_1 \neq 0) ,$$

we would *not* be able to show the theorem: in the inductive case for $a = a_0 \,\mathbf{div}\, a_1$, the IH would still give us derivations of $\langle a_0, \sigma \rangle \downarrow n_0$ and $\langle a_1, \sigma \rangle \downarrow n_1$, but we would not always be able to put those together into the desired conclusion, because there is no applicable rule for the case where $n_1 = 0$.

Knowing that expression evaluation is total allows us to show additional semantic equivalences, such as $\overline{0} \times a \sim \overline{0}$. Here, the left-to-right direction of the biimplication in $\sim$ always holds, but for the right-to-left one, we need to establish that every a evaluates to *some* number, even if that number doesn't matter for the ultimate value of $a \times \overline{0}$.

Similarly, we can show:

Theorem 2.4 *For every boolean expression b and store σ, there exist a truth value t and a derivation $\mathcal{E}$ of $\langle b, \sigma \rangle \downarrow t$.*

Proof. By induction on the syntax of b. The cases are:

- Case $b = t_0$, for some t_0. Then we simply take $t = t_0$, and by the rule EB-Cst,

$$\mathcal{E} = \frac{}{\langle t_0, \sigma \rangle \downarrow t_0} .$$

- Case $b = (a_0 = a_1)$. By Theorem 2.3 on a_0 (note: *not* by IH), we get a derivation $\mathcal{E}_0$ of $\langle a_0, \sigma\rangle \downarrow n_0$ for some integer n_0. Likewise, by the same theorem on a_1, we get $\mathcal{E}_1$ of $\langle a_1, \sigma\rangle \downarrow n_1$ for some n_1. Now there are two possibilities:[1]
  - Subcase $n_0 = n_1$. Then we can take $n = n_0$ (which also makes $n = n_1$), $t = \mathbf{true}$, and use rule EB-EqT to construct:
$$\mathcal{E} = \frac{\begin{array}{cc}\mathcal{E}_0 & \mathcal{E}_1\\ \langle a_0, \sigma\rangle \downarrow n & \langle a_1, \sigma\rangle \downarrow n\end{array}}{\langle a_0 = a_1, \sigma\rangle \downarrow \mathbf{true}}\,.$$
  - Subcase $n_0 \neq n_1$. Then we can use the rule EB-EqF directly, because its side condition is satisfied, and construct:
$$\mathcal{E} = \frac{\begin{array}{cc}\mathcal{E}_0 & \mathcal{E}_1\\ \langle a_0, \sigma\rangle \downarrow n_0 & \langle a_1, \sigma\rangle \downarrow n_1\end{array}}{\langle a_0 = a_1, \sigma\rangle \downarrow \mathbf{false}}\,.$$
- Case $b = (a_0 \leq a_1)$. Analogous to the above, again exploiting that at least one (in fact, exactly one, though that's not important here) of the relations $n_0 \leq n_1$ or $n_0 > n_1$ will hold for any pair of integers n_0 and n_1.
- Case $b = \neg b_0$. By the IH on b_0, we get a derivation $\mathcal{E}_0$ of $\langle b_0, \sigma\rangle \downarrow t_0$ for some t_0. The grammar of truth values tells us that t_0 must be one of **true** or **false**. In the former subcase, we take $t = \mathbf{false}$ and use EB-NegT to construct:
$$\mathcal{E} = \frac{\begin{array}{c}\mathcal{E}_0\\ \langle b_0, \sigma\rangle \downarrow \mathbf{true}\end{array}}{\langle \neg b_0, \sigma\rangle \downarrow \mathbf{false}}\,;$$
  the latter subcase ($t_0 = \mathbf{false}$) is analogous, using the EB-NegF rule.
- Case $b = b_0 \wedge b_1$. By IH on b_0, we get an $\mathcal{E}_0$ of $\langle b_0, \sigma\rangle \downarrow t_0$ for some t_0. Again there are only two possibilities for t_0:
  - Subcase $t_0 = \mathbf{true}$. By IH on b_1 we obtain $\mathcal{E}_1$ of $\langle b_1, \sigma\rangle \downarrow t_1$ for some t_1. Now, taking $t = t_1$, we can use rule EB-AndT to construct:
$$\mathcal{E} = \frac{\begin{array}{cc}\mathcal{E}_0 & \mathcal{E}_1\\ \langle b_0, \sigma\rangle \downarrow \mathbf{true} & \langle b_1, \sigma\rangle \downarrow t_1\end{array}}{\langle b_0 \wedge b_1, \sigma\rangle \downarrow t_1}\,.$$
  - Subcase $t_0 = \mathbf{false}$. In this case, we do not even need to use the IH on b_1, but we can directly take $t = \mathbf{false}$ and use rule EB-AndF:
$$\mathcal{E} = \frac{\begin{array}{c}\mathcal{E}_0\\ \langle b_0, \sigma\rangle \downarrow \mathbf{false}\end{array}}{\langle b_0 \wedge b_1, \sigma\rangle \downarrow \mathbf{false}}\,.$$

∎

[1] Note that this assertion does *not* make the proof non-constructive: We are not using the general LEM here, but the much weaker domain-specific axiom, $\forall n, n' : \mathbf{Z}.\, n = n' \vee n \neq n'$. This says that the equality relation on integers is *decidable*. On the other hand, in a constructive logic, there is no axiom like $\forall \sigma, \sigma' : \Sigma.\, \sigma = \sigma' \vee \sigma \neq \sigma'$ because two infinite-domain functions – even if total and computable – cannot in general be tested for equality in finite time.

There is no analogous totality theorem for command execution: in general, it is *not* the case that, for any command c and store σ, there exists a derivation of $\langle c, \sigma\rangle \downarrow \sigma'$ for some σ'. For consider, e.g., $c = \textbf{while true do skip}$, and suppose that there exists at least one derivation of $\langle \textbf{while true do skip}, \sigma_0\rangle \downarrow \sigma_1$ for some σ_0 and σ_1. In general, there might be several different derivations of the judgment, so let us pick a *minimal* one (for example, with respect to the total number of rule applications, or the length of the longest branch in the derivation tree), and call it $\mathcal{E}_0$. Since c is a **while**-command, and there is only one possible rule each for evaluating **true** and executing **skip**, $\mathcal{E}_0$ must necessarily have the following shape:

$$\mathcal{E}_0 = \text{EC-WhileT}\dfrac{\text{EB-Cst}\dfrac{}{\langle \textbf{true}, \sigma_0\rangle \downarrow \textbf{true}} \qquad \text{EC-Skip}\dfrac{}{\langle \textbf{skip}, \sigma_0\rangle \downarrow \sigma_0} \qquad \overset{\mathcal{E}_0'}{\langle \textbf{while true do skip}, \sigma_0\rangle \downarrow \sigma_1}}{\langle \textbf{while true do skip}, \sigma_0\rangle \downarrow \sigma_1},$$

for some subderivation $\mathcal{E}_0'$. But $\mathcal{E}_0'$ is evidently strictly smaller than $\mathcal{E}_0$, while having exactly the same conclusion, which contradicts that $\mathcal{E}_0$ was minimal. Therefore, can be no such $\mathcal{E}_0$ in the first place, and thus $\langle \textbf{while true do skip}, \sigma_0\rangle \downarrow \sigma_1$ is not derivable.

We can, however, show that all of IMP's semantic judgments (evaluation of arithmetic and boolean expressions, and execution of commands) are *deterministic*, i.e., return *at most* one result.

2.2.3 Determinism of evaluation and command execution

We start by formulating determinism for arithmetic expressions. The proof is fairly similar to that of totality:

Theorem 2.5 *If $\langle a, \sigma\rangle \downarrow n$, and also $\langle a, \sigma\rangle \downarrow n'$, then $n = n'$.*

Proof. Let $\mathcal{E}$ be the derivation of $\langle a, \sigma\rangle \downarrow n$, and $\mathcal{E}'$ of $\langle a, \sigma\rangle \downarrow n'$. The proof proceeds by induction on the syntax of a. The cases are:

- Case $a = \overline{n_0}$. Since there is only one rule for evaluation of numerals, we must have

  $$\mathcal{E} = \frac{}{\langle \overline{n_0}, \sigma\rangle \downarrow n_0},$$

  and thus $n = n_0$. Analogously, since the above is also the only possible shape for $\mathcal{E}'$, we must have $n' = n_0$. But then, by symmetry and transitivity of mathematical equality, we get $n = n_0 = n'$ as required.

- Case $a = X$. Again, since EA-Loc is the only rule for evaluating locations, it must be the case that

  $$\mathcal{E} = \mathcal{E}' = \frac{}{\langle X, \sigma\rangle \downarrow \sigma(X)},$$

  and thus $n = \sigma(X) = n'$.

- Case $a = a_0 + a_1$. Since evaluation of an addition expression could only have ended in a use of rule EA-Plus, we must have, for some $\mathcal{E}_0$, $\mathcal{E}_1$, $\mathcal{E}_0'$, and $\mathcal{E}_1'$:

  $$\mathcal{E} = \frac{\overset{\mathcal{E}_0}{\langle a_0, \sigma\rangle \downarrow n_0} \qquad \overset{\mathcal{E}_1}{\langle a_1, \sigma\rangle \downarrow n_1}}{\langle a_0 + a_1, \sigma\rangle \downarrow n_0 + n_1} \quad \text{and} \quad \mathcal{E}' = \frac{\overset{\mathcal{E}_0'}{\langle a_0, \sigma\rangle \downarrow n_0'} \qquad \overset{\mathcal{E}_1'}{\langle a_1, \sigma\rangle \downarrow n_1'}}{\langle a_0 + a_1, \sigma\rangle \downarrow n_0' + n_1'},$$

with $n = n_0 + n_1$ and $n' = n'_0 + n'_1$. But by IH on a_0 with $\mathcal{E}_0$ and $\mathcal{E}'_0$, we get $n_0 = n'_0$; and by IH on a_1 with $\mathcal{E}_1$ and $\mathcal{E}'_1$, $n_1 = n'_1$, so by usual mathematical reasoning, also $n = n_0 + n_1 = n'_0 + n'_1 = n'$.

- The cases for subtraction and multiplication are analogous.

■

Like totality, deterministic evaluation of expressions is not a given. In particular, the semantics may be (deliberately or accidentally) *underspecified*, so that certain expressions may evaluate to unpredictable results. The language C is notorious for this, including, e.g., when accessing uninitialized local variables in a function, indexing arrays out of bounds, or even overflowing *signed* integer arithmetic (though in practice, these days, virtually all C implementations use a two's complement representation without overflow detection). As a simple example in an IMP context, we could extend the previously introduced integer-division operator with another rule, saying that an attempted division by zero is allowed to return an arbitrary result:

$$\text{EA-DivZ} : \frac{\langle a_0, \sigma\rangle \downarrow n_0 \qquad \langle a_1, \sigma\rangle \downarrow 0}{\langle a_0 \,\mathbf{div}\, a_1, \sigma\rangle \downarrow n}\,.$$

Note that the n in the conclusion does not appear anywhere else in the rule, meaning that it may be instantiated with any integer (not necessarily the same even in repeated uses of EA-DivZ) in a derivation. Such an underspecified semantics may make the compiler's job a bit easier, and sometimes allows it to generate slightly faster code, but often at a considerable cost for portability and reproducibility. Most newer imperative languages, such as Java, take great care to ensure that the expected evaluation results are completely specified (possibly as throwing an exception) in all situations.

We need determinism of expression evaluation to prove, e.g., the semantic equivalence $\overline{2} \times a \sim a + a$. Here the left-to-right direction holds even if expression evaluation is not necessarily deterministic (or total); but for the right-to-left one, we need to know that the two copies of a on the RHS can only evaluate to the same number n, in order to construct a derivation showing that the LHS evaluates to $2 \times n$.

Similarly, we can show:

Theorem 2.6 *If $\langle b, \sigma\rangle \downarrow t$, and also $\langle b, \sigma\rangle \downarrow t'$, then $t = t'$.*

The proof is left as an exercise.

Although command execution in IMP is also deterministic, we cannot actually show it by induction on the syntactic structure of the command. The reason is that, in the case pertaining to **while**-commands, the command in the last premise of rule EC-WhileT is not a *proper* subcommand of the original one. Of course, the failure to account for just this one case in a proof by induction on the syntax of c means that the almost-complete proof is invalid, even if the result itself holds. We can, however, prove it in a different way:

Theorem 2.7 *If $\langle c, \sigma_0\rangle \downarrow \sigma_1$ (by some derivation $\mathcal{E}$), and also $\langle c, \sigma_0\rangle \downarrow \sigma_1'$ (by $\mathcal{E}'$), then $\sigma_1 = \sigma_1'$.*

Proof. By induction on the derivation $\mathcal{E}$. The cases are:

- Case $\mathcal{E} = \dfrac{}{\langle \mathbf{skip}, \sigma_0\rangle \downarrow \sigma_0}$, so $c = \mathbf{skip}$ and $\sigma_1 = \sigma_0$.

  Since there is only one rule for **skip**-commands, we know that $\mathcal{E}'$ must also have shape
  $$\mathcal{E}' = \frac{}{\langle \mathbf{skip}, \sigma_0\rangle \downarrow \sigma_0}\,,$$
  and thus $\sigma_1' = \sigma_0$. And therefore we have $\sigma_1 = \sigma_0 = \sigma_1'$.

- Case $\mathcal{E} = \dfrac{\begin{array}{c}\mathcal{E}_0\\ \langle a, \sigma_0\rangle \downarrow n\end{array}}{\langle X := a, \sigma_0\rangle \downarrow \sigma_0[X \mapsto n]}$, so $c = (X := a)$ and $\sigma_1 = \sigma_0[X \mapsto n]$.

  Again, since there is only one rule for assignment commands, we must have
  $$\mathcal{E}' = \frac{\begin{array}{c}\mathcal{E}_0'\\ \langle a, \sigma_0\rangle \downarrow n'\end{array}}{\langle X := a, \sigma_0\rangle \downarrow \sigma_0[X \mapsto n']}\,,$$
  for some subderivation $\mathcal{E}_0'$, and $\sigma_1' = \sigma_0[X \mapsto n']$. But by Theorem 2.5 on $\mathcal{E}_0$ and $\mathcal{E}_0'$, we get $n = n'$, and hence also $\sigma_1 = \sigma_0[X \mapsto n] = \sigma_0[X \mapsto n'] = \sigma_1'$, as required.

- Case $\mathcal{E} = \dfrac{\begin{array}{cc}\mathcal{E}_0 & \mathcal{E}_1\\ \langle c_0, \sigma_0\rangle \downarrow \sigma_2 & \langle c_1, \sigma_2\rangle \downarrow \sigma_1\end{array}}{\langle c_0; c_1, \sigma_0\rangle \downarrow \sigma_1}$, so $c = (c_0; c_1)$.

  Since EC-Seq is the only rule for executing sequence-commands, the only possible shape of $\mathcal{E}'$ is:
  $$\mathcal{E}' = \frac{\begin{array}{cc}\mathcal{E}_0' & \mathcal{E}_1'\\ \langle c_0, \sigma_0\rangle \downarrow \sigma_2' & \langle c_1, \sigma_2'\rangle \downarrow \sigma_1'\end{array}}{\langle c_0; c_1, \sigma_0\rangle \downarrow \sigma_1'}\,.$$
  Now, by IH on $\mathcal{E}_0$ with $\mathcal{E}_0'$, we get that $\sigma_2 = \sigma_2'$. But that means that $\mathcal{E}_1'$ is also a derivation of $\langle c_1, \sigma_2\rangle \downarrow \sigma_1'$. And thus by IH on $\mathcal{E}_1$ with $\mathcal{E}_1'$ (which evidently both talk about the same command and starting store), we get $\sigma_1 = \sigma_1'$, as required.

- Case $\mathcal{E} = \dfrac{\begin{array}{cc}\mathcal{E}_0 & \mathcal{E}_1\\ \langle b, \sigma_0\rangle \downarrow \mathbf{true} & \langle c_0, \sigma_0\rangle \downarrow \sigma_1\end{array}}{\langle \mathbf{if}\ b\ \mathbf{then}\ c_0\ \mathbf{else}\ c_1, \sigma_0\rangle \downarrow \sigma_1}$, so $c = \mathbf{if}\ b\ \mathbf{then}\ c_0\ \mathbf{else}\ c_1$.

  Unlike in the previous cases, more than one rule applies for the execution of **if**-commands. Hence, there are a priori two possibilities for the shape of $\mathcal{E}'$:

  - Subcase $\mathcal{E}' = \dfrac{\begin{array}{cc}\mathcal{E}_0' & \mathcal{E}_1'\\ \langle b, \sigma_0\rangle \downarrow \mathbf{true} & \langle c_0, \sigma_0\rangle \downarrow \sigma_1'\end{array}}{\langle \mathbf{if}\ b\ \mathbf{then}\ c_0\ \mathbf{else}\ c_1, \sigma_0\rangle \downarrow \sigma_1'}$, i.e., $\mathcal{E}'$ also ended in EC-IfT.

    In this case, by IH on $\mathcal{E}_1$ with $\mathcal{E}_1'$, we get that $\sigma_1 = \sigma_1'$, as required. (Note that we did not use the subderivation $\mathcal{E}_0'$ for anything.)

– Subcase $\mathcal{E}' = \dfrac{\overset{\mathcal{E}'_0}{\langle b, \sigma_0\rangle \downarrow \mathbf{false}} \qquad \overset{\mathcal{E}'_1}{\langle c_1, \sigma_0\rangle \downarrow \sigma'_1}}{\langle \mathbf{if}\ b\ \mathbf{then}\ c_0\ \mathbf{else}\ c_1, \sigma_0\rangle \downarrow \sigma'_1}$, i.e., $\mathcal{E}'$ ended in EC-IfF.

  This subcase is actually impossible. For by determinism of boolean expressions (Theorem 2.6) on $\mathcal{E}_0$ and $\mathcal{E}'_0$, we get that $\mathbf{true} = \mathbf{false}$, which is a contradiction – from which anything follows, including in particular that $\sigma_1 = \sigma'_1$. (Which is just as well, since $\mathcal{E}_1$ and $\mathcal{E}'_1$ actually talk about different commands here, so we couldn't use the IH on them.)

- The case where $\mathcal{E}$ ends with EC-IfF is completely analogous to the previous one.

- Case $\mathcal{E} = \dfrac{\overset{\mathcal{E}_0}{\langle b, \sigma_0\rangle \downarrow \mathbf{false}}}{\langle \mathbf{while}\ b\ \mathbf{do}\ c_0, \sigma_0\rangle \downarrow \sigma_0}$, so $c = \mathbf{while}\ b\ \mathbf{do}\ c_0$, and $\sigma_1 = \sigma_0$.

  Like for **if**-commands, for this c, there are a priori two possibilities for the shape of $\mathcal{E}'$; but by determinism of boolean expressions again, the subcase where $\mathcal{E}'$ uses EC-WhileT is impossible, so we must also have

$$\mathcal{E}' = \frac{\overset{\mathcal{E}'_0}{\langle b, \sigma_0\rangle \downarrow \mathbf{false}}}{\langle \mathbf{while}\ b\ \mathbf{do}\ c_0, \sigma_0\rangle \downarrow \sigma_0}\,,$$

  so also $\sigma'_1 = \sigma_0$, and thus $\sigma_1 = \sigma_0 = \sigma'_1$ as required.

- Case $\mathcal{E} = \dfrac{\overset{\mathcal{E}_0}{\langle b, \sigma_0\rangle \downarrow \mathbf{true}} \qquad \overset{\mathcal{E}_1}{\langle c_0, \sigma_0\rangle \downarrow \sigma_2} \qquad \overset{\mathcal{E}_2}{\langle \mathbf{while}\ b\ \mathbf{do}\ c_0, \sigma_2\rangle \downarrow \sigma_1}}{\langle \mathbf{while}\ b\ \mathbf{do}\ c_0, \sigma_0\rangle \downarrow \sigma_1}$.

  Analogously to the previous case, the subcase where $\mathcal{E}'$ uses EC-WhileF is impossible, so we know that $\mathcal{E}'$ must have the shape:

$$\mathcal{E}' = \frac{\overset{\mathcal{E}'_0}{\langle b, \sigma_0\rangle \downarrow \mathbf{true}} \qquad \overset{\mathcal{E}'_1}{\langle c_0, \sigma_0\rangle \downarrow \sigma'_2} \qquad \overset{\mathcal{E}'_2}{\langle \mathbf{while}\ b\ \mathbf{do}\ c_0, \sigma'_2\rangle \downarrow \sigma'_1}}{\langle \mathbf{while}\ b\ \mathbf{do}\ c_0, \sigma_0\rangle \downarrow \sigma'_1}\,.$$

  Now, by IH on $\mathcal{E}_1$ with $\mathcal{E}'_1$, we get that $\sigma_2 = \sigma'_2$. Hence, $\mathcal{E}'_2$ is also a derivation of $\langle \mathbf{while}\ b\ \mathbf{do}\ c_0, \sigma_2\rangle \downarrow \sigma'_1$. And thus, by IH on $\mathcal{E}_2$ with $\mathcal{E}'_2$, we finally get that $\sigma_1 = \sigma'_1$.

■

2.3 Small-step semantics of IMP, and equivalence of the semantics

The big-step semantics we have seen so far is intuitive and well suited for many kinds of formal reasoning about the behavior of IMP programs. It is also fairly straightforward to adapt to extensions and variants of IMP (though some conceptually simple modifications may require wide-ranging changes to the core judgment forms). However, it has a couple of intrinsic shortcomings that make it problematic for certain applications, most notably:

- It does not distinguish between possible the *failures* of a program to yield a result. In particular, there is no formal difference between evaluation *getting stuck* because of a missing rule to cover a particular case (for example, an attempted division by zero if we only include the EA-DivNZ rule), and *divergence*, where a successful derivation would need to be infinitely large (e.g., when trying to execute **while true do skip**). In many situations, we can safely ignore the difference, or rework the semantics to allow it to return and propagate an explicit error result (much like a non-catchable exception). However, for some applications, such as showing type soundness, it is rather more convenient to express the semantics in such a way that certain error situations are represented as explicitly stuck derivations.

- It is poorly suited for specifying the semantics of *multi-threaded* languages, where we want to express that independent command executions may happen *concurrently*, with the threads communicating by exchanging messages and/or through a shared store. Here, we want to say that two or more computations may proceed in an arbitrarily interleaved fashion, which is awkward to express in terms of a judgment that only returns the final result of an evaluation or execution.

For such situations, we may instead express the semantics in a *small-step* style, conceptually specifying how a program execution proceeds incrementally from an initial configuration to a final one (if any). Crucially, however, we will be able to show that the two semantics still define the same language, only with the small-step one making a finer distinction between programs that *fail* to finish execution for whatever reason.

2.3.1 Small-step semantics

We formulate the small-step semantics of IMP slightly differently from *FSoPL* Section 2.6. The essential changes (apart from using $\to$ instead of $\to_1$) are:

- For expressions, we choose to make it explicit in the shape of the semantic judgment that evaluation of an arithmetic or boolean expression cannot change the store. This also demonstrates how a small-step semantics can maintain some manifestly invariant data, rather than making it all part of the configuration.

- For commands, we only operate with one shape of configurations, $\langle c, \sigma \rangle$. *FSoPL* distinguishes syntactically between finished and non-finished configurations, which makes it a bit awkward to state some results concisely.

- We define the reflexive-transitive closure of the step relations as explicit judgments, given by inference rules. This tends to make the proofs more uniform, since everything is now proved by induction on derivations.

The judgments and rules are shown in Figure 2.2. Note in particular how the strict separation between numerals and numbers allows us to maintain a distinction between "syntactic" and "semantic" addition operators in rules like SA-Plus.

2.3.2 Equivalence for arithmetic expressions

We aim to show that $\langle a, \sigma \rangle \downarrow n$ iff $\sigma \vdash a \to^* \overline{n}$. As in most non-trivial semantic equivalences, the two directions of the bi-implication involve rather different proofs.

Judgment $\boxed{\sigma \vdash a \to a'}$:

(no rules for $\overline{n}$: already fully evaluated) $\quad$ SA-Loc : $\dfrac{}{\sigma \vdash X \to \overline{\sigma(X)}}$

SA-Plus1 : $\dfrac{\sigma \vdash a_0 \to a_0'}{\sigma \vdash a_0 + a_1 \to a_0' + a_1}$ $\quad$ SA-Plus2 : $\dfrac{\sigma \vdash a_1 \to a_1'}{\sigma \vdash \overline{n_0} + a_1 \to \overline{n_0} + a_1'}$

SA-Plus : $\dfrac{}{\sigma \vdash \overline{n_0} + \overline{n_1} \to \overline{n_0 + n_1}}$

(analogous rule sets for $a_0 - a_1$ and $a_0 \times a_1$ omitted)

Judgment $\boxed{\sigma \vdash a \to^* a'}$:

SSA-Zero : $\dfrac{}{\sigma \vdash a \to^* a}$ $\quad$ SSA-More : $\dfrac{\sigma \vdash a \to a'' \qquad \sigma \vdash a'' \to^* a'}{\sigma \vdash a \to^* a'}$

Judgment $\boxed{\sigma \vdash b \to b'}$ (incomplete, left as an exercise):

SB-And1 : $\dfrac{\sigma \vdash b_0 \to b_0'}{\sigma \vdash b_0 \wedge b_1 \to b_0' \wedge b_1}$

SB-AndT : $\dfrac{}{\sigma \vdash \mathbf{true} \wedge b_1 \to b_1}$ $\quad$ SB-AndF : $\dfrac{}{\sigma \vdash \mathbf{false} \wedge b_1 \to \mathbf{false}}$

Judgment $\boxed{\sigma \vdash b \to^* b'}$: analogous to $\sigma \vdash a \to^* a'$

Judgment $\boxed{\langle c, \sigma\rangle \to \langle c', \sigma'\rangle}$:

(No rules for **skip**: already fully executed)

SC-Assign1 : $\dfrac{\sigma \vdash a \to a'}{\langle X := a, \sigma\rangle \to \langle X := a', \sigma\rangle}$ $\quad$ SC-Assign : $\dfrac{}{\langle X := \overline{n}, \sigma\rangle \to \langle \mathbf{skip}, \sigma[X \mapsto n]\rangle}$

SC-Seq1 : $\dfrac{\langle c_0, \sigma\rangle \to \langle c_0', \sigma'\rangle}{\langle c_0; c_1, \sigma\rangle \to \langle c_0'; c_1, \sigma'\rangle}$ $\quad$ SC-Seq : $\dfrac{}{\langle \mathbf{skip}; c_1, \sigma\rangle \to \langle c_1, \sigma\rangle}$

SC-If1 : $\dfrac{\sigma \vdash b \to b'}{\langle \mathbf{if}\ b\ \mathbf{then}\ c_0\ \mathbf{else}\ c_1, \sigma\rangle \to \langle \mathbf{if}\ b'\ \mathbf{then}\ c_0\ \mathbf{else}\ c_1, \sigma\rangle}$

SC-IfT : $\dfrac{}{\langle \mathbf{if\ true\ then}\ c_0\ \mathbf{else}\ c_1, \sigma\rangle \to \langle c_0, \sigma\rangle}$ $\quad$ SC-IfF : $\dfrac{}{\langle \mathbf{if\ false\ then}\ c_0\ \mathbf{else}\ c_1, \sigma\rangle \to \langle c_1, \sigma\rangle}$

SC-While : $\dfrac{}{\langle \mathbf{while}\ b\ \mathbf{do}\ c_0, \sigma\rangle \to \langle \mathbf{if}\ b\ \mathbf{then}\ (c_0; \mathbf{while}\ b\ \mathbf{do}\ c_0)\ \mathbf{else\ skip}, \sigma\rangle}$

Judgment $\boxed{\langle c, \sigma\rangle \to^* \langle c', \sigma'\rangle}$:

SSC-Zero : $\dfrac{}{\langle c, \sigma\rangle \to^* \langle c, \sigma\rangle}$ $\quad$ SSC-More : $\dfrac{\langle c, \sigma\rangle \to \langle c'', \sigma''\rangle \qquad \langle c'', \sigma''\rangle \to^* \langle c', \sigma'\rangle}{\langle c, \sigma\rangle \to^* \langle c', \sigma'\rangle}$

Figure 2.2: Small-step semantics of IMP

From big-step to small-step derivations

This is the conceptually simpler direction. Before we begin the proof proper, we first establish that, for any fixed σ, the relation $\sigma \vdash \cdot \to^* \cdot$ is indeed transitive:

Lemma 2.8 *If $\sigma \vdash a \to^* a'$ (by $\mathcal{SS}$) and $\sigma \vdash a' \to^* a''$ (by $\mathcal{SS}'$), then $\sigma \vdash a \to^* a''$ by some $\mathcal{SS}''$.*

Proof. Simple induction on the derivation $\mathcal{SS}$. Its possible shapes are:

- Case $\mathcal{SS} = \dfrac{}{\sigma \vdash a \to^* a}$. Then $a' = a$, so the second assumption $\mathcal{SS}'$ can be used directly as the required derivation $\mathcal{SS}''$ of $\sigma \vdash a \to^* a''$.
- Case $\mathcal{SS} = \dfrac{\overset{\mathcal{S}_0}{\sigma \vdash a \to a'''} \qquad \overset{\mathcal{SS}_0}{\sigma \vdash a''' \to^* a'}}{\sigma \vdash a \to^* a'}$. Then, by IH on $\mathcal{SS}_0$ with $\mathcal{SS}'$, we get a derivation $\mathcal{SS}'_0$ of $\sigma \vdash a''' \to^* a''$. And taking that together with the first step $\mathcal{S}_0$, we can construct the required derivation of $\sigma \vdash a \to^* a''$:

$$\mathcal{SS}'' = \frac{\overset{\mathcal{S}_0}{\sigma \vdash a \to a'''} \qquad \overset{\mathcal{SS}'_0}{\sigma \vdash a''' \to^* a''}}{\sigma \vdash a \to^* a''}\,.$$

■

Further, we can show that the *context rules*, such as SA-Plus1, also "extend" from single steps to multi-step sequences:

Lemma 2.9 (for SA-Plus1**)** *If $\sigma \vdash a_0 \to^* a'_0$ (by $\mathcal{SS}$), then for any a_1, also $\sigma \vdash a_0 + a_1 \to^* a'_0 + a_1$ (by some $\mathcal{SS}'$).*

Proof. Simple induction on the derivation $\mathcal{SS}$. The case where $\mathcal{SS}$ uses SSA-Zero, i.e., $a'_0 = a_0$, is immediate, since $\mathcal{SS}'$ can then also use SSA-Zero for the expression $a_0 + a_1$. And for a non-empty $\mathcal{SS}$,

$$\mathcal{SS} = \frac{\overset{\mathcal{S}_0}{\sigma \vdash a_0 \to a''_0} \qquad \overset{\mathcal{SS}_0}{\sigma \vdash a''_0 \to^* a'_0}}{\sigma \vdash a_0 \to^* a'_0}\,,$$

the IH on $\mathcal{SS}_0$ gives us a derivation $\mathcal{SS}'_0$ of $\sigma \vdash a''_0 + a_1 \to^* a'_0 + a_1$, so by also using SA-Plus1 on the first step of $\mathcal{SS}$, we can construct the required:

$$\mathcal{SS}' = \text{SSA-More}\,\frac{\text{SA-Plus1}\,\dfrac{\overset{\mathcal{S}_0}{\sigma \vdash a_0 \to a''_0}}{\sigma \vdash a_0 + a_1 \to a''_0 + a_1} \qquad \overset{\mathcal{SS}'_0}{\sigma \vdash a''_0 + a_1 \to^* a'_0 + a_1}}{\sigma \vdash a_0 + a_1 \to^* a'_0 + a_1}\,.$$

■

Naturally, completely analogous lemmas hold for the other context rules. In the following, rather than explicitly stating such lemmas, we will often just refer to them implicitly with phrases like "by repeated application of SA-Plus2 to the steps of $\mathcal{SS}$"; but remember that, formally, each such argument involves proving (if we hadn't already) and applying a small result about the relevant context rule.

We can now state and prove the main theorem:

Theorem 2.10 *If $\langle a, \sigma\rangle \downarrow n$, then $\sigma \vdash a \to^* \overline{n}$.*

Proof. By induction on the derivation $\mathcal{E}$ of $\langle a, \sigma\rangle \downarrow n$, we construct the required derivation $\mathcal{SS}$ of $\sigma \vdash a \to^* \overline{n}$. We encounter the following cases for the shape of $\mathcal{E}$:

- Case $\mathcal{E} = \dfrac{}{\langle \overline{n}, \sigma\rangle \downarrow n}$, so $a = \overline{n}$.

  We get the required $\mathcal{SS} = \dfrac{}{\sigma \vdash \overline{n} \to^* \overline{n}}$ directly by SSA-Zero.

- Case $\mathcal{E} = \dfrac{}{\langle X, \sigma\rangle \downarrow \sigma(X)}$, so $a = X$.

  Here we need three rule applications to construct $\mathcal{SS}$:

$$\mathcal{SS} = \text{SSA-More}\ \dfrac{\text{SA-Loc}\ \dfrac{}{\sigma \vdash X \to \overline{\sigma(X)}} \qquad \text{SSA-Zero}\ \dfrac{}{\sigma \vdash \overline{\sigma(X)} \to^* \overline{\sigma(X)}}}{\sigma \vdash X \to^* \overline{\sigma(X)}}.$$

  In the remainder of these notes, we will just use without explicit comment that whenever we have $\sigma \vdash a \to a'$, then also $\sigma \vdash a \to^* a'$, by an application of SSA-Zero and SSA-More, as demonstrated above.

- Case $\mathcal{E} = \dfrac{\begin{array}{cc}\mathcal{E}_0 & \mathcal{E}_1\\ \langle a_0, \sigma\rangle \downarrow n_0 & \langle a_1, \sigma\rangle \downarrow n_1\end{array}}{\langle a_0 + a_1, \sigma\rangle \downarrow n_0 + n_1}$, so $a = a_0 + a_1$.

  By IH on $\mathcal{E}_0$, we obtain a small-step-sequence derivation $\mathcal{SS}_0$ of $\sigma \vdash a_0 \to^* \overline{n_0}$; and step-wise application of SA-Plus1 (i.e., Lemma 2.9) on $\mathcal{SS}_0$ gives us a $\mathcal{SS}'_0$ showing that $\sigma \vdash a_0 + a_1 \to^* \overline{n_0} + a_1$.

  Analogously, by the induction hypothesis on $\mathcal{E}_1$ we get a derivation $\mathcal{SS}_1$ of $\sigma \vdash a_1 \to^* \overline{n_1}$; and using SA-Plus2 on each step of $\mathcal{SS}_1$, we get a derivation $\mathcal{SS}'_1$ of $\sigma \vdash \overline{n_0} + a_1 \to^* \overline{n_0} + \overline{n_1}$.

  Finally, using SA-Plus, we get a one-step sequence $\mathcal{SS}_2$ showing $\sigma \vdash \overline{n_0} + \overline{n_1} \to^* \overline{n_0 + n_1}$. Concatenating $\mathcal{SS}'_0$, $\mathcal{SS}'_1$, and $\mathcal{SS}_2$ using Lemma 2.8 (twice), we finally obtain the required derivation $\mathcal{SS}$ of $\sigma \vdash a_0 + a_1 \to^* \overline{n_0 + n_1}$.

- The cases for subtraction and multiplication are analogous.

■

From small-step to big-step derivations

The other direction of the equivalence is more subtle. The key to getting a clean proof is formulating a suitable lemma, expressing a general relationship between small-step and big-step derivations, saying that we can "prepend" a single small step to a big-step evaluation:

Lemma 2.11 *Given a derivation $\mathcal{S}$ of $\sigma \vdash a \to a'$, and a derivation $\mathcal{E}'$ of $\langle a', \sigma\rangle \downarrow n$, then there also exists a derivation $\mathcal{E}$ of $\langle a, \sigma\rangle \downarrow n$.*

Proof. By induction on the derivation $\mathcal{S}$:

- Case $\mathcal{S} = \dfrac{}{\sigma \vdash X \to \overline{\sigma(X)}}$, so $a = X$ and $a' = \overline{\sigma(X)}$.

  Since EA-Num is the only possible big-step rule for evaluation of the numeral a', we must have

  $$\mathcal{E}' = \frac{}{\langle \overline{\sigma(X)}, \sigma\rangle \downarrow \sigma(X)}\,,$$

  so $n = \sigma(X)$. And thus, we can construct the required $\mathcal{E}$ using EA-Loc:

  $$\mathcal{E} = \frac{}{\langle X, \sigma\rangle \downarrow \sigma(X)}\,.$$

- Case $\mathcal{S} = \dfrac{\begin{array}{c}\mathcal{S}_0\\ \sigma \vdash a_0 \to a_0'\end{array}}{\sigma \vdash a_0 + a_1 \to a_0' + a_1}$, so $a = a_0 + a_1$ and $a' = a_0' + a_1$.

  Since only EA-Plus could have been used in the big-step evaluation of the addition expression a', the given big-step derivation $\mathcal{E}'$ must have the shape:

  $$\mathcal{E}' = \frac{\begin{array}{cc}\mathcal{E}_0' & \mathcal{E}_1\\ \langle a_0', \sigma\rangle \downarrow n_0 & \langle a_1, \sigma\rangle \downarrow n_1\end{array}}{\langle a_0' + a_1, \sigma\rangle \downarrow n_0 + n_1}\,,$$

  with $n = n_0 + n_1$. But then, by IH on $\mathcal{S}_0$ with $\mathcal{E}_0'$, we get a derivation $\mathcal{E}_0$ of $\langle a_0, \sigma\rangle \downarrow n_0$. And thus, we can construct the required big-step derivation $\mathcal{E}$ that starts already from $a_0 + a_1$:

  $$\mathcal{E} = \frac{\begin{array}{cc}\mathcal{E}_0 & \mathcal{E}_1\\ \langle a_0, \sigma\rangle \downarrow n_0 & \langle a_1, \sigma\rangle \downarrow n_1\end{array}}{\langle a_0 + a_1, \sigma\rangle \downarrow n_0 + n_1}\,.$$

- Case $\mathcal{S} = \dfrac{\begin{array}{c}\mathcal{S}_1\\ \sigma \vdash a_1 \to a_1'\end{array}}{\sigma \vdash \overline{n_0} + a_1 \to \overline{n_0} + a_1'}$, so $a = \overline{n_0} + a_1$ and $a' = \overline{n_0} + a_1'$.

  Like above, by considering the shape of a', the only possible shape for $\mathcal{E}'$ must be:

  $$\mathcal{E}' = \frac{\begin{array}{cc}\overline{\langle \overline{n_0}, \sigma\rangle \downarrow n_0} & \mathcal{E}_1'\\ & \langle a_1', \sigma\rangle \downarrow n_1\end{array}}{\langle \overline{n_0} + a_1', \sigma\rangle \downarrow n_0 + n_1}\,.$$

  Again, we must have $n = n_0 + n_1$; and by IH on $\mathcal{S}_1$ with $\mathcal{E}_1'$, we get a derivation $\mathcal{E}_1$ of $\langle a_1, \sigma\rangle \downarrow n_1$, so we can construct:

  $$\mathcal{E} = \frac{\begin{array}{cc}\overline{\langle \overline{n_0}, \sigma\rangle \downarrow n_0} & \mathcal{E}_1\\ & \langle a_1, \sigma\rangle \downarrow n_1\end{array}}{\langle \overline{n_0} + a_1, \sigma\rangle \downarrow n_0 + n_1}\,.$$

- Case $\mathcal{S} = \dfrac{}{\sigma \vdash \overline{n_0} + \overline{n_1} \to \overline{n_0 + n_1}}$, so $a = \overline{n_0} + \overline{n_1}$ and $a' = \overline{n_0 + n_1}$.

  In this case, a' is just a numeral, so $\mathcal{E}'$ must have used EA-Num, and we have $n = n_0 + n_1$. But then, we can use the big-step rules for constants and addition to construct:

$$\mathcal{E} = \frac{\overline{\langle \overline{n_0}, \sigma \rangle \downarrow n_0} \qquad \overline{\langle \overline{n_1}, \sigma \rangle \downarrow n_1}}{\langle \overline{n_0} + \overline{n_1}, \sigma \rangle \downarrow n_0 + n_1}.$$

■

Having done almost all the actual work in the above Lemma, we can now easily prove the converse of Theorem 2.10:

Theorem 2.12 *If $\sigma \vdash a \to^* \overline{n}$, then $\langle a, \sigma \rangle \downarrow n$.*

Proof. By induction on the derivation $\mathcal{SS}$ of the assumption. The two cases are:

- Case $\mathcal{SS} = \dfrac{}{\sigma \vdash \overline{n} \to^* \overline{n}}$, so $a = \overline{n}$.

  Here, the result follows directly by EA-Num, $\overline{\langle \overline{n}, \sigma \rangle \downarrow n}$.

- Case $\mathcal{SS} = \dfrac{\begin{array}{cc}\mathcal{S}_0 & \mathcal{SS}_0 \\ \sigma \vdash a \to a' & \sigma \vdash a' \to^* \overline{n}\end{array}}{\sigma \vdash a \to^* \overline{n}}$.

  Then we can use the IH on $\mathcal{SS}_0$ to obtain a big-step derivation of $\langle a', \sigma \rangle \downarrow n$. And "prepending" the step $\mathcal{S}_0$ to this derivation, using Lemma 2.11, gives us the required big-step derivation of $\langle a, \sigma \rangle \downarrow n$. ■

Theorems 2.10 and 2.12 together say that the big-step and small-step semantics of arithmetic expressions are equivalent. Note, however, that we will still need Lemma 2.11 itself, not only Theorem 2.12, to prove equivalence for commands.

2.3.3 Equivalence for boolean expressions

Once the small-step semantics of boolean expressions is correctly completed, we expect to show:

Lemma 2.13 *If $\sigma \vdash b \to^* b'$ and $\sigma \vdash b' \to^* b''$, then $\sigma \vdash b \to^* b''$.*

Theorem 2.14 *If $\langle b, \sigma \rangle \downarrow t$, then $\sigma \vdash b \to^* t$.*

Lemma 2.15 *If $\sigma \vdash b \to b'$ and $\langle b', \sigma \rangle \downarrow t$, then $\langle b, \sigma \rangle \downarrow t$.*

Theorem 2.16 *If $\sigma \vdash b \to^* t$, then $\langle b, \sigma \rangle \downarrow t$.*

The details are left as exercises.

2.3.4 Equivalence for commands

The proofs of equivalence for commands are similar in structure to the ones for expressions, but involve a greater variety of arguments, reflecting the greater complexities of the two semantics.

From big-step to small-step

Again, we will first need a concatenation lemma:

Lemma 2.17 *If* $\langle c, \sigma\rangle \to^* \langle c', \sigma'\rangle$ *and* $\langle c', \sigma'\rangle \to^* \langle c'', \sigma''\rangle$, *then* $\langle c, \sigma\rangle \to^* \langle c'', \sigma''\rangle$.

Proof. Analogous to Lemma 2.8. ∎

We also have variants of Lemma 2.9, extending all the context rules for command execution (SC-Assign1, etc.) to multistep derivations. With those, we can show the theorem:

Theorem 2.18 *If* $\langle c, \sigma\rangle \downarrow \sigma'$, *then* $\langle c, \sigma\rangle \to^* \langle \mathbf{skip}, \sigma'\rangle$.

Proof. By induction on the big-step derivation $\mathcal{E}$ of the assumption. The following cases arise:

- Case $\mathcal{E} = \dfrac{}{\langle \mathbf{skip}, \sigma\rangle \downarrow \sigma}$. Here $c = \mathbf{skip}$ and $\sigma' = \sigma$, and so we can construct the zero-step sequence $\dfrac{}{\langle \mathbf{skip}, \sigma\rangle \to^* \langle \mathbf{skip}, \sigma\rangle}$.

- Case $\mathcal{E} = \dfrac{\begin{array}{c}\mathcal{E}_0\\ \langle a, \sigma\rangle \downarrow n\end{array}}{\langle X := a, \sigma\rangle \downarrow \sigma[X \mapsto n]}$. By Theorem 2.10 on $\mathcal{E}_0$, we get a derivation $\mathcal{SS}_0$ of $\sigma \vdash a \to^* \overline{n}$, which we can convert to a derivation $\mathcal{SS}'_0$ of $\langle X := a, \sigma\rangle \to^* \langle X := \overline{n}, \sigma\rangle$ by using SC-Assign1 on each step. Moreover, SC-Assign allows us to construct a one-step sequence $\mathcal{SS}_1$ showing $\langle X := \overline{n}, \sigma\rangle \to^* \langle \mathbf{skip}, \sigma[X \mapsto n]\rangle$; and concatenating $\mathcal{SS}'_0$ and $\mathcal{SS}_1$ using Lemma 2.17, we obtain the required derivation of $\langle X := a, \sigma\rangle \to^* \langle \mathbf{skip}, \sigma[X \mapsto n]\rangle$.

- (The case for sequencing is left as an exercise.)

- Case $\mathcal{E} = \dfrac{\begin{array}{cc}\mathcal{E}_0 & \mathcal{E}_1\\ \langle b, \sigma\rangle \downarrow \mathbf{true} & \langle c_0, \sigma\rangle \downarrow \sigma'\end{array}}{\langle \mathbf{if}\ b\ \mathbf{then}\ c_0\ \mathbf{else}\ c_1, \sigma\rangle \downarrow \sigma'}$.
  By Theorem 2.14 on $\mathcal{E}_0$ we get a reduction sequence $\sigma \vdash b \to^* \mathbf{true}$. Using SC-If1 on each step of that, we obtain a corresponding sequence

  $$\langle \mathbf{if}\ b\ \mathbf{then}\ c_0\ \mathbf{else}\ c_1, \sigma\rangle \to^* \langle \mathbf{if}\ \mathbf{true}\ \mathbf{then}\ c_0\ \mathbf{else}\ c_1, \sigma\rangle\,,$$

  which can be concatenated with the one-step sequence $\langle \mathbf{if}\ \mathbf{true}\ \mathbf{then}\ c_0\ \mathbf{else}\ c_1, \sigma\rangle \to^* \langle c_0, \sigma\rangle$ that we get from the SC-IfT reduction rule, and then the sequence $\langle c_0, \sigma\rangle \to^* \langle \mathbf{skip}, \sigma'\rangle$ from the IH on $\mathcal{E}_1$, to show the required $\langle \mathbf{if}\ b\ \mathbf{then}\ c_0\ \mathbf{else}\ c_1, \sigma\rangle \to^* \langle \mathbf{skip}, \sigma'\rangle$.

- The case where $\mathcal{E}$ uses EC-IfF is completely analogous.
- Case $\mathcal{E} = \dfrac{\overset{\mathcal{E}_0}{\langle b, \sigma\rangle \downarrow \mathbf{false}}}{\langle \mathbf{while}\ b\ \mathbf{do}\ c_0, \sigma\rangle \downarrow \sigma}$, so $\sigma' = \sigma$.

  Here $c = \mathbf{while}\ b\ \mathbf{do}\ c_0$, so by SC-While, we get a one-step sequence:

  $$\langle \mathbf{while}\ b\ \mathbf{do}\ c_0, \sigma\rangle \to^* \langle \mathbf{if}\ b\ \mathbf{then}\ (c_0; c)\ \mathbf{else}\ \mathbf{skip}, \sigma\rangle\,.$$

  By Theorem 2.14 on $\mathcal{E}_0$ we get a sequence $\sigma \vdash b \to^* \mathbf{false}$, which by multiple uses of SC-If1 induces a sequence

  $$\langle \mathbf{if}\ b\ \mathbf{then}\ (c_0; c)\ \mathbf{else}\ \mathbf{skip}, \sigma\rangle \to^* \langle \mathbf{if}\ \mathbf{false}\ \mathbf{then}\ (c_0; c)\ \mathbf{else}\ \mathbf{skip}, \sigma\rangle\,.$$

  Finally, by the SC-IfF rule, we have in one step:

  $$\langle \mathbf{if}\ \mathbf{false}\ \mathbf{then}\ (c_0; c)\ \mathbf{else}\ \mathbf{skip}, \sigma\rangle \to^* \langle \mathbf{skip}, \sigma\rangle\,.$$

  Concatenating those three sequences together, we obtain the desired sequence $\langle \mathbf{while}\ b\ \mathbf{do}\ c_0, \sigma\rangle \to^* \langle \mathbf{skip}, \sigma\rangle$.
- Case $\mathcal{E} = \dfrac{\overset{\mathcal{E}_0}{\langle b, \sigma\rangle \downarrow \mathbf{true}} \qquad \overset{\mathcal{E}_1}{\langle c_0, \sigma\rangle \downarrow \sigma''} \qquad \overset{\mathcal{E}_2}{\langle \mathbf{while}\ b\ \mathbf{do}\ c_0, \sigma''\rangle \downarrow \sigma'}}{\langle \mathbf{while}\ b\ \mathbf{do}\ c_0, \sigma\rangle \downarrow \sigma'}$.

  Here we again have $c = \mathbf{while}\ b\ \mathbf{do}\ c_0$. Like in the EC-WhileF case, we first use the SC-While rule to get a one-step sequence:

  $$\langle \mathbf{while}\ b\ \mathbf{do}\ c_0, \sigma\rangle \to^* \langle \mathbf{if}\ b\ \mathbf{then}\ (c_0; c)\ \mathbf{else}\ \mathbf{skip}, \sigma\rangle\,.$$

  Next, by Theorem 2.14 on $\mathcal{E}_0$ we get $\sigma \vdash b \to^* \mathbf{true}$, from which the iterated SC-If1 gives us:

  $$\langle \mathbf{if}\ b\ \mathbf{then}\ (c_0; c)\ \mathbf{else}\ \mathbf{skip}, \sigma\rangle \to^* \langle \mathbf{if}\ \mathbf{true}\ \mathbf{then}\ (c_0; c)\ \mathbf{else}\ \mathbf{skip}, \sigma\rangle\,.$$

  Then, by the SC-IfT rule:

  $$\langle \mathbf{if}\ \mathbf{true}\ \mathbf{then}\ (c_0; c)\ \mathbf{else}\ \mathbf{skip}, \sigma\rangle \to^* \langle c_0; c, \sigma\rangle\,.$$

  Now, by IH on $\mathcal{E}_1$ we get $\langle c_0, \sigma\rangle \to^* \langle \mathbf{skip}, \sigma''\rangle$, from which we get, by the iterated SC-Seq1 (with $c_1 = c$):

  $$\langle c_0; c, \sigma\rangle \to^* \langle \mathbf{skip}; c, \sigma''\rangle\,.$$

  By SC-Seq we have, in one step:

  $$\langle \mathbf{skip}; c, \sigma''\rangle \to^* \langle c, \sigma''\rangle\,.$$

  Finally, by IH on $\mathcal{E}_2$, we get:

  $$\langle c, \sigma''\rangle \to^* \langle \mathbf{skip}, \sigma'\rangle\,.$$

  And concatenating the six displayed sequences together, we obtain the desired

  $$\langle \mathbf{while}\ b\ \mathbf{do}\ c_0, \sigma\rangle \to^* \langle \mathbf{skip}, \sigma'\rangle\,.$$

■

From small-step to big-step derivations

Like for expressions, the essence of the proof lies in the following lemma:

Lemma 2.19 *Given a derivation $\mathcal{S}$ of $\langle c, \sigma\rangle \to \langle c', \sigma'\rangle$, and a derivation $\mathcal{E}'$ of $\langle c', \sigma'\rangle \downarrow \sigma''$, then there also exists a derivation $\mathcal{E}$ of $\langle c, \sigma\rangle \downarrow \sigma''$.*

Proof. By induction on the derivation $\mathcal{S}$:

- Case $\mathcal{S} = \dfrac{\begin{array}{c}\mathcal{S}_0\\ \sigma \vdash a \to a'\end{array}}{\langle X := a, \sigma\rangle \to \langle X := a', \sigma\rangle}$.

  We have $c = (X := a)$, $c' = (X := a')$, and $\sigma' = \sigma$. Since there is only one big-step rule for assignment commands, $\mathcal{E}'$ must have the shape:

  $$\mathcal{E}' = \frac{\begin{array}{c}\mathcal{E}'_0\\ \langle a', \sigma\rangle \downarrow n\end{array}}{\langle X := a', \sigma\rangle \downarrow \sigma[X \mapsto n]}\,,$$

  and $\sigma'' = \sigma[X \mapsto n]$. By Lemma 2.11 on $\mathcal{S}_0$ and $\mathcal{E}'_0$, we obtain a big-step derivation $\mathcal{E}_0$ of $\langle a, \sigma\rangle \downarrow n$, which we can use to construct the required execution derivation:

  $$\mathcal{E} = \frac{\begin{array}{c}\mathcal{E}_0\\ \langle a, \sigma\rangle \downarrow n\end{array}}{\langle X := a, \sigma\rangle \downarrow \sigma[X \mapsto n]}\,.$$

- Case $\mathcal{S} = \dfrac{}{\langle X := \overline{n}, \sigma\rangle \to \langle \mathbf{skip}, \sigma[X \mapsto n]\rangle}$.

  Here $c = (X := \overline{n})$, $c' = \mathbf{skip}$ and $\sigma' = \sigma[X \mapsto n]$. Since $\mathcal{E}'$ must use EC-Skip, we also have $\sigma'' = \sigma'$. We can thus directly use EA-Num and EA-Assign to construct:

  $$\mathcal{E} = \frac{\overline{\langle \overline{n}, \sigma\rangle \downarrow n}}{\langle X := \overline{n}, \sigma\rangle \downarrow \sigma[X \mapsto n]}\,.$$

- (The cases for SC-Seq1 and SC-Seq are left as an exercise.)

- Case $\mathcal{S} = \dfrac{\begin{array}{c}\mathcal{S}_0\\ \sigma \vdash b \to b'\end{array}}{\langle \mathbf{if}\ b\ \mathbf{then}\ c_0\ \mathbf{else}\ c_1, \sigma\rangle \to \langle \mathbf{if}\ b'\ \mathbf{then}\ c_0\ \mathbf{else}\ c_1, \sigma\rangle}$.

  Again, we have $\sigma' = \sigma$. But $c' = \mathbf{if}\ b'\ \mathbf{then}\ c_0\ \mathbf{else}\ c_1$, so there are now two possibilities for how $\mathcal{E}'$ can look, depending on which of the two big-step rules for conditionals was used:

  - Subcase $\mathcal{E}' = \dfrac{\begin{array}{c}\mathcal{E}'_0\\ \langle b', \sigma\rangle \downarrow \mathbf{true}\end{array} \qquad \begin{array}{c}\mathcal{E}_1\\ \langle c_0, \sigma\rangle \downarrow \sigma''\end{array}}{\langle \mathbf{if}\ b'\ \mathbf{then}\ c_0\ \mathbf{else}\ c_1, \sigma\rangle \downarrow \sigma''}$

In this case, by Lemma 2.15 on the derivations $\mathcal{S}_0$ and $\mathcal{E}'_0$, we obtain a derivation $\mathcal{E}_0$ of $\langle b, \sigma\rangle \downarrow \mathbf{true}$, and we can then put together the required derivation $\mathcal{E}$ as follows:

$$\mathcal{E} = \frac{\overset{\mathcal{E}_0}{\langle b, \sigma\rangle \downarrow \mathbf{true}} \qquad \overset{\mathcal{E}_1}{\langle c_0, \sigma\rangle \downarrow \sigma''}}{\langle \mathbf{if}\ b\ \mathbf{then}\ c_0\ \mathbf{else}\ c_1, \sigma\rangle \downarrow \sigma''}\,.$$

– The subcase where $\mathcal{E}'$ ends in EC-IfF is analogous.

- Case $\mathcal{S} = \dfrac{}{\langle \mathbf{if\ true\ then}\ c_0\ \mathbf{else}\ c_1, \sigma\rangle \to \langle c_0, \sigma\rangle}$.

  Here $\sigma' = \sigma$, so we can use the supplied big-step derivation $\mathcal{E}'$ directly to construct the required $\mathcal{E}$, using EB-Cst and EC-IfT:

$$\mathcal{E} = \frac{\overline{\langle \mathbf{true}, \sigma\rangle \downarrow \mathbf{true}} \qquad \overset{\mathcal{E}'}{\langle c_0, \sigma\rangle \downarrow \sigma''}}{\langle \mathbf{if\ true\ then}\ c_0\ \mathbf{else}\ c_1, \sigma\rangle \downarrow \sigma''}\,.$$

- The case where $\mathcal{S}$ uses SC-IfF is analogous.

- (The case where $\mathcal{S}$ uses SC-While is left as an exercise.)

∎

Again, we can now show the converse of Theorem 2.18:

Theorem 2.20 *If* $\langle c, \sigma\rangle \to^* \langle \mathbf{skip}, \sigma'\rangle$, *then* $\langle c, \sigma\rangle \downarrow \sigma'$.

Proof. Analogous to the proof of Theorem 2.12: If the derivation $\mathcal{SS}$ of the assumption uses the rule SSC-Zero, we must already have $c = \mathbf{skip}$ and $\sigma' = \sigma$, and so we can use EC-Skip to obtain $\langle c, \sigma\rangle \downarrow \sigma'$. Otherwise, $\mathcal{SS}$ must end in rule SSC-More, so we have subderivations $\mathcal{S}_0$ of $\langle c, \sigma\rangle \to \langle c'', \sigma''\rangle$ and $\mathcal{SS}_0$ of $\langle c'', \sigma''\rangle \to^* \langle \mathbf{skip}, \sigma'\rangle$. Using the IH on $\mathcal{SS}_0$, we get an $\mathcal{E}_0$ of $\langle c'', \sigma''\rangle \downarrow \sigma'$, and then Lemma 2.19 on $\mathcal{S}_0$ and $\mathcal{E}_0$ gives us the required $\langle c, \sigma\rangle \downarrow \sigma'$. ∎

2.4 Exercises

1. Prove Theorem 2.6 by (a) induction on syntax, or (b) induction on derivations.

2. Supply the missing cases for Theorem 2.18 and Lemma 2.19 having to do with sequencing commands $(c_0; c_1)$.

3. Supply the missing case for **while**-loops in Lemma 2.19. *Hint:* use Theorem 2.2.

4. Complete the small-step semantics in Figure 2.2 by giving all the missing rules relating to boolean expressions. (The ones for conjunction are complete, and should not be extended or modified.)

5. Prove Theorem 2.14 and Lemma 2.15 for the cases relating to conjunctions (EB-AndT and EB-AndF vs. SB-And1, SB-AndT, and SB-AndF).