Programming Languages and Computation

Week 8: Big Steps, Blocks and Procedures

We first practice working with small-step and big-step semantics, and consider the notion of derivation trees more formally. Optional exercises provide depth on all topics covered in this part of the unit (including induction, derivations, and the choices made in defining semantics) beyond what we will expect in assessment, but that you may find useful as foundations for pathways focused on programming languages and their theory.

Small-Step and Big-Step Derivations

Recall that the inference rules we use to inductively define our semantics relations ($\leftarrow$ and $\Downarrow$) can be used to construct *derivations* for particular facts. We constructed a few small-step derivations for particular transitions in the lecture, using the small-step semantics rules.

We can similarly construct derivations for the big-step semantics relation $\Downarrow$. For example, the following is a derivation for the fact $\langle \mathtt{v} \leftarrow 2;\ \mathtt{x} \leftarrow \mathtt{v}\ +\ 3, []\rangle \Downarrow [\mathtt{v} \mapsto 2; \mathtt{x} \mapsto 5]$.

$$\dfrac{\dfrac{}{\langle \mathtt{v} \leftarrow 2, []\rangle \Downarrow [\mathtt{v} \mapsto 2]} \qquad \dfrac{}{\langle \mathtt{x} \leftarrow \mathtt{v}\ +\ 3, [\mathtt{v} \mapsto 2]\rangle \Downarrow [\mathtt{v} \mapsto 2; \mathtt{x} \mapsto 5]}}{\langle \mathtt{v} \leftarrow 2;\ \mathtt{x} \leftarrow \mathtt{v}\ +\ 3, []\rangle \Downarrow [\mathtt{v} \mapsto 2; \mathtt{x} \mapsto 5]}$$

* 1. Consider the While programs from Week 7 Question 4. Write big-step derivations for their semantics (when they exist) in the same initial state $\sigma = [\mathtt{n} \mapsto 3; \mathtt{a} \mapsto 5]$.

Solution

(a)

$$\dfrac{}{\langle \mathtt{n} \leftarrow \mathtt{n}\ +\ 2, \sigma\rangle \Downarrow \sigma[\mathtt{n} \mapsto 5]}$$

(b)

$$\dfrac{[\![\mathtt{0\ <\ n}]\!]^{\mathscr{B}}(\sigma) = \top \qquad \dfrac{}{\langle \mathtt{r} \leftarrow \mathtt{n}, \sigma\rangle \Downarrow \sigma[\mathtt{r} \mapsto 5]}}{\langle \mathtt{if\ \ (0\ <\ n)\ \ then}\ \mathtt{r} \leftarrow \mathtt{n}\ \mathtt{else}\ \mathtt{r} \leftarrow \mathtt{0\ -\ n}, \sigma\rangle \Downarrow \sigma[\mathtt{r} \mapsto 5]}$$

(c)

$$\dfrac{\dfrac{\dfrac{}{\langle \texttt{b} \leftarrow \texttt{a + b}, \sigma\rangle \Downarrow \sigma[\texttt{b} \mapsto 5]} \qquad \dfrac{}{\langle \texttt{a} \leftarrow \texttt{b - a}, \sigma[\texttt{b} \mapsto 5]\rangle \Downarrow \sigma[\texttt{b} \mapsto 5; \texttt{a} \mapsto 0]}}{\langle \texttt{b} \leftarrow \texttt{a + b;}\ \ \texttt{a} \leftarrow \texttt{b - a}, \sigma\rangle \Downarrow \sigma[\texttt{b} \mapsto 5; \texttt{a} \mapsto 0]} \qquad \dfrac{}{\langle \texttt{b} \leftarrow \texttt{b - a}, \sigma[\texttt{b} \mapsto 5; \texttt{a} \mapsto 0]\rangle}}{\langle \texttt{b} \leftarrow \texttt{a + b;}\ \ \texttt{a} \leftarrow \texttt{b - a;}\ \ \texttt{b} \leftarrow \texttt{b - a}, \sigma\rangle \Downarrow \sigma[\texttt{a} \mapsto 0; \texttt{b} \mapsto 5]}$$

(d) It’s a big tree. A note (on this and the above): I’m not particularly fussed about the associativity of sequential composition. You can build right-leaning trees if you want—although this one is going to already be leaning right because of the `while`.

$$\dfrac{[\![\texttt{0 < n}]\!]^{\mathscr{B}}(\sigma) = \top \qquad \dfrac{\dfrac{}{\langle \texttt{r} \leftarrow \texttt{n}, \sigma\rangle \Downarrow \sigma[\texttt{r} \mapsto 3]} \qquad \dfrac{}{\langle \texttt{n} \leftarrow \texttt{n - 2}, \sigma[\texttt{r} \mapsto 3]\rangle \Downarrow \sigma[\texttt{r} \mapsto 3; \texttt{n} \mapsto 1]}}{\langle \texttt{r} \leftarrow \texttt{n;}\ \ \texttt{n} \leftarrow \texttt{n - 2}, \sigma\rangle \Downarrow \sigma[\texttt{r} \mapsto 3; \texttt{n} \mapsto 1]} \qquad \dfrac{[\![\texttt{0 < n}]\!]^{\mathscr{B}}(\sigma[\texttt{r} \mapsto 3; \texttt{n} \mapsto 1]) = \top \qquad \vdots \qquad \dfrac{[\![\texttt{0 < n}]\!]^{\mathscr{B}}(\sigma[\texttt{r} \mapsto 1; \texttt{n} \mapsto -1])}{\langle s, \sigma[\texttt{r} \mapsto 1; \texttt{n} \mapsto -1]\rangle \Downarrow \sigma[\texttt{r} \mapsto 1; \texttt{n} \mapsto -1]}}{\langle s, \sigma[\texttt{r} \mapsto 3; \texttt{n} \mapsto 1]\rangle \Downarrow \sigma[\texttt{r} \mapsto 1; \texttt{n} \mapsto -1]}}{\langle s, \sigma\rangle \Downarrow \sigma[\texttt{r} \mapsto 1; \texttt{n} \mapsto -1]}$$

(e) It still doesn't terminate, so there is no derivation (recall that $\Downarrow$ only captures terminating executions!)

** 2. Without using procedures or blocks, write a While programme that computes the nth Pingala number $p(n)$—with p as defined below, where n is the value of variable `n` in the initial state.[a] The value of $p(n)$ should be, in the final state, stored in variable `r`. You may, if needed, assume that the initial value of `n` is non-negative. You may also, if needed, use variables other than `n` and `r` without worry.

$$p(n) = \begin{cases} 0 & \text{if } n \leq 0 \\ 1 & \text{if } n = 1 \\ p(n-2) + p(n-1) & \text{otherwise} \end{cases}$$

Hint: First work out the logic of turning the recursive definition into an iterative programme, then write it as a programme. Our language is restricted enough that it's a bit painful to do directly. Feel free to check that your programme is correct on a couple of small examples by unrolling execution traces.

[a] These are commonly known as Fibonacci number. Indian mathematician Pingala first wrote about them about 1000 years before Fibonacci.

Solution

```
if n < 1 {
 r ← 0
} else if n = 1 {
 r ← 1
} else {
 a ← 0
 b ← 1
 i ← n
 while 2 < i {
  t ← a
  a ← b
  b ← t + b
  i ← i - 1
 }
 r ← a + b
}
```

There are several ways of writing this programme. The one given above uses the loop to find the values of $p(n-2)$ and $p(n-1)$ (in `a` and `b`, respectively), then returns their sum as the final result. You could also shift the loop so that one of the two variables contains the final result instead.

** 3. Prove that our small-step semantics is functional.
Namely, show that, for any statement s and any states σ_0, σ and σ', if we have both $\langle s, \sigma_0 \rangle \to^* \langle \texttt{skip}, \sigma \rangle$ and $\langle s, \sigma_0 \rangle \to^* \langle \texttt{skip}, \sigma' \rangle$, then $\sigma = \sigma'$.
You may want to first argue that it is sufficient to show that the transition relation $\to$ itself is functional.

Solution

By induction on s, prove that $\to$ is a total function. Then conclude either by a composition of functions is a function, or by an induction on the length of the trace for those who need more computational intuition.

CORRECTION: The above sketch (and the hint) are incorrect given the fact that my not looking at my notes in the lecture gave you a strange combination of rules for `skip` and `;`. For example, consider configuration $\left\langle \texttt{skip}; s_2, \sigma \right\rangle$, which can step to both $\langle s_2, \sigma \rangle$ (by SSeqSkip) and $\left\langle \texttt{skip}; s_2, \sigma \right\rangle$ (by SSeqStep and SSkip).

An easy fix is to consider restricting the use of SSeqStep only to productive transitions.

A more proper fix requires a direct strong induction on the length of (for example) the trace $\langle s, \sigma_0 \rangle \to^* \left\langle \texttt{skip}, \sigma \right\rangle$. (Note that we can't simply assume both traces have the same length!)

An indirect proof First we prove that any sequence of derivation in our small step semantics gives rise to a sequence of derivation that start and end in the same configurations, but do not use rule (SSkip) either directly, or to justify an application of (SSeqStep) rule. (Technically, we will prove that for every sequence of transitions in our small-step semantics, there exists an equivalent sequence of transitions in the small-step semantics defined exactly as ours, but without the (SSkip) rule—we'll call such transitions *skipless* and denote the corresponding relation $\overset{skipless}{\to}$.)

We prove, by mathematical induction on k, that, forall $k \in \mathbb{N}$, $s, s' \in \mathscr{S}$ and $\sigma, \sigma' \in \text{State}$, if $\langle s, \sigma \rangle \to^k \left\langle s', \sigma' \right\rangle$, then $\langle s, \sigma \rangle \overset{skipless^*}{\to} \left\langle s', \sigma' \right\rangle$.

Case 0: If $\langle s, \sigma \rangle \to^0 \left\langle s', \sigma' \right\rangle$, then $\langle s, \sigma \rangle \overset{skipless^0}{\to} \left\langle s', \sigma' \right\rangle$.

Case $k+1$: If $\langle s, \sigma \rangle \to^{1+k} \left\langle s', \sigma' \right\rangle$, then there exist $s'' \in \mathscr{S}$ and $\sigma'' \in \text{State}$ such that $\langle s, \sigma \rangle \to \left\langle s'', \sigma'' \right\rangle$ and $\left\langle s'', \sigma'' \right\rangle \to^k \left\langle s', \sigma' \right\rangle$.

By induction hypothesis, we know that $\left\langle s'', \sigma'' \right\rangle \overset{skipless^*}{\to} \left\langle s', \sigma' \right\rangle$, and it is sufficient to show that the first step can be simulated without the (SSkip) rule. We do so by case analysis on the rule used to derive this first step.

Case (SSkip): In this case, we have $s = s''$ and $\sigma = \sigma''$, and the skipless derivation we obtained by induction hypothesis suffices.

Case (SSeqStep): In this case, we once again have $s = s''$ and $\sigma = \sigma''$, and the skipless derivation we obtained by induction hypothesis again suffices.

Other cases: In all other cases, the first transition is justifiable also in $\overset{skipless}{\to}$ and we can construct the following trace $\langle s, \sigma \rangle \overset{skipless}{\to} \left\langle s'', \sigma'' \right\rangle \overset{skipless^*}{\to} \left\langle s', \sigma' \right\rangle$.

Let us consider traces $\langle s, \sigma_0 \rangle \to^* \left\langle \texttt{skip}, \sigma_1 \right\rangle$ and $\langle s, \sigma_0 \rangle \to^* \left\langle \texttt{skip}, \sigma_2 \right\rangle$. By the result above, we know there exist corresponding skipless traces $\langle s, \sigma_0 \rangle \overset{skipless^*}{\to} \left\langle \texttt{skip}, \sigma_1 \right\rangle$ and $\langle s, \sigma_0 \rangle \overset{skipless^*}{\to} \left\langle \texttt{skip}, \sigma_2 \right\rangle$. A finite composition of partial functional relations is a partial functional relation, so it is sufficient to show that $\overset{skipless}{\to}$ is a partial function.

Let us prove, by induction on s, that if $\langle s, \sigma \rangle \overset{skipless}{\to} \left\langle s', \sigma' \right\rangle$ and $\langle s, \sigma \rangle \overset{skipless}{\to} \left\langle s'', \sigma'' \right\rangle$ then $\langle s_1, \sigma_1 \rangle = \langle s_2, \sigma_2 \rangle$.

Case `skip`: $\left\langle \texttt{skip}, \sigma \right\rangle \overset{skipless}{\to} \left\langle s', \sigma' \right\rangle$ cannot hold, so this case is vacuously true.

Case $\mathtt{v} \leftarrow a$: $\langle \mathtt{v} \leftarrow a, \sigma \rangle \overset{skipless}{\to} \left\langle s', \sigma' \right\rangle$ can only be derived using rule (SAss), which gives $s' = \texttt{skip}$ and $\sigma' = \sigma\left[\mathtt{v} \mapsto [\![a]\!]^{\mathscr{A}}(\sigma)\right]$. Similarly, $\langle \mathtt{v} \leftarrow a, \sigma \rangle \overset{skipless}{\to} \left\langle s'', \sigma'' \right\rangle$ can only be derived using rule (SAss), which gives $s'' = \texttt{skip}$ and $\sigma'' = \sigma\left[\mathrm{v} \mapsto [\![a]\!]^{\mathscr{A}}(\sigma)\right]$. Since $[\![\cdot]\!]^{\mathscr{A}}$ is functional (as proved in the Week 5 problem sheet), we can conclude this case.

Case $s_1 \,;\, s_2$**:** In this case, two possible rules apply, depending on whether $s_1 = \texttt{skip}$.

Subcase $s_1 = \texttt{skip}$**:** Since we are considering the skipless semantics, we know that there is no transition from configurations of the form $\langle s_1, \sigma\rangle$. It must therefore be that transition $\left\langle \texttt{skip} \;\; ; s_2, \sigma\right\rangle \overset{\mathit{skipless}}{\rightarrow} \left\langle s', \sigma'\right\rangle$ is taken by (SSeqSkip), and we have $s' = s_2$ and $\sigma' = \sigma$. Similarly, transition $\left\langle \texttt{skip} \;\; ; s_2, \sigma\right\rangle \overset{\mathit{skipless}}{\rightarrow} \left\langle s'', \sigma''\right\rangle$ must be taken by (SSeqSkip), and we have $s'' = s_2$ and $\sigma'' = \sigma$.

Subcase $s_1 \neq \texttt{skip}$**:** In this case, it must be that transition $\langle s_1 \,;\, s_2, \sigma\rangle \overset{\mathit{skipless}}{\rightarrow} \left\langle s', \sigma'\right\rangle$ is taken by (SSeqStep), and we therefore have a transition $\langle s_1, \sigma\rangle \overset{\mathit{skipless}}{\rightarrow} \left\langle s_1', \sigma'\right\rangle$ such that $s' = s_1' \,;\, s_2$. Similarly, transition $\langle s_1 \,;\, s_2, \sigma\rangle \overset{\mathit{skipless}}{\rightarrow} \left\langle s'', \sigma''\right\rangle$ is taken by (SSeqStep), and we therefore have a transition $\langle s_1, \sigma\rangle \overset{\mathit{skipless}}{\rightarrow} \left\langle s_1'', \sigma''\right\rangle$ such that $s'' = s_1'' \,;\, s_2$. By induction hypothesis on s_1, and given the two transitions above starting in configuration $\langle s_1, \sigma\rangle$, it must be that $s_1' = s_1''$—and therefore also $s' = s_1' \,;\, s_2 = s_1'' \,;\, s_2 = s''$; and $\sigma' = \sigma''$.

Other cases are simple consequences of the fact that $[\![\cdot]\!]^{\mathscr{B}}$ is functional, which means only one rule applies in any given configuration.

*** 4. Prove that the rules we gave for our big-step semantics are *complete* with respect to our small-step semantics.

Namely, show that, for any statement s and any states σ, σ' such that $\langle s, \sigma\rangle \rightarrow^* \left\langle \texttt{skip}, \sigma'\right\rangle$, there exists a big-step derivation for $\langle s, \sigma\rangle \Downarrow \sigma'$.

Note: We want to prove that the rules are in line with the definition we gave for $\Downarrow$, so don't use the definition itself in this proof—use the rules.

Hint: The proof is by *strong* mathematical induction on the length of the trace for $\langle s, \sigma\rangle \rightarrow^* \left\langle \texttt{skip}, \sigma'\right\rangle$. Strong mathematical induction works the same as mathematical induction, but in the inductive cases you get to assume that the induction hypothesis holds for all smaller values of k (instead of assuming it only for the immediate predecessor).

The idea of the proof is to "step" through the small-step trace and "construct" the big-step derivation step-by-step, but this doesn't work as smoothly as we'd like for sequential composition. (Can you see why?) In addition to strong induction, you may want to take a look at the properties you proved last week on the small-step semantics of sequential composition.

Solution

By strong mathematical induction on the length of the trace $\langle s, \sigma\rangle \rightarrow^* \left\langle \texttt{skip}, \sigma'\right\rangle$.

Case 0**:** In this case, $s = \texttt{skip}$, and therefore $\sigma' = \sigma$ and we have $\langle s, \sigma\rangle \Downarrow \sigma'$ by (BSkip).

Case $k+1$**:** Consider the first transition (it exists, since the trace has length at least 1). We carry out a case analysis on the rule that justifies it.

Case (SSkip)**:** As before.

Case (SAss)**:** In this case, $s = \mathrm{x} \leftarrow a$ and $\sigma' = \sigma[\mathrm{x} \mapsto [\![\mathrm{a}]\!]^{\mathscr{A}}(\sigma)]$, and we have $\langle s, \sigma\rangle \Downarrow \sigma'$ by (BAss). In both this case and that above, we note that the rest of the trace must be a sequence of (SSkip)s.

Case (SIf$_\top$)**:** Then $s = \texttt{if}\ b\ \texttt{then}\ s_1\ \texttt{else}\ s_2$, $[\![\mathtt{b}]\!]^{\mathscr{B}}(\sigma) = \top$, and the rest of the trace is a trace of length k for $\langle s_1, \sigma\rangle \to^* \langle \mathtt{skip}, \sigma'\rangle$. By induction hypothesis on this trace of length k, we have a derivation (say, D) for $\langle s_1, \sigma\rangle \Downarrow \sigma'$, and we can construct a derivation for $\langle s, \sigma\rangle \Downarrow \sigma'$ as follows

$$\dfrac{[\![\mathtt{b}]\!]^{\mathscr{B}}(\sigma) \qquad \dfrac{D}{\langle s_1, \sigma\rangle \Downarrow \sigma'}}{\langle s, \sigma\rangle \Downarrow \sigma'}$$

Case (SSeqStep)**:** This is the only difficult case (except if you also count the fact that you also need to prove it for (SWhileTrue)). If the first transition is given to us by (SSeqStep), then we cannot simply construct a big-step derivation, which relies on a full execution of s_1, followed by a full execution of s_2.

However, Question 8b) from Week 7 tells us that in this situation, there does exist an intermediate point on the trace where s_1 has been fully executed and s_2 has not yet started to execute. We but the trace at this point, and obtain two traces of length less than or equal to k, and to which the strong induction hypothesis applies.

Other cases are similar to (SIf$_\top$).

Blocks and Procedures

We now explore the While language extended with block structures and procedures.

** 5. Using only blocks, rewrite your solution to Question 2 so that, if the initial state is σ, then the final state is $\sigma[\mathtt{r} \mapsto p(\sigma(\mathtt{n}))]$, without any other changes (including to `n`).

Solution

```
if n < 1 {
 r ← 0
} else if n = 1 {
 r ← 1
} else {
 var a ← 0
 var b ← 1
 var i ← n
 while 2 < i {
  var t ← a
  a ← b
  b ← t + b
  i ← i - 1
 }
 r ← a + b
}
```

** 6. *Without using recursion*, rewrite your solution to Question 2 *as a procedure* `pingala` that reads its only parameter from top-level variable `n` and writes its only output to top-level variable `r`, and otherwise leaves the state at point of call unchanged.

Solution

```
proc pingala := {
 if n < 1 {
  r ← 0
 } else if n = 1 {
  r ← 1
 } else {
  var a ← 0
  var b ← 1
  var i ← n
  while 2 < i {
   var t ← a
   a ← b
   b ← t + b
   i ← i - 1
  }
  r ← a + b
 }
}
```

*** 7. Do the same as in Question 6 using recursion instead of a `while` loop.

Solution

```
proc pingala := {
 var x ← n
 var p ← 0
 proc aux := {
  if x < 1 {
   p ← 0
  } else if x = 1 {
   p ← 1
  } else {
   var t ← x
   var r ← 0
   x ← t - 1
   call aux
   r ← p
   x ← t - 2
   call aux
   p ← r + p
  }
 }
 call aux
 r ← p
}
```

This is to illustrate that recursion is not particularly nice in a procedure-based world: our procedures force us to define (in a somewhat ad hoc way, not as part of the language) *calling conventions* (where procedures expect to read their arguments from, and where they are expected to write their results to) and our programmes need to make sure that they follow them, ensuring not to clobber other variables (by scoping them in as a block), and further copying intermediate values in and out of the variables we use for argument and result passing.

In a sense, this should remind you (or at least those of you who learned some assembly) of register-based programming.

Induction over Derivations: Small- and Big-Step Semantics (Optional)

Go back to a time before you knew about blocks, procedures, and scoped variables.

As noted by one of your colleagues in the lecture, rule ($\mathsf{BWhile}_\mathsf{T}$) looks very recursive. But so did the corresponding small-step rule ($\mathsf{SWhile}_\mathsf{T}$).

When reasoning about properties of the small-step semantics in Sheet 7, you ended up reasoning by mathematical induction over the length of the trace, because reasoning by induction over the statement would have led to a circular proof on `while` statements.

In the big step semantics, we do not have a trace: we have a derivation for $\Downarrow$. Fortunately for us, the relation is defined inductively, and we can reason by induction over the derivation itself! Base cases are the axioms (rules with no premises), and inductive cases get to assume that the induction hypothesis holds on derivations of the premises.

As an example, the following is the start of a proof that $\Downarrow$ is a partial function.

Proof. We want to show that, if $\langle s, \sigma_0 \rangle \Downarrow \sigma$ and $\langle s, \sigma_0 \rangle \Downarrow \sigma'$, then $\sigma = \sigma'$.

By induction over the derivation of $\langle s, \sigma_0 \rangle \Downarrow \sigma$, we show that if there is a derivation of $\langle s, \sigma_0 \rangle \Downarrow \sigma'$ for some σ', then it must be that $\sigma = \sigma'$.

Case (BSkip): if $\langle s, \sigma_0 \rangle \Downarrow \sigma$ by (BSkip), then we have $s = \texttt{skip}$ and $\sigma = \sigma_0$. (BSkip) is the only rule to provide a derivation where the statement is `skip` and we therefore also have $\sigma' = \sigma_0$.

Case (BAss): if $\langle s, \sigma_0 \rangle \Downarrow \sigma$ by (BAss), then we have $s = \texttt{x} \leftarrow a$ for some variable $\texttt{x}$ and some arithmetic expression a, and $\sigma = \sigma_0\left[\texttt{x} \mapsto [\![\texttt{a}]\!]^{\mathscr{A}}(\sigma_0)\right]$. (BAss) is the only rule to provide a derivation where the statement is an assignment, therefore we also have $\sigma' = \sigma_0\left[\texttt{x} \mapsto [\![\texttt{a}]\!]^{\mathscr{A}}(\sigma_0)\right]$. We conclude by observing that $[\![\texttt{a}]\!]^{\mathscr{A}}(\cdot)$ is a function.

Case (BSeq): if $\langle s, \sigma_0 \rangle \Downarrow \sigma$ by (BSeq), then we have $s = s_1; s_2$ for some statements s_1 and s_2, and we have σ_1 such that $\langle s_1, \sigma_0 \rangle \Downarrow \sigma_1$ and $\langle s_2, \sigma_1 \rangle \Downarrow \sigma$. (BSeq) is the only rule to provide a derivation where the statement is a sequential composition, therefore we also have σ_1' such that $\langle s_1, \sigma_0 \rangle \Downarrow \sigma_1'$ and $\left\langle s_2, \sigma_1' \right\rangle \Downarrow \sigma'$.

By induction hypothesis (on the derivation of $\langle s_1, \sigma_0 \rangle \Downarrow \sigma_1$), we have $\sigma_1 = \sigma_1'$, and also therefore (by induction hypothesis on the derivation of $\langle s_2, \sigma_1 \rangle \Downarrow \sigma$), we have $\sigma = \sigma'$.

*** 8. Finish the proof.

Solution

See me.

*** 9. Prove that the rules we gave in the lecture for big-step semantics are *sound* with respect to our small-step semantics.

Namely, show that, for any statement s, and any states σ and σ', if $\langle s, \sigma \rangle \Downarrow \sigma'$, then $\langle s, \sigma \rangle \to^* \left\langle \texttt{skip}, \sigma' \right\rangle$.

> This proof will be by induction over the derivation for $\langle s, \sigma \rangle \Downarrow \sigma'$, and the idea of the proof is to construct a trace corresponding to the derivation. Once again, you may want to rely on results proved last week.

Solution

See me.