Lecture notes for CS6110 (Spring '09) taught by Andrew Myers at Cornell; edited by Amal Ahmed, Fall '09.

1 Inference rules and rule instances

We defined small-step and big-step semantics using inference rules. These rules are another kind of inductive definition. To prove properties of them, we would like to use well-founded induction.

To do this, we can change our view and look at reduction as a binary relation. To say that $\langle c, \sigma\rangle \longrightarrow \langle c', \sigma'\rangle$ according to the small-step SOS rules just means that $(\langle c, \sigma\rangle,\ \langle c', \sigma'\rangle)$ is a member of some reduction relation, which is a subset of $(Com \times \Sigma) \times (Com \times \Sigma)$. In fact, not only is it a relation, it is a partial function.

Here is an example of the kind of the rule we have been looking at so far.

$$\frac{a_1 \longrightarrow a_1'}{a_1 + a_2 \longrightarrow a_1' + a_2}\ (|a_1| > 0) \tag{1}$$

Here a_1, a_2, and a_1' are *metavariables*. Everything above the line is part of the *premise*, and everything below the line is the *conclusion*. The expression on the right side is a *side condition*.

A *rule instance* is a substitution for all the metavariables such that the side condition is satisfied. For example, here is an instance of the above rule:

$$\frac{3 * 4 \longrightarrow 12}{(3 * 4 + 1) \longrightarrow (12 + 1)}\ (|3 * 4| > 0)$$

where the substitutions are $a_1 = 3 * 4$, $a_1' = 12$, $a_2 = 1$.

Another valid instance of the rule is

$$\frac{3 * 4 \longrightarrow 11}{(3 * 4 + 1) \longrightarrow (11 + 1)}\ (|3 * 4| > 0)$$

where the substitutions are $a_1 = 3 * 4$, $a_1' = 11$, $a_2 = 1$.

Note that since an inductively defined set is defined by the rule instances, we are free to introduce metanotation in rules as long as it is clear what rule instances are generated.

With rules like (1), we are usually trying to define some set or relation. For example, this rule might be part of the definition of some reduction relation $\longrightarrow$ that is a subset of $AExp \times AExp$. Such rules are typically of the form

$$\frac{X_1\ X_2\ \ldots\ X_n}{X}\ (\phi) \tag{2}$$

where $X_1, X_2, \ldots, X_n$ represent elements that are already members of the set or relation being defined, X represents a new member of the relation added by this rule, and ϕ is a collection of side conditions that must hold in order for the rule to be applied.

The difference between a premise and a side condition is that the side condition is not part of the relation that the rule is trying to define, while the premises are. The side condition is some restriction that determines when an instance of the rule may be applied.

Now suppose we have written down a set of rules in an attempt to define a set A. How do we know whether A is well-defined? Certainly we would like to have $X \in A$ whenever $X_1, X_2, \ldots, X_n \in A$ and

$$\frac{X_1\ X_2\ \ldots\ X_n}{X}$$

is a rule instance, but this is hardly a definition of A. What do we put in A to start with?

2 Rule operator

One approach is to find a well-founded relation such that the rules constitute an inductively defined function, as described above. Define a *rule operator* R on sets as follows. Given a set B, let

$$R(B) \triangleq \{X \mid \{X_1, X_2, \ldots, X_n\} \subseteq B \text{ and } \frac{X_1\ X_2\ \ldots\ X_n}{X} \text{ is a rule instance}\}$$

Then

- $R(B)$ is the set of members of A that can be inferred from the members of set B;
- $R(\varnothing)$ is the set of members that can be inferred from nothing;
- $R(R(\varnothing))$ is the set of members that can be inferred from $R(\varnothing)$; the elements of $R(\varnothing)$ are in this set because they are inferred from the empty set, which is a subset of $R(\varnothing)$.

Formally, given a set of rules and a subset $B \subseteq S$, define a rule operator R that captures the inferences that can be made from the rule instances:

$$R(B) \triangleq \{X \mid \{X_1, X_2, \ldots, X_n\} \subseteq B \text{ and } \frac{X_1\ X_2\ \ldots\ X_n}{X} \text{ is a rule instance}\}. \tag{3}$$

Then R is a function mapping subsets of S to subsets of S; that is, $R : 2^S \to 2^S$, where 2^S denotes the *powerset* (set of all subsets) of S. An important property of R is that it is clearly *monotone*: if $B \subseteq C$, then $R(B) \subseteq R(C)$. This is because if $x \in R(B)$, then there must exist in B the premises $\{x_i\}$ for some rule that has x in the conclusion. Therefore, we have $\{x_i\} \subseteq B \subseteq C$. By the definition of R, the same rule can be used on C, so $x \in R(C)$ as well.

What set $A \subseteq S$ is defined by the rules? At the very least, we would like A to satisfy the following two properties:

- A is *R-consistent*: $A \subseteq R(A)$. We would like this to hold because we would like every element of A to be included in A only as a result of applying a rule.
- A is *R-closed*: $R(A) \subseteq A$. We would like this to hold because we would like every element that the rules say should be in A to actually be in A.

These two properties together say that $A = R(A)$, or in other words, A should be a fixed point of R. There are two natural questions to ask:

- Does R actually have a fixed point?
- Is the fixed point unique? If not, which one should we take?

3 Least fixed points

In fact, any monotone set operator R always has at least one fixed point. It may have many. But among all its fixed points, it has a unique minimal one with respect to set inclusion $\subseteq$; that is, a fixed point that is a subset of all other fixed points of R. We call this the *least fixed point* of R, and we take A to be this set.

The least fixed point of R can be defined in two different ways, "from below" and "from above":

$$A_* \triangleq \bigcup \{R^n(\varnothing) \mid n \geq 0\} = R(\varnothing) \cup R(R(\varnothing)) \cup R(R(R(\varnothing))) \cup \cdots \tag{4}$$

$$A^* \triangleq \bigcap \{B \subseteq S \mid R(B) \subseteq B\}. \tag{5}$$

The set A_* is the union of all sets of the form $R^n(\varnothing)$, the sets obtained by applying R some finite number of times to the empty set. It consists of all elements of S that are in $R^n(\varnothing)$ for some $n \geq 0$.

The set A^* is the intersection of all the R-closed subsets of S. It consists of all elements of S that are in every R-closed set.

We will show that $A_* = A^*$ and that this set is the least fixed point of R, so we will take $A \triangleq A_* = A^*$.

4 Proof

Obviously, $\varnothing \subseteq R(\varnothing)$. Since R is monotone, we can apply it to both sides, obtaining $R(\varnothing) \subseteq R^2(\varnothing)$. Clearly $R^n(\varnothing) \subseteq R^{n+1}(\varnothing)$ for all n. So we can see that the successive sets $R^n(\varnothing)$ in the union defining A_* are in a chain with respect to $\subseteq$:

$$\varnothing \subseteq R(\varnothing) \subseteq R^2(\varnothing) \subseteq R^3(\varnothing) \subseteq R^4(\varnothing) \subseteq \ldots$$

4.1 Closure

First, we show that A_* is R-closed: i.e., $R(A_*) \subseteq A_*$. Consider an arbitrary $x \in R(A_*)$. It must be there is a rule $\frac{x_1,\ldots x_n}{x}$ where all of the x_i are in A_*. Each of x_i must be found first in one of the sets $R^m(\varnothing)$. Since there are a finite number of premises (n), there must be a set $R^m(\varnothing)$ in the chain that includes all of the x_i. But then $R^{m+1}(\varnothing)$ must include x, so therefore so does A_*. Since this is true for arbitrary $x \in R(A_*)$, we have $R(A_*) \subseteq A_*$.

4.2 Consistency

We need to show that $A_* \subseteq R(A_*)$. Consider an arbitrary $x \in A_*$. We must have $x \in R^m(\varnothing)$ for some m. In this case, we know that all its premises x_i are found in $R^{m-1}(\varnothing)$. Since $R^{m-1}(\varnothing) \subseteq A_*$, we can apply R to both sides to obtain $R^m(\varnothing) \subseteq R(A_*)$. Therefore x must be in $R(A_*)$, and so A_* is R-consistent.

4.3 Least fixed point

Suppose we have another fixed point of R. Call it B. Given $B = R(B)$, we want to show that $A_* \subseteq B$. Clearly we have $\varnothing \subseteq B$. Applying R to both sides, we obtain $R(\varnothing) \subseteq R(B) = B$ We can repeat as necessary to obtain $R^n(\varnothing) \subseteq B$ (that is, this holds by induction on n). Therefore the union of all $R^n(\varnothing)$ cannot contain any elements not in B, so $A_* \subseteq B$. In fact, by the same argument, not only is A_* the least fixed point, it is the least R-closed set.

5 Other fixed points

In general the rule operator will have other fixed points. These correspond to allowing some proof trees with infinite derivations. For example, consider these three rules over $\mathbb{Z}$:

$$\frac{}{0} \quad \frac{n}{n+1} \quad \frac{}{1}$$

The least fixed point will consist of $\{n \mid n \geq 1\}$. The *greatest* fixed point will be $\mathbb{N}$, including 0, because $\{0, 1, \ldots\}$ is a fixed point of R, even though there is no finite proof tree deriving 0. The greatest fixed point is the union of all R-consistent sets.

However, we use the least fixed point because it ensures that all the proof trees are finite. Hence, the subderivation relation on proof trees is well-founded. This would not be the case with other fixed points.

6 Finiteness of premises

One importance feature of inference rules is that the number of premises is finite. We saw that was needed in the argument that A_* is R-closed. If inference rules have an infinite number of premises, A_* need not be closed! Consider these rules over $\mathbb{Z}$:

$$\frac{}{1} \quad \frac{n}{n+1} \quad \frac{m \quad {\scriptstyle (\forall m > n)}}{n}$$

In this case, the A_* construction gives us the set $\{1, 2, \ldots\}$, but $0 \in R(A_*)$.

1 Summary

In this lecture we:

- define induction on a well-founded relation;
- illustrate the definition with some examples, including the inductive definition of free variables $FV(e)$;

2 Introduction

Recall that some of the substitution rules mentioned the function $FV : \{\lambda\text{-terms}\} \to \mathbf{Var}$:

$$\begin{aligned} FV(x) &= \{x\} \\ FV(e_1\ e_2) &= FV(e_1) \cup FV(e_2) \\ FV(\lambda x.\, e) &= FV(e) - \{x\}. \end{aligned}$$

Why does this definition uniquely determine the function FV? There are two issues here:

- Existence: whether FV is defined on all λ-terms;
- Uniqueness: whether the definition is unique.

Of relevance here is the fact that there are three clauses in the definition of FV corresponding to the three clauses in the definition of λ-terms and that a λ-term can be formed in one and only one way by one of these three clauses. Note also that although the symbol FV occurs on the right-hand side in two of these three clauses, they are applied to proper (*proper* = strictly smaller) subterms.

The idea underlying this definition is called *structural induction*. This is an instance of a general induction principle called *induction on a well-founded relation*.

3 Well-Founded Relations

A binary relation $\prec$ is said to be *well-founded* if it has no infinite descending chains. An *infinite descending chain* is an infinite sequence of elements $a_0, a_1, a_2, \ldots$ such that $a_{i+1} \prec a_i$ for all $i \geq 0$. Note that a well-founded relation cannot be reflexive.

Here are some examples of well-founded relations:

- the successor relation $\{(m, m+1) \mid m \in \mathbb{N}\}$ on $\mathbb{N}$;
- the less-than relation $<$ on $\mathbb{N}$;
- the element-of relation $\in$ on sets. The axiom of foundation (or axiom of regularity) of Zermelo–Fraenkel (ZF) set theory asserts exactly that $\in$ is well-founded. Among other things, this prevents a set from being a member of itself;
- the proper subset relation $\subset$ on the set of finite subsets of $\mathbb{N}$.

The following are not well-founded relations:

- the predecessor relation $\{(m+1, m) \mid m \in \mathbb{N}\}$ on $\mathbb{N}$ ($0, 1, 2, \ldots$ is an infinite *descending* chain!);
- the greater-than relation $>$ on $\mathbb{N}$;
- the less-than relation $<$ on $\mathbb{Z}$ ($0, -1, -2, \ldots$ is an infinite descending chain);
- the less-than relation $<$ on the real interval $[0, 1]$ ($1, \frac{1}{2}, \frac{1}{3}, \frac{1}{4}, \ldots$ is an infinite descending chain);
- the proper subset relation $\subset$ on subsets of $\mathbb{N}$ ($\mathbb{N}, \mathbb{N} - \{0\}, \mathbb{N} - \{0, 1\}, \ldots$ is an infinite descending chain).

4 Well-Founded Induction

Let $\prec$ be a well-founded binary relation on a set A. Abstractly, a *property* is just a map $P : A \to \{true, false\}$, or equivalently, a subset $P \subseteq A$ (the set of all elements of A for which the property is true).

The principle of well-founded induction on the relation $\prec$ says that in order to prove that a property P holds for all elements of A, it suffices to prove that P holds of any $a \in A$ whenever P holds for all $b \prec a$. In other words,

$$\forall a \in A.(\forall b \in A.b \prec a \Rightarrow P(b)) \Rightarrow P(a) \quad \Rightarrow \quad \forall a \in A.P(a). \tag{1}$$

Expressed as a proof rule,

$$\frac{\forall a \in A.(\forall b \in A.b \prec a \Rightarrow P(b)) \Rightarrow P(a)}{\forall a \in A.P(a)}. \tag{2}$$

The basis of the induction is the case when a has no $\prec$-predecessors; in that case, the statement $\forall b \in A.b \prec a \Rightarrow P(b)$ is vacuously true.

For the well-founded relation $\{(m, m+1) \mid m \in \mathbb{N}\}$, (1) and (2) reduce to the familiar notion of mathematical induction on $\mathbb{N}$: to prove $\forall n.P(n)$, it suffices to prove that $P(0)$ and that $P(n+1)$ whenever $P(n)$.

For the well-founded relation $<$ on $\mathbb{N}$, (1) and (2) reduce to *strong* induction on $\mathbb{N}$: to prove $\forall n.P(n)$, it suffices to prove that $P(n)$ whenever $P(0), P(1), \ldots, P(n-1)$. When $n = 0$, the induction hypothesis is vacuously true.

4.1 Equivalence of Well-Foundedness and the Validity of Induction

In fact, one can show that the induction principle (1)–(2) is valid for a binary relation $\prec$ on A if and only if $\prec$ is well-founded.

To show that well-foundedness implies the validity of the induction principle, suppose the induction principle is not valid. Then there exists a property P for which the premise of (2) holds but not the conclusion. Thus P is false for some element $a_0 \in A$. The premise of (2) is equivalent to

$$\forall a \in A.\neg P(a) \Rightarrow \exists b \in A\ b \prec a \wedge \neg P(b)$$

This implies that there exists an $a_1 \prec a_0$ such that P is false for a_1. Continuing in this fashion, using the axiom of choice one can construct an infinite descending chain $a_0, a_1, a_2, \ldots$ for which P is false, so $\prec$ is not well-founded.

Conversely, suppose that there is an infinite descending chain $a_0, a_1, a_2, \ldots$. Then the property "$a \notin \{a_0, a_1, a_2, \ldots\}$" violates (2), since the premise of (2) holds but not the conclusion.

5 Structural Induction

Now let's define a well-founded relation on the set of all λ-terms. Define $e < e'$ if e is a *proper* subterm of e'. A λ-term e is a *proper* (or *strict*) subterm of e' if it is a subterm of e' and if $e \neq e'$. If we think of λ-terms as syntax trees, then e' is a tree that has e as a subtree. Since these trees are finite, the relation is well-founded. Induction on this relation is called *structural induction*.

We can now show that $FV(e)$ exists and is uniquely defined for any λ-term e. In the grammar for λ-terms, for any e, exactly one case in the definition of FV applies to e, and all references in the definition of FV are to subterms, which are strictly smaller. The function FV exists and is uniquely defined for the base case of the smallest λ-terms $x \in Var$. So $FV(e)$ exists and is uniquely defined for any λ-term e by induction on the well-founded subexpression relation.

We often have a set of expressions in a language built from a set of *constructors* starting from a set of *generators*. For example, in the case of λ-terms, the generators are the variables $x \in Var$ and the constructors are the application operator $\cdot$ and the abstraction operators λx. The set of expressions defined by the generators and constructors is the smallest set containing the generators and closed under the constructors.

If a function is defined on expressions in such a way that

- there is one clause in the definition for every generator or constructor pattern,
- the right-hand sides refer to the value of the function only on proper subexpressions,

then the function is well-defined and unique.

6 Rule Induction

Let us use our newfound wisdom on well-founded induction to prove some properties of the semantics we have seen so far.

6.1 Example 1: evaluation preserves closedness

Theorem If $e \to e'$ under the CBV reduction rules, then $FV(e') \subseteq FV(e)$. In other words, CBV reductions cannot introduce any new free variables.

Proof. By induction on the CBV derivation of $e \to e'$. There is one case for each CBV rule, corresponding to each way $e \to e'$ could be derived.

Case 1: $\dfrac{e_1 \to e_1'}{e_1\ e_2 \to e_1'\ e_2}$.

We assume that the desired property is true of the premise—this is the induction hypothesis—and we wish to prove under this assumption that it is true for the conclusion. Thus we are assuming that $FV(e_1') \subseteq FV(e_1)$ and wish to prove that $FV(e_1'\ e_2) \subseteq FV(e_1\ e_2)$.

$$\begin{aligned} FV(e_1'\ e_2) &= FV(e_1') \cup FV(e_2) \quad \text{by the definition of } FV \\ &\subseteq FV(e_1) \cup FV(e_2) \quad \text{by the induction hypothesis} \\ &= FV(e_1\ e_2) \quad \text{again by the definition of } FV. \end{aligned}$$

Case 2: $\dfrac{e_2 \to e_2'}{v\ e_2 \to v\ e_2'}$.

This case is similar to Case 1, where now $e_2 \to e_2'$ is used in the induction hypothesis.

Case 3: $\dfrac{}{(\lambda x.\, e)v \to e\{v/x\}}$.

There is no induction hypothesis for this case, since there is no premise in the rule; thus this case constitutes the basis of our induction. We wish to show, independently of any inductive assumption, that $FV(e\{v/x\}) \subseteq FV((\lambda x.\, e)v)$.

This case requires a lemma, stated below, to show that $FV(e\{v/x\}) \subseteq (FV(e) - \{x\}) \cup FV(v)$. Once that is shown, we have

$$\begin{aligned} FV(e\{v/x\}) &\subseteq (FV(e) - \{x\}) \cup FV(v) \quad \text{by the lemma to be proved} \\ &= FV(\lambda x.\, e) \cup FV(v) \quad \text{by the definition of } FV \\ &= FV((\lambda x.\, e)v) \quad \text{again by the definition of } FV. \end{aligned}$$

We have now considered all three rules of derivation for the CBV λ-calculus, so the theorem is proved. □

Lemma $FV(e\{v/x\}) \subseteq (FV(e) - \{x\}) \cup FV(v)$ (this lemma is used by case 3 in the above theorem).

Proof. By structural induction on e. There is one case for each clause in the definition of the substitution operator. We have assumed previously that values are closed terms, so $FV(v) = \emptyset$ for any value v; but actually we do not need this for the proof, and we do not assume it.

Case 1: $e = x$.

$$\begin{aligned} FV(e\{v/x\}) &= FV(x\{v/x\}) \\ &= FV(v) \quad \text{by the definition of the substitution operator} \\ &= (\{x\} - \{x\}) \cup FV(v) \\ &= (FV(x) - \{x\}) \cup FV(v) \quad \text{by the definition of } FV \\ &= (FV(e) - \{x\}) \cup FV(v). \end{aligned}$$

Case 2: $e = y$, $y \neq x$.

$$\begin{aligned} FV(e\{v/x\}) &= FV(y\{v/x\}) \\ &= FV(y) \quad \text{by the definition of the substitution operator} \\ &= \{y\} \quad \text{by the definition of } FV \\ &\subseteq (\{y\} - \{x\}) \cup FV(v) \\ &= (FV(y) - \{x\}) \cup FV(v) \quad \text{again by the definition of } FV \\ &= (FV(e) - \{x\}) \cup FV(v). \end{aligned}$$

Case 3: $e = e_1\ e_2$.

$$\begin{aligned} FV(e\{v/x\}) &= FV((e_1\ e_2)\{v/x\}) \\ &= FV(e_1\{v/x\}\ e_2\{v/x\}) \quad \text{by the definition of the substitution operator} \\ &\subseteq (FV(e_1) - \{x\}) \cup FV(v) \cup (FV(e_2) - \{x\}) \cup FV(v) \quad \text{by the induction hypothesis} \\ &= ((FV(e_1) \cup FV(e_2)) - \{x\}) \cup FV(v) \\ &= (FV(e_1\ e_2) - \{x\}) \cup FV(v) \quad \text{again by the definition of } FV \\ &= (FV(e) - \{x\}) \cup FV(v). \end{aligned}$$

Case 4: $e = \lambda x.\, e'$.

$$\begin{aligned} FV(e\{v/x\}) &= FV((\lambda x.\, e')\{v/x\}) \\ &= FV(\lambda x.\, e') \quad \text{by the definition of the substitution operator} \\ &= FV(\lambda x.\, e') - \{x\} \quad \text{because } x \notin FV(\lambda x.\, e') \\ &\subseteq (FV(e) - \{x\}) \cup FV(v). \end{aligned}$$

Case 5: $e = \lambda y.\, e'$, $y \neq x$. This is the most interesting case, because it involves a change of bound variable. Using the fact $FV(v) = \emptyset$ for values v would give a slightly simpler proof. Let v be a value and z a variable such that $z \neq x$, $z \notin FV(e')$, and $z \notin FV(v)$.

$$\begin{aligned} FV(e\{v/x\}) &= FV((\lambda y.\, e')\{v/x\}) \\ &= FV(\lambda z.\, e'\{z/y\}\{v/x\}) \quad \text{by the definition of the substitution operator} \\ &= FV(e'\{z/y\}\{v/x\}) - \{z\} \quad \text{by the definition of } FV \\ &= ((((FV(e') - \{y\}) \cup FV(z)) - \{x\}) \cup FV(v)) - \{z\} \quad \text{by the induction hypothesis twice} \\ &= (((FV(\lambda y.\, e') \cup \{z\}) - \{x\}) \cup FV(v)) - \{z\} \quad \text{by the definition of } FV \\ &= ((FV(\lambda y.\, e') - \{x\}) \cup FV(v) \cup \{z\}) - \{z\} \\ &= (FV(e) - \{x\}) \cup FV(v). \end{aligned}$$

□

There is a subtle point that arises in case 5. We said at the beginning of the proof that we would be doing structural induction on e; that is, induction on the well-founded subterm relation $<$. This was a lie. Because of the change of bound variable necessary in case 5, we are actually doing induction on the relation of subterm modulo α-equivalence:

$$e <_\alpha e' \quad \stackrel{\triangle}{=} \quad \exists e''.\ e'' < e' \ \wedge\ e =_\alpha e''.$$

But a moment's thought reveals that this definition is still well-founded, since α-reduction does not change the size or shape of the term, so we are okay.

6.2 Example 2: agreement of big-step and small-step semantics

As we saw earlier, we can express the idea that the two semantics should agree on terminating executions by connecting the $\longrightarrow^*$ and $\Downarrow$ relations:

$$\langle c,\, \sigma\rangle \longrightarrow^* \langle \mathbf{skip},\, \sigma'\rangle \quad \Longleftrightarrow \quad \langle c,\, \sigma\rangle \Downarrow \sigma'$$

This can be proved using induction. To prove the $\Longrightarrow$ direction, we can use induction on the length of the reduction sequence $\langle c,\, \sigma\rangle \longrightarrow^* \langle \mathbf{skip},\, \sigma'\rangle$. (**QUESTION:** Could we have used structural induction on c? Why or why not?) The $\Longleftarrow$ direction requires induction on the derivation of the big-step evaluation. We are given $\langle c,\, \sigma\rangle \Downarrow \sigma'$, so we know that there is a derivation. The form of the derivation depends on the form of c. Here we just sketch a few of the cases for c.

Proof Sketch. By induction on the derivation of $\langle c,\, \sigma\rangle \Downarrow \sigma'$.

- *Case:* $\dfrac{}{\langle \mathbf{skip},\, \sigma\rangle \Downarrow \sigma}$

  We are required to show that $\langle \mathbf{skip},\, \sigma\rangle \longrightarrow^* \langle \mathbf{skip},\, \sigma\rangle$ which is immediate by the definition of $\longrightarrow^*$. (See definition in lecture 5 notes, Section 2.) (Note: We know that there is no small-step rule for **skip**, since $\langle \mathbf{skip},\, \sigma\rangle$ is a final configuration, but this is not a problem.)

- *Case:* $\dfrac{\langle a,\, \sigma\rangle \Downarrow_a n}{\langle x := a,\, \sigma\rangle \Downarrow \sigma[x \mapsto n]}$

  In this case, we know from the premise that $\langle a,\, \sigma\rangle \Downarrow n$ for some n. We have to show that $\langle x := a,\, \sigma\rangle \longrightarrow^* \langle \mathbf{skip},\, \sigma[x \mapsto n]\rangle$.

  We will need a lemma that $\langle a,\, \sigma\rangle \Downarrow n \implies \langle a,\, \sigma\rangle \longrightarrow^* n$. This can be proved using the same technique being used on commands. We will also need a lemma showing that $\langle a,\, \sigma\rangle \longrightarrow^* n$ implies $\langle x := a,\, \sigma\rangle \longrightarrow^* \langle x := n,\, \sigma\rangle$. This obvious result can be proved easily using induction on the number of steps taken.

  Given these lemmas, we have $\langle x := a,\, \sigma\rangle \longrightarrow^* \langle x := n,\, \sigma\rangle$ and by the small-step operational semantics we have $\langle x := n,\, \sigma\rangle \longrightarrow \langle \mathbf{skip},\, \sigma[x \mapsto n]\rangle$. Hence, we have $\langle x := a,\, \sigma\rangle \longrightarrow^* \langle \mathbf{skip},\, \sigma[x \mapsto n]\rangle$.

- *Case:* $\dfrac{\langle b,\, \sigma\rangle \Downarrow_b \mathbf{false}}{\langle \mathbf{while}\ b\ \mathbf{do}\ c,\, \sigma\rangle \Downarrow \sigma}$

  We are required to show that $\langle \mathbf{while}\ b\ \mathbf{do}\ c,\, \sigma\rangle \longrightarrow^* \langle \mathbf{skip},\, \sigma\rangle$.

  We will need a lemma that $\langle b,\, \sigma\rangle \Downarrow t \implies \langle b,\, \sigma\rangle \longrightarrow^* t$. This can be proved using the same technique being used on commands. We will also need a lemma showing that $\langle b,\, \sigma\rangle \longrightarrow^* t$ implies $\langle \mathbf{while}\ b\ \mathbf{do}\ c,\, \sigma\rangle \longrightarrow^* \langle \mathbf{while}\ t\ \mathbf{do}\ c,\, \sigma\rangle$. Again, this can be proved easily using induction on the number of steps taken.

  Given these two lemmas, we have that $\langle \mathbf{while}\ b\ \mathbf{do}\ c,\, \sigma\rangle \longrightarrow^* \langle \mathbf{while\ false\ do}\ c,\, \sigma\rangle$, and from the small-step operational semantics we have $\langle \mathbf{while\ false\ do}\ c,\, \sigma\rangle \longrightarrow \langle \mathbf{skip},\, \sigma\rangle$. Hence, we have $\langle \mathbf{while}\ b\ \mathbf{do}\ c,\, \sigma\rangle \longrightarrow^* \langle \mathbf{skip},\, \sigma\rangle$.

- *Case:* $\dfrac{\langle b, \sigma\rangle \Downarrow_b \mathbf{true} \qquad \langle c, \sigma\rangle \Downarrow \sigma' \qquad \langle \mathbf{while}\, b\, \mathbf{do}\, c, \sigma'\rangle \Downarrow \sigma''}{\langle \mathbf{while}\, b\, \mathbf{do}\, c, \sigma\rangle \Downarrow \sigma''}$

  We are required to show that $\langle \mathbf{while}\, b\, \mathbf{do}\, c, \sigma\rangle \longrightarrow^* \langle \mathbf{skip}, \sigma''\rangle$.

  This is the most interesting case in the whole proof. We need one more obvious lemma for stiching together small-step executions:

  $$\langle c_1, \sigma\rangle \longrightarrow^* \langle \mathbf{skip}, \sigma'\rangle \ \wedge\ \langle c_2, \sigma'\rangle \longrightarrow^* \langle \mathbf{skip}, \sigma''\rangle \implies \langle c_1; c_2, \sigma\rangle \longrightarrow^* \langle \mathbf{skip}, \sigma''\rangle \tag{3}$$

  This can be proved by induction on the number of steps.

  Now, from the premises we have that $\langle c, \sigma\rangle \Downarrow \sigma'$ and $\langle \mathbf{while}\, b\, \mathbf{do}\, c, \sigma'\rangle \Downarrow \sigma''$. Furthermore, because the derivations of these two assertions are subderivations of that for $\langle \mathbf{while}\, b\, \mathbf{do}\, c, \sigma\rangle \Downarrow \sigma''$, it follows by the induction hypothesis that $\langle c, \sigma\rangle \longrightarrow^* \langle \mathbf{skip}, \sigma'\rangle$ and that $\langle \mathbf{while}\, b\, \mathbf{do}\, c, \sigma'\rangle \longrightarrow^* \langle \mathbf{skip}, \sigma''\rangle$. Using Lemma (3), we have that $\langle c; \mathbf{while}\, b\, \mathbf{do}\, c, \sigma\rangle \longrightarrow^* \langle \mathbf{skip}, \sigma''\rangle$.

  Now consider the small-step side. We have an initial step

  $$\langle \mathbf{while}\, b\, \mathbf{do}\, c, \sigma\rangle \longrightarrow \langle \mathbf{if}\, b\, \mathbf{then}\, (c; \mathbf{while}\, b\, \mathbf{do}\, c)\, \mathbf{else}\, \mathbf{skip}, \sigma\rangle$$

  From prior lemmas, we know that this will step to $\langle c; \mathbf{while}\, b\, \mathbf{do}\, c, \sigma\rangle$, which we just showed will step to $\langle \mathbf{skip}, \sigma''\rangle$ as desired.

Notice that we could not have used structural induction on c for this proof because in this last case the induction step involved relating an evaluation of the command $\mathbf{while}\, b\, \mathbf{do}\, c$ to a different evaluation of the same command rather than to an evaluation of a subexpression.

7 Remark

The value of the reasoning framework we have set up is that formal reasoning about the semantics of programming languages, including such seemingly complicated notions as reductions and substitutions, can be reduced to the mindless application of a few simple rules. There is no hand-waving or magic involved. There is nothing hidden, it is all right there in front of you. To the extent that we can do this for real programming languages, we will be better able to understand what is going on.