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:

$$e <_\alpha e' \quad \triangleq \quad \exists e''\ e'' < e' \wedge e =_\alpha e''.$$

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

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:

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

This can be proved using induction. To prove the $\Rightarrow$ direction, we can use structural induction on c. The $\Leftarrow$ 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 show just a few of the cases for c.

- Case **skip**. In this case we know $\sigma = \sigma'$, and trivially, $\langle \mathbf{skip}, \sigma\rangle \longrightarrow^* \langle \mathbf{skip}, \sigma\rangle$.
- Case $x := a$. In this case we know from the premises that $\langle a, \sigma\rangle \Downarrow n$ for some n, and that $\sigma' = \sigma[x \mapsto n]$.

  We will need a lemma that $\langle a, \sigma\rangle \Downarrow n \Rightarrow \langle a, \sigma\rangle \longrightarrow^* n$. 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 an induction on the number of steps taken.

  Given these lemmas, we have $\langle x := a, \sigma\rangle \longrightarrow^* \langle x := n, \sigma\rangle$ and $\langle x := n, \sigma\rangle \longrightarrow \langle \mathbf{skip}, \sigma[x \mapsto n]\rangle$, so $\langle x := a, \sigma\rangle \longrightarrow^* \langle \mathbf{skip}, \sigma[x \mapsto n]\rangle$.
- Case **while** b **do** c, where b evaluates to *false*. In this case we have $\langle b, \sigma\rangle \Downarrow \mathit{false}$ and $\sigma = \sigma'$. Consider what happens in the small-steps semantics. Given two more lemmas, that $\langle b, \sigma\rangle \Downarrow t \Rightarrow \langle b, \sigma\rangle \longrightarrow^* t$, and that $\langle b, \sigma\rangle \longrightarrow^* t \Rightarrow \langle \mathbf{while}\ b\ \mathbf{do}\ c, \sigma\rangle \longrightarrow^* \langle \mathbf{while}\ t\ \mathbf{do}\ c, \sigma\rangle$, we have $\langle \mathbf{while}\ b\ \mathbf{do}\ c, \sigma\rangle \longrightarrow^* \langle \mathbf{skip}, \sigma\rangle$, as desired.
- Case **while** b **do** c, where b evaluates to *true*. This is the most interesting case in the whole proof. We need one more obvious lemma for stitching 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, because $\langle \mathbf{while}\ b\ \mathbf{do}\ c, \sigma\rangle \Downarrow \langle \mathbf{skip}, \sigma'\rangle$, we know that $\langle c, \sigma\rangle \Downarrow \sigma''$ and $\langle \mathbf{while}\ b\ \mathbf{do}\ c, \sigma''\rangle \Downarrow \sigma'$. Further, because the derivations of these two assertions are subderivations of that for $\langle \mathbf{while}\ b\ \mathbf{do}\ c, \sigma\rangle$, the induction hypothesis gives us that $\langle c, \sigma\rangle \longrightarrow^* \langle \mathbf{skip}, \sigma''\rangle$ and that $\langle \mathbf{while}\ b\ \mathbf{do}\ c, \sigma''\rangle \longrightarrow^* \sigma'$. Using Lemma 3, we have $\langle c; \mathbf{while}\ b\ \mathbf{do}\ c, \sigma\rangle \longrightarrow^* \langle \mathbf{skip}, \sigma'\rangle$.

  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 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 for this proof, because the induction step involved relating an evaluation of the command **while** b **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.