Scribe: Hongzhou Liu and Misha Zatsman Lecturer: Andrew Myers

1 Overview

Induction is used to prove hard things:

- equivalence of semantics
- termination
- determinism
- equivalence of programs
- equivalence of expressions (used in optimization)

2 Equivalence

Winskel gives the following definition of equivalence of programs and commands:

$$c_1 \sim c_2 : \forall \sigma, \sigma_0 \langle c_1, \sigma_0 \rangle \Downarrow \sigma \iff \langle c_2, \sigma_0 \rangle \Downarrow \sigma$$

We can also state the definition in small-step semantics:

$$c_1 \sim c_2 : \forall \sigma, \sigma_0 \langle c_1, \sigma_0 \rangle \mapsto^* \langle \mathbf{skip}, \sigma \rangle \iff \langle c_2, \sigma_0 \rangle \mapsto^* \langle \mathbf{skip}, \sigma \rangle$$

If we only care about some of the output we can simply compare a projection of σ . Also we can restrict the input to a subset: $\Sigma' \subset \Sigma$ so that our rule only holds $\forall \sigma, \sigma_0 \in \Sigma'$.

Equivalence of commands is a little trickier than equivalence of programs. If $c_1 \sim c_2$ then any program c that contains c_1 should be equivalent to the same program with c_2 in place of c_1.

We define a command context to be a program with a hole in it. We can give the context a command to insert in the hole and we get a program. Here's a sample command context:

```
if x < y then (y:=x-y, []) else skip
```

Formally stated, a command context is a function $\mathcal{C} : \text{Com} \to \text{Com}$. We can define it using BNF:

$$\mathcal{C} \quad ::= \quad [] \mid \mathcal{C}; c \mid c; \mathcal{C} \mid \mathbf{if}\ b\ \mathbf{then}\ \mathcal{C}\ \mathbf{else}\ c \mid \mathbf{if}\ b\ \mathbf{then}\ c\ \mathbf{else}\ \mathcal{C} \mid \mathbf{while}\ b\ \mathbf{do}\ \mathcal{C}$$

$\forall \mathcal{C}, \mathcal{C}[c_1] \sim \mathcal{C}[c_2]$ is called "term equivalence". A term is an expression representing a computation. A declaration is an example of an expression not representing computation. In Imp all expressions are terms.

3 Types of Induction

3.1 Standard Induction

$$\frac{P(1) \qquad \forall n \geq 1 \,.\, P(n) \Rightarrow P(n+1)}{\forall n \geq 1 \,.\, P(n)}$$

3.2 Course-of-Value Induction

$$\frac{P(1) \qquad \forall n \geq 1 \;.\; (\forall m \in [1..n] \;.\; P(m)) \Rightarrow P(n+1)}{\forall n \geq 1 \;.\; P(n)}$$

Note that course-of-value induction on $P(n)$ is the same as performing standard induction on $P'(n) \equiv \forall m \in [1..n] \;.\; P(m)$.

3.3 Structural Induction

$$\frac{(\forall e' \prec e \;.\; P(e')) \Rightarrow P(e)}{\forall e \;.\; P(e)}, \text{ where } e' \text{ is a subexpression of e}$$

Note that the base cases (axioms) of structural induction are included for free in this definition. Since axioms have no premises, the induction hypothesis is vacuously true; thus, the induction hypothesis gives us no help in proving the property holds for the base cases.

3.4 Well-founded Induction

$$\frac{\forall a \;.\; (\forall a' \prec a \;.\; P(a')) \Rightarrow P(a)}{\forall a \;.\; P(a)}$$

Well-founded induction is a generalization of induction, and it is used to show $a \in A \Rightarrow P(a)$. We need $\prec$ to be a well-founded relation, meaning there are no infinite descending chains: $\ldots \prec a_3 \prec a_2 \prec a_1$. This condition implies that $\prec$ is irreflexive and that there are no cycles in the $\prec$ relation. Note that the lack of infinite descending chains does not imply that A is finite.

We can derive normal induction from well-founded induction by choosing $A \equiv \mathbb{N}$, $x \prec y \equiv x + 1 = y$ and course-of-values induction by choosing $A \equiv \mathbb{N}$, $x \prec y \equiv x < y$

3.5 Induction on Derivation

Suppose that α represents any provable assertion and we would like to show that $P(\alpha)$ holds for all such assertions. For each such α there must be a proof tree:

$$\frac{\begin{array}{ccc}\vdots & \vdots & \vdots \\ \alpha_1 & \alpha_2 & \alpha_3\end{array}}{\alpha}$$

We define the well-founded relation $\prec$ on proof trees as:

$$D_1 \prec D_2 \iff D_1 \textit{ is contained in } D_2$$

One example from the proof tree above would be if we call D the tree concluding α, D_1 the proof tree concluding α_1 and D_2 the proof tree concluding α_2. $D_1 \prec D$ and $D_2 \prec D$, but $D_2 \not\prec D_1$ (in general). We obtain the following rule from well-founded induction:

$$\frac{\forall \alpha \;.\; (\forall i \in 1..n \;.\; P(\alpha_i)) \Rightarrow P(\alpha)}{\forall \alpha \;.\; P(\alpha)}$$

We can use induction on derivations to prove determinism of IMP. Determinism is:

$$\forall \sigma, \sigma_1, \sigma_2 \quad \langle c, \sigma \rangle \Downarrow \sigma_1 \wedge \langle c, \sigma \rangle \Downarrow \sigma_2 \Rightarrow \sigma_1 = \sigma_2$$

Let:

$$\alpha \equiv \langle c, \sigma \rangle \Downarrow \sigma_1$$

$$P(\alpha) \equiv \forall \sigma_2 \langle c, \sigma \rangle \Downarrow \sigma_2 \Rightarrow \sigma_1 = \sigma_2$$

The only tricky part of determinism is proving while statements to be deterministic, since the conclusion is contained is structurally identical to one of its premises. Assuming we have proven everything else to be deterministic (notably boolean expressions) the proof is rather simple when we use induction on derivations:

$$\frac{\langle b, \sigma \rangle \Downarrow \textbf{true} \quad \langle c, \sigma \rangle \Downarrow \sigma_3 \quad \langle \textbf{while } b \textbf{ do } c,\ \sigma_3 \rangle \Downarrow \sigma_1}{\langle \textbf{while } b \textbf{ do } c,\ \sigma \rangle \Downarrow \sigma_1}$$

$$\frac{\langle b, \sigma \rangle \Downarrow \textbf{true} \quad \langle c, \sigma \rangle \Downarrow \sigma_4 \quad \langle \textbf{while } b \textbf{ do } c,\ \sigma_4 \rangle \Downarrow \sigma_2}{\langle \textbf{while } b \textbf{ do } c,\ \sigma \rangle \Downarrow \sigma_2}$$

$\sigma_3 = \sigma_4$ by the induction hypothesis, therefore the two while expressions in the premises are also equivalent. By the induction hypothesis (since they have smaller proof trees) the expression is deterministic and therefore they evaluate to the same value, therefore $\sigma_1 = \sigma_2$.