

PROVING PROPERTIES OF PROGRAMS USING OPERATIONAL RULES

We can use the rules for the simple imperative language to prove properties about particular programs. For instance, if we start with the program w :

while b do c end

Where b is any relational expression and c any command, we can show that this equivalent to the program:

if b then $c; w$ else nop end

What we want to prove is:

$$[w, \sigma] \rightarrow \sigma' \text{ iff } [\text{if } b \text{ then } c; w \text{ else nop}, \sigma] \rightarrow \sigma'$$

Since *iff* means \Leftrightarrow , we need to prove the double implication as a forward implication and a backward one. For the forward one, we assume the left hand side and try to prove the right. Since $[w, \sigma] \rightarrow \sigma'$, there must be a derivation using the rules that ends with it as a consequence. Since w is a while loop, the derivation must end with either the rule for the test false, or for the test true. For the former, the whole derivation must have this form:

$$\frac{\begin{array}{c} \vdots \\ [b, \sigma] \rightarrow \text{false} \end{array}}{[w, \sigma] \rightarrow \sigma}$$

Since we are assuming the consequent, we can use exactly this derivation of b , and we can build a derivation of if b then $c; w$ else nop:

$$\frac{\begin{array}{c} \vdots \\ [b, \sigma] \rightarrow \text{false} \end{array} \quad \overline{[\text{nop}, \sigma] \rightarrow \sigma}}{[\text{if } b \text{ then } w; c \text{ else nop end}, \sigma] \rightarrow \sigma}$$

For the case where the test is true, the derivation must have the form:

$$\frac{\begin{array}{c} \vdots \\ [b, \sigma] \rightarrow \text{true} \end{array} \quad \begin{array}{c} \vdots \\ [c, \sigma] \rightarrow \sigma'' \end{array} \quad \begin{array}{c} \vdots \\ [w, \sigma''] \rightarrow \sigma' \end{array}}{[w, \sigma] \rightarrow \sigma'}$$

The second two derivations above the line can give a derivation:

$$\frac{\begin{array}{c} \vdots \\ [c, \sigma] \rightarrow \sigma'' \end{array} \quad \begin{array}{c} \vdots \\ [w, \sigma''] \rightarrow \sigma' \end{array}}{[c; w, \sigma] \rightarrow \sigma'}$$

The derivation for b as true, and this derivation then give a derivation of if b then $c; w$ else nop:

$$\frac{\begin{array}{c} \vdots \\ [b, \sigma] \rightarrow \text{true} \end{array} \quad \frac{\begin{array}{c} \vdots \\ [c, \sigma] \rightarrow \sigma'' \end{array} \quad \begin{array}{c} \vdots \\ [w, \sigma''] \rightarrow \sigma' \end{array}}{[c; w, \sigma] \rightarrow \sigma'}}{[\text{if } b \text{ then } c; w \text{ else nop}, \sigma] \rightarrow \sigma'}$$

We have thus shown that the forward implication is true because it is true whatever derivational form occurs. To prove the backward form of the implication, assume $[\text{if } b \text{ then } c; w \text{ else nop end}, \sigma] \rightarrow \sigma'$. There must be a derivation using the rules for if when the test is true or the test is false. The false case is :

$$\frac{\vdots}{\frac{[b, \sigma] \rightarrow false \quad [nop, \sigma] \rightarrow \sigma}{[if\ b\ then\ c; \ w\ else\ nop, \sigma] \rightarrow \sigma}}$$

and the true case is:

$$\frac{\frac{\vdots}{[b, \sigma] \rightarrow true} \quad \frac{\vdots}{[c; w, \sigma] \rightarrow \sigma'}}{[if\ b\ then\ c; \ w\ else\ nop, \sigma] \rightarrow \sigma'}$$

From the first case, we can construct a derivation of $[w, \sigma] \rightarrow \sigma$ by appealing to the while rule when the test is false. From the second case we can also construct $[w, \sigma] \rightarrow \sigma'$ by noting that:

$$\frac{\frac{\vdots}{[c, \sigma] \rightarrow \sigma''} \quad \frac{\vdots}{[w, \sigma''] \rightarrow \sigma'}}{[c; w, \sigma] \rightarrow \sigma'}$$

for any store σ'' , and the derivations above the line can be assembled with with the

derivation of b true to give:

$$\frac{\frac{\vdots}{[b, \sigma] \rightarrow true} \quad \frac{\frac{\vdots}{[c, \sigma] \rightarrow \sigma''} \quad \frac{\vdots}{[w, \sigma''] \rightarrow \sigma'}}{[c; w, \sigma] \rightarrow \sigma'}}{[w, \sigma] \rightarrow \sigma'}$$

Thus the *iff* has been proved both ways and thus $w \equiv if\ b\ then\ c; \ w\ else\ nop\ end$

INDUCTIVE PROOFS

Some properties of general programs need to be proved by using induction. One such form is called structural induction since its principles are based on the structure of syntactic forms. If we take the arithmetic expression syntax:

$$E = N \mid I \mid E \circ_A E$$

we will prove that arithmetic expressions are deterministic, i.e. that:

$$[a, \sigma] \rightarrow m \wedge [a, \sigma] \rightarrow m' \Rightarrow m = m'$$

The inductive principle for structural induction says that if a particular property can be proved for the terminal expression (N and I) and is preserved by all ways of forming expressions from other ones. This technique is similar to mathematical induction, and is also a special case of well-founded induction, which we will describe later. The inductive hypothesis is:

$$P(a) \text{ iff } \forall m, m', \sigma. [a, \sigma] \rightarrow m \wedge [a, \sigma] \rightarrow m' \Rightarrow m = m'$$

The inductive principle can be expressed as:

$$\begin{aligned} & (\forall m \in N. P(m)) \wedge (\forall X \in I. P(X)) \wedge \\ & (\forall a_0, a_1 \in E, o \in op_A. P(a_0) \wedge P(a_1) \Rightarrow P(a_0 \circ_A a_1)) \\ & \Rightarrow \\ & \forall a \in E. P(a) \end{aligned}$$

To prove this, take each case in turn. If $a \equiv N$, there is only one operational rule, and there can only be one answer, so $m = m' = n$.

If $a \equiv I$, again there is only one rule, and we have $m = \sigma(I) = m'$

For arithmetic expressions, we will show addition; the others are similar. If $a \equiv a_0 + a_1$, assuming

$[a, \sigma] \rightarrow m \wedge [a, \sigma] \rightarrow m'$ for both a_0 and a_1 , and using the rule for addition, we have

$[a_0, \sigma] \rightarrow m_0$ and $[a_1, \sigma] \rightarrow m_1$ with $m = m_0 + m_1$

and

$[a_0, \sigma] \rightarrow m'_0$ and $[a_1, \sigma] \rightarrow m'_1$ with $m' = m'_0 + m'_1$

Using the induction hypothesis, we have $m_0 = m'_0$, and $m_1 = m'_1$. Thus $m = m_0 + m_1 = m'_0 + m'_1 = m'$.

The cases for the other operators are similar. We can then conclude, using the principle of structural induction, that $P(a)$ is true for any a .

WELL-FOUNDED INDUCTION

Mathematical induction and structural induction are examples of a more general scheme called well-founded induction. It is based on the idea of a well-founded relation, written $<$, on a set A . This is defined such that the set A has not infinite descending chains $\dots a_i < \dots < a_1 < a_0$. This can be shown to be equivalent to the statement that $<$ is a well-founded relation iff any non-empty subset Q of A has a minimal element m such that $m \in Q \wedge \forall b < m. b \notin Q$.

From this definition, we can define the principle of well-founded induction for a set A by the implication

$$\forall a \in A. ([\forall b \in A. b < a \Rightarrow P(b)] \Rightarrow P(a)) \Rightarrow \forall a \in A. P(a)$$

To see why this is true, assume that it is false, and prove a contradiction. If it is false, the left hand side must be true and the right hand false. If the right hand side is false, then $\sim P(a)$ for some $a \in A$. Since every subset of A has a minimal element, there must be a minimal element m of $\{a \in A \mid \sim P(a)\}$, therefore

$[\forall b \in A. b < m \Rightarrow P(b)] \Rightarrow P(m)$ from the assumption of the left-hand side. Now either m is minimal in A or it is not. If it is, there is no lesser element, $[\forall b \in A. b < m \Rightarrow P(b)]$ is trivially true, and then $P(m)$ is true. If m is not

minimal in A , $P(b)$ must be true, because otherwise, b would be in $\{a \in A \mid \sim P(a)\}$ but not minimal (since m is). Thus $P(m)$ is true either way. Since m is a member of $\{a \in A \mid \sim P(a)\}$, we must have $\sim P(m)$, which is the contradiction we were looking for. Therefore, the principle is proved.

USING WELL-FOUNDED INDUCTION ON OPERATIONAL DERIVATION TREES

We will use the principle just proved to prove an important result for our imperative language – that it is deterministic. We have already proved that expressions are deterministic using structural induction, now we show the same for commands.

The rules we have define a set that consists of all the elements for which there is a derivation – basically all syntactically correct programs. These derivations have one of the following two forms, the first for axioms, the second for non-axioms.

$$\text{— or } \frac{\begin{array}{ccc} \vdots & \dots & \vdots \\ x_1 & & x_n \end{array}}{x}$$

A rule instance is obtained by substituting real expressions for the x s. Since every rule has a finite set of premises, we can write all rules as a pair $[X, y]$ where X is a (possibly empty) set of premises and y is the conclusion.

Call R the set of all such rule instances. An R -derivation of y is then either $[\emptyset, y]$ or $[\{d_1, \dots, d_n\}, y]$ where

$[\{x_1, \dots, x_n\}, y]$ is a rule instance, and the d_i are R -derivations of the corresponding x_i .

Now we can write $d \Vdash_R y$ to mean d is an R -derivation of y . Thus

$[\emptyset, y] \Vdash_R y$ if $[\emptyset, y] \in R$, and

$[\{d_1, \dots, d_n\}, y] \Vdash_R y$ if $[\{x_1, \dots, x_n\}, y] \in R$, and $d_i \Vdash_R x_i$ for all i .

We will abbreviate these to $d \Vdash y$. If y is derivable we simply write $\Vdash y$.

Now we look at the relationship between derivations. If d has the form $[D, y]$ and $d' \in D$, then we can write this as $d' \prec_1 d$, i.e. d' is a sub-derivation of d . The transitive closure of this relation is \prec , and because all derivations are finite (at least of terminating programs), this relation is well-founded (no infinite chains), the minimal element being the derivation with the whole program in its conclusion.

We are now in a position to express our program property – determinism. For a configuration $[c, \sigma_0]$ where c is any command and σ_0 is any store:

$$\forall \sigma, \sigma_1. [c, \sigma_0] \rightarrow \sigma \wedge [c, \sigma_0] \rightarrow \sigma_1 \Rightarrow \sigma = \sigma_1$$

i.e. that a command cannot result in two different stores.

The statement of the induction hypothesis is:

$$P(d) \Leftrightarrow \forall c \in \text{commands}, \sigma, \sigma_0, \sigma_1 \in \text{stores}. d \Vdash [c, \sigma_0] \rightarrow \sigma \wedge \Vdash [c, \sigma_0] \rightarrow \sigma_1 \Rightarrow \sigma = \sigma_1$$

(note that only the first rule instance mentions the derivation d)

By the principle of well-founded induction, we will show that $\forall d' \prec d. P(d') \Rightarrow P(d)$ which will prove the required result. Assume the left-hand side, and let $d \Vdash [c, \sigma_0] \rightarrow \sigma \wedge \Vdash [c, \sigma_0] \rightarrow \sigma_1$. Then $\Vdash [c, \sigma_0] \rightarrow \sigma_1$ for some d_1 . By examining the different cases for the structure of c , we will show that $\sigma = \sigma_1$ in each case.

$$\text{If } c \equiv \text{nop}, d = d_1 = \frac{}{[\text{nop}, \sigma_0] \rightarrow \sigma_0}$$

If $c \equiv I := a$, where I is any identifier and a is any expression,

$$d = \frac{\frac{\vdots}{[a, \sigma_0] \rightarrow m}}{[I := a, \sigma_0] \rightarrow [I \mapsto m] \sigma_0} \quad d_1 = \frac{\frac{\vdots}{[a, \sigma_0] \rightarrow m_1}}{[I := a, \sigma_0] \rightarrow [I \mapsto m_1] \sigma_0}$$

Since we have already shown that expressions are deterministic, $m = m_1$, and then $\sigma = \sigma_1$.

If $c \equiv c_0; c_1$,

$$d = \frac{\frac{\frac{\vdots}{[c_0, \sigma_0] \rightarrow \sigma'} \quad \frac{\vdots}{[c_1, \sigma'] \rightarrow \sigma}}{[c_0; c_1, \sigma_0] \rightarrow \sigma}}{d_1 = \frac{\frac{\frac{\vdots}{[c_0, \sigma_0] \rightarrow \sigma'_1} \quad \frac{\vdots}{[c_1, \sigma'_1] \rightarrow \sigma_1}}{[c_0; c_1, \sigma_0] \rightarrow \sigma_1}}$$

Let d^0 be the subderivation of $\frac{\vdots}{[c_0, \sigma_0] \rightarrow \sigma'}$, and d^1 be the subderivation of $\frac{\vdots}{[c_1, \sigma'] \rightarrow \sigma}$.

Since $d^0 \prec d$ and $d^1 \prec d$, from the assumption we can say that $P(d^0)$ and $P(d^1)$, and therefore from the form of P we can say that $\sigma' = \sigma'_1$ and $\sigma = \sigma_1$.

If $c \equiv \text{if } b \text{ then } c_0 \text{ else } c_1 \text{ end}$, we need to look at the cases where b is true and where it is false.

For the true case we have:

$$d = \frac{\frac{\vdots}{[b, \sigma_0] \rightarrow \text{true}} \quad \frac{\vdots}{[c_0, \sigma_0] \rightarrow \sigma}}{[\text{if } b \text{ then } c_0 \text{ else } c_1, \sigma_0] \rightarrow \sigma} \quad d_1 = \frac{\frac{\vdots}{[b, \sigma_0] \rightarrow \text{true}} \quad \frac{\vdots}{[c_0, \sigma_0] \rightarrow \sigma_1}}{[\text{if } b \text{ then } c_0 \text{ else } c_1, \sigma_0] \rightarrow \sigma_1}$$

Let d' be the subderivation of $[c_0, \sigma_0] \rightarrow \sigma$ in d . Then $d' \prec d$ and therefore $P(d')$, which means that $\sigma = \sigma_1$.

Finally, if $c \equiv \text{while } b \text{ do } c \text{ end}$ again we have two forms, for b true and for b false. For b false, we have

$$d = \frac{\frac{\vdots}{[b, \sigma_0] \rightarrow \text{false}}}{[\text{while } b \text{ do } c \text{ end}, \sigma_0] \rightarrow \sigma_0} \quad d_1 = \frac{\frac{\vdots}{[b, \sigma_0] \rightarrow \text{false}}}{[\text{while } b \text{ do } c \text{ end}, \sigma_0] \rightarrow \sigma_0}$$

So clearly $\sigma = \sigma_0 = \sigma_1$.

If b is true, we have

$$d = \frac{\frac{\frac{\vdots}{[b, \sigma_0] \rightarrow \text{true}} \quad \frac{\vdots}{[c, \sigma_0] \rightarrow \sigma'}}{\text{while } b \text{ do } c \text{ end}, \sigma'} \rightarrow \sigma}{[\text{while } b \text{ do } c \text{ end}, \sigma_0] \rightarrow \sigma}$$

$$d_1 = \frac{\frac{\frac{\vdots}{[b, \sigma_0] \rightarrow \text{true}} \quad \frac{\vdots}{[c, \sigma_0] \rightarrow \sigma_1}}{\text{while } b \text{ do } c \text{ end}, \sigma_1'} \rightarrow \sigma_1}{[\text{while } b \text{ do } c \text{ end}, \sigma_0] \rightarrow \sigma_1}$$

Let d' be the subderivation of $[c, \sigma_0] \rightarrow \sigma'$ and d'' be the subderivative of $[\text{while } b \text{ do } c \text{ end}, \sigma'] \rightarrow \sigma$ in d . Then $d' \prec d$ and $d'' \prec d$, so $P(d')$ and $P(d'')$. It follows from the form of P that $\sigma' = \sigma_1'$ and $\sigma = \sigma_1$.

We have shown in every case that $d \Vdash [c, \sigma_0] \rightarrow \sigma \wedge \Vdash [c, \sigma_0] \rightarrow \sigma_1$ implies $\sigma = \sigma_1$, and thus the property of determinism is proved.