

Rule Induction

COMP3610 – Principles of Programming Languages

Ben Lippmeier

Australian National University
Semester 2, 2009

General Induction

$$\forall n. (\forall k < n. P(k) \Rightarrow P(n)) \Rightarrow \forall m. P(m)$$

- Assume $P(k)$ is true for all previous things k
- With this information, prove that $P(n)$ is true for the next thing n .
- Therefore, $P(m)$ is true for all things m .
- No constraint is placed on what these “things” might be.
- All we need is an ordering relationship $<$.

Structural Induction Rules

- Induction over Natural Numbers.

$$\frac{P(0) \quad \forall n. P(n) \Rightarrow P(n+1)}{\forall n. P(n)}$$

- Induction over the structure of a list

$$\frac{P(\text{Nil}) \quad \forall x \text{ xs}. P(\text{xs}) \rightarrow P(\text{Cons } x \text{ xs})}{\forall \text{xs}. P(\text{xs})}$$

- Induction over the structure of a tree

$$\frac{P(\text{Null}) \quad \forall a \ t_1 \ t_2. P(t_1) \wedge P(t_2) \Rightarrow P(\text{Node } a \ t_1 \ t_2)}{\forall t. P(t)}$$

- Notice a pattern?

Ordering Relationship

For natural numbers (easy!)

$$\forall n. 0 < n$$

$$\forall n. n < n + 1$$

For lists:

$$\forall x \text{ xs}. \text{Nil} < \text{Cons } x \text{ xs}$$

$$\forall x \text{ xs}. \text{xs} < \text{Cons } x \text{ xs}$$

- Note that ordering on lists is defined in terms of “containment”.
- Example: $(\text{Cons } 1 (\text{Cons } 2 \text{ Nil}) < (\text{Cons } 3 (\text{Cons } 1 (\text{Cons } 2 \text{ Nil})))$
The larger list contains the smaller one.

Ordering Relationship

For trees:

$$\forall a t_1 t_2. t_1 < \text{Node } a t_1 t_2$$

$$\forall a t_1 t_2. t_2 < \text{Node } a t_1 t_2$$

$$\forall a t_1 t_2. \text{Null} < \text{Node } a t_1 t_2$$

The previous orderings define structural induction rules because the ordering is defined in terms of the structure of the data.

$$\frac{\langle a_0, \sigma \rangle \downarrow n_0 \quad \langle a_1, \sigma \rangle \downarrow n_1}{\langle a_0 + a_1, \sigma \rangle \downarrow n_0 + n_1}$$

$$\begin{aligned} \text{Plus} &:: \forall (\sigma : \text{State}) (a_0 : \text{AExp}) (a_1 : \text{AExp}) (n_0 : \text{Nat}) (n_1 : \text{Nat}) \\ &. \text{Proof}(\langle a_0, \sigma \rangle \downarrow n_0) \\ &\rightarrow \text{Proof}(\langle a_1, \sigma \rangle \downarrow n_1) \\ &\rightarrow \text{Proof}(\langle a_0 + a_1, \sigma \rangle \downarrow n_0 + n_1) \end{aligned}$$

Given proofs of $\langle a_0, \sigma \rangle \downarrow n_0$ and $\langle a_1, \sigma \rangle \downarrow n_1$ the constructor *Plus* produces a proof of $\langle a_0 + a_1, \sigma \rangle \downarrow n_0 + n_1$, for any values of $\sigma, a_0, a_1, n_0, n_1 \dots$

Compare:

$$\text{Node} :: \forall a. a \rightarrow \text{Tree } a \rightarrow \text{Tree } a \rightarrow \text{Tree } a$$

Represent proof rules as data

$$\frac{}{\langle n, \sigma \rangle \downarrow n} \text{num}$$

$$\text{Num} :: \forall (\sigma : \text{State}) (n : \text{Nat}). \text{Proof}(\langle n, \sigma \rangle \downarrow n)$$

$$\frac{}{\langle x, \sigma \rangle \downarrow \sigma(x)} \text{var}$$

$$\text{Var} :: \forall (\sigma : \text{State}) (x : \text{Var}). \text{Proof}(\langle x, \sigma \rangle \downarrow \sigma(x))$$

Num and *Var* are proof constructors (like data constructors). *Num* can produce a proof of the statement $\langle n, \sigma \rangle \downarrow n$ for all values of σ and n .

Note: I'm glossing over the details of how to handle variable binding.

Example Derivation

$$\frac{(1) \langle x, \sigma_2 \rangle \downarrow 2 \quad (2) \langle 3, \sigma_2 \rangle \downarrow 3}{(3) \langle x + 3, \sigma_2 \rangle \downarrow 5} \quad \sigma_2 = [x \mapsto 2]$$

$$t_1 = \text{Var } \sigma_2 \text{ "x"} \quad :: \text{Proof}(\langle x, \sigma_2 \rangle \downarrow 2)$$

$$t_2 = \text{Num } \sigma_2 3 \quad :: \text{Proof}(\langle 3, \sigma_2 \rangle \downarrow 3)$$

$$t_3 = \text{Plus } \sigma_2 \text{ "x"} 3 2 3 t_1 t_2 \quad :: \text{Proof}(\langle x + 3, \sigma_2 \rangle \downarrow 5)$$

or:

$$t_3 = \text{Plus } \sigma_2 \text{ "x"} 3 2 3 (\text{Var } \sigma_2 \text{ "x"}) (\text{Num } \sigma_2 3)$$

Induction over the structure of a binary tree

$$\frac{P(\text{Null}) \quad \forall a t_1 t_2. P(t_1) \wedge P(t_2) \Rightarrow P(\text{Node } a t_1 t_2)}{\forall t. P(t)}$$

Null :: $\forall a. \text{Tree } a$

Node :: $\forall a. a \rightarrow \text{Tree } a \rightarrow \text{Tree } a \rightarrow \text{Tree } a$

Note: I have elided some of the type information in the step case rules on the previous slides. The full statement for the first one would be:

$$\begin{aligned} \forall \quad & (\sigma : \text{State}) (a_0 : \text{AExp}) (a_1 : \text{AExp}) (n_0 : \text{Nat}) (n_1 : \text{Nat}) \\ & (p_1 : \text{Proof}(\langle a_0, \sigma \rangle \downarrow n_0)) \\ & (p_2 : \text{Proof}(\langle a_1, \sigma \rangle \downarrow n_1)) \\ \cdot \quad & P(p_1) \wedge P(p_2) \\ \Rightarrow \quad & P(\text{Plus } \sigma a_0 a_1 n_0 n_1 p_1 p_2) \end{aligned}$$

Of course, *Plus* doesn't accept *any* values p_1 and p_2 , only ones that satisfy the premises of its associated rule.

Induction over the structure of the proof tree for some property of arithmetic expressions

$$\frac{\begin{aligned} \forall \dots p_0 p_1. P(p_0) \wedge P(p_1) &\Rightarrow P(\text{Plus } \dots p_0 p_1) \\ \forall \sigma n. P(\text{Num } \sigma n) \quad \forall \dots p_0 p_1. P(p_0) \wedge P(p_1) &\Rightarrow P(\text{Minus } \dots p_0 p_1) \\ \forall \sigma n. P(\text{Var } \sigma n) \quad \forall \dots p_0 p_1. P(p_0) \wedge P(p_1) &\Rightarrow P(\text{Times } \dots p_0 p_1) \end{aligned}}{\forall p. P(p)}$$

Num :: $\forall \sigma n. \text{Proof}(\langle n, \sigma \rangle \downarrow n)$

Var :: $\forall \sigma x. \text{Proof}(\langle x, \sigma \rangle \downarrow \sigma(x))$

Plus :: $\forall \sigma a_0 a_1 n_0 n_1. \text{Proof}(\langle a_0, \sigma \rangle \downarrow n_0) \rightarrow \text{Proof}(\langle a_1, \sigma \rangle \downarrow n_1) \rightarrow \text{Proof}(\langle a_0 + a_1, \sigma \rangle \downarrow n_0 + n_1)$

...

Proof Technique

Base Cases

- For each axiom, prove the property holds for that axiom.

Step Cases

- For each rule with premises:
Assume the property holds for all the premises,
then prove the property holds for the conclusion.

What do we get?... Metatheory!

$$\forall p : Proof(< a, \sigma > \downarrow n). P(p)$$

We read P as being a property of a *whole proof tree*.

P is satisfied when its final statement matches the property of arithmetic expressions that we're trying to prove.

P is some property of a proof, and now we know how to prove it.

We're actually inducting over two proof trees at once

$$\forall (x : Com) (\sigma : State) (\sigma' : State) (\sigma'' : State).$$

if $< c, \sigma > \downarrow \sigma'$ and $< c, \sigma > \downarrow \sigma''$ then $\sigma' = \sigma''$

$$\begin{array}{c} T0 \\ \vdots \\ \hline < \mathbf{skip}, \sigma > \downarrow \sigma' \end{array} \qquad \begin{array}{c} T1 \\ \vdots \\ \hline < \mathbf{skip}, \sigma > \downarrow \sigma'' \end{array}$$

$$\begin{array}{cc} T0 & T1 \\ \vdots & \vdots \\ \hline < c_0; c_1, \sigma > \downarrow \sigma' \end{array} \qquad \begin{array}{cc} T2 & T3 \\ \vdots & \vdots \\ \hline < c_0; c_1, \sigma > \downarrow \sigma'' \end{array}$$

Proving Commands are Deterministic (from yesterday)

$$\forall (x : Com) (\sigma : State) (\sigma' : State) (\sigma'' : State).$$

if $< c, \sigma > \downarrow \sigma'$ and $< c, \sigma > \downarrow \sigma''$ then $\sigma' = \sigma''$

$$Skip \quad :: \forall \dots \quad Proof(< \mathbf{skip}, \sigma > \downarrow \sigma)$$

$$Asst \quad :: \forall \dots \quad Proof(< a, \sigma > \downarrow n) \\ \Rightarrow Proof(< x := a, \sigma > \downarrow \sigma[x \mapsto n])$$

$$Seq \quad :: \forall \dots \quad Proof(< c_0, \sigma > \downarrow \sigma') \\ \Rightarrow Proof(< c_1, \sigma' > \downarrow \sigma'') \\ \Rightarrow Proof(< c_0; c_1, \sigma > \downarrow \sigma'')$$

...

The conclusion of our induction rule should be:

$$\forall (c : Com) (\sigma : State) (\sigma' : State) \\ (p_1 : Proof(< c, \sigma > \downarrow \sigma')) \\ (p_2 : Proof(< c, \sigma > \downarrow \sigma'')) \\ \cdot P(p_1, p_2)$$

Base cases:

$$\forall (\sigma : State) (\sigma' : State). P(Skip \ \sigma, Skip \ \sigma')$$

...

Step cases:

\forall ...

$(p_0 : \text{Proof}(\langle a, \sigma_0 \rangle \downarrow \sigma'_0))$

$(p_1 : \text{Proof}(\langle a, \sigma_1 \rangle \downarrow \sigma'_1))$

$\cdot P(p_0, p_1) \Rightarrow P(\text{Asst} \dots p_0, \text{Asst} \dots p_1)$

\forall ...

$(p_0 : \text{Proof}(\langle c_0, \sigma_0 \rangle \downarrow \sigma'_0))$

$(p_1 : \text{Proof}(\langle c_1, \sigma'_0 \rangle \downarrow \sigma_1))$

$(p_2 : \text{Proof}(\langle c_0, \sigma_2 \rangle \downarrow \sigma'_2))$

$(p_3 : \text{Proof}(\langle c_1, \sigma'_2 \rangle \downarrow \sigma_3))$

$P(p_0, p_2) \wedge P(p_1, p_3) \Rightarrow P(\text{Seq} \dots p_0 p_1, \text{Seq} \dots p_2 p_3)$

...

Twelf

In 2-5 years all programming language theory courses will be based around theorem provers/checkers. Doing proofs by hand is a losing battle.

`of : exp -> tp -> type. %name of Dof.`

`%mode of +E -T.`

```
of/app : of (app E1 E2) T
         <- of E1 (arr T2 T)
         <- of E2 T2.
```

```
of/fun : of (fun T1 T2 ([f] [x] E f x)) (arr T1 T2)
         <- ({f:exp} of f (arr T1 T2)
            -> {x:exp} of x T1 -> of (E f x) T2).
```

Programmers are not to be measured by their ingenuity and their logic but by the completeness of their case analysis.

– Alan Perlis