Lecture 5

Pete Manolios
Northeastern

Induction & Deduction

Aristotle made the distinction between deduction and induction. He described induction as an "argument from the particular to the universal" and as the mechanism by which we discover the indemonstrable first principles of the sciences.

Induction is the process of generalizing from our known and limited experience, and framing wider rules for the future than we have been able to test fully.

Jacob Bronowski

Inductive reasoning is, of course, good guessing, not sound reasoning, but the finest results in science have been obtained in this way. Calling the guess a "working hypothesis," its consequences are tested by experiment in every conceivable way.

Joseph William Mellor

Science, in its ultimate ideal, consists of a set of propositions arranged in a hierarchy, the lowest level of the hierarchy being concerned with particular facts, and the highest with some general law, governing everything in the universe. The various levels in the hierarchy have a two-fold logical connection, travelling one up, one down; the upward connection proceeds by induction, the downward by deduction.

Bertrand Russell

Slides by Pete Manolios for CS4820

Mathematical Induction

Mathematical Induction is a deductive form of reasoning: anything we derive using mathematical induction *must* be true.

It is sometimes thought of as being almost magical.

If we have no idea why a statement is true, we can still prove it by induction.

Gian-Carlo Rota

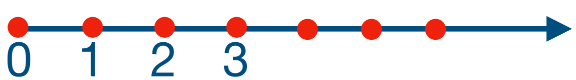
Induction Examples

Induction on Natural Numbers

$P(0)$

$\underline{[n > 0 \wedge P(n-1) \Rightarrow P(n)]}$

$[P(n)]$

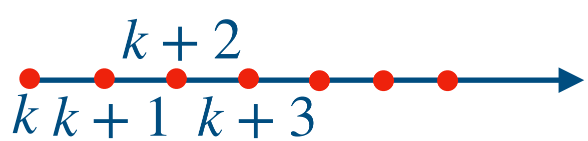


Induction integers $\geq k$

$P(k)$

$\underline{[n > k \wedge P(n-1) \Rightarrow P(n)]}$

$[P(n)]$



Slides by Pete Manolios for CS4820

Induction Examples

Induction on Integers (Generalize Nats)

$$P(0)$$

$$[n > 0 \land P(n-1) \Rightarrow P(n)]$$

$$\underline{[n < 0 \land P(n+1) \Rightarrow P(n)]}$$

$$[P(n)]$$

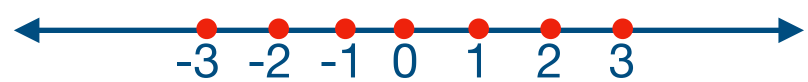
Works for Rationals?

$$P(0)$$

$$[n > 0 \land P(n-1) \Rightarrow P(n)]$$

$$\underline{[n < 0 \land P(n+1) \Rightarrow P(n)]}$$

$$[P(n)]$$



Induction on Rationals

$$[0 \leq n < 1 \Rightarrow P(n)]$$

$$[n \geq 1 \land P(n-1) \Rightarrow P(n)]$$

$$\underline{[n < 0 \land P(n+1) \Rightarrow P(n)]}$$

$$[P(n)]$$

Strong Induction

Induction on Natural Numbers

$$\frac{[\langle \forall k < n :: P(k) \rangle \Rightarrow P(n)]}{[P(n)]}$$

Induction on Integers

$$\frac{\begin{array}{l}\langle \forall n \geq 0 :: \langle \forall k : 0 \leq k < n : P(k) \rangle \Rightarrow P(n) \rangle \\ \langle \forall n < 0 :: \langle \forall k > n :: P(k) \rangle \Rightarrow P(n) \rangle\end{array}}{[P(n)]}$$

Most powerful kind of induction?

Well-Founded Induction: $\langle W, \prec \rangle$ is *well founded* iff $\prec$ is terminating (there are no infinite $\prec$-decreasing sequences; $\prec$ is a relation on W)

$$\frac{\langle \forall y \in W :: \langle \forall x \in W : x \prec y : P(x) \rangle \;\Rightarrow\; P(y) \rangle}{\langle \forall y \in W :: P(y) \rangle}$$

Exercise: Show that all the induction principles from this lecture are special cases of well-founded induction.

ACL2s Induction Schemes

Key Idea:

We already prove termination for functions

So, the relations they give rise to are well-founded!

So, we can induct using schemes derived from function definitions

```
(definec nat-ind (n :nat) :nat
  (if (zp n)
      0
    (nat-ind (1- n))))
```

Induction on natural numbers

```
(definec tree-ind (x :all) :all
  (if (atom x)
      x
    (list (tree-ind (car x))
          (tree-ind (cdr x)))))
```

Induction on trees

```
(definec tlp (l :all) :bool
  (if (consp l)
      (tlp (rest l))
    (equal l () )))
```

Induction on true lists

What is decreasing?

The measure.

ACL2s Induction Schemes

```
(defdata
  (saexpr (or rational
              var
              usaexpr
              bsaexpr))
  (usaexpr ...)
  (bsaexpr ...))
```

Defdata Induction

$$\frac{P(\text{(rationalp e)}) \quad P(\text{(varp e)}) \quad P(\text{(usaexprp e)}) \quad P(\text{(bsaexprp e)})}{[P(\text{e})]}$$

Common themes:

Induction on data definitions

Induction on functions in conjectures

Custom inductions

Can direct ACL2s to use specific induction schemes

```
(definec saeval (e :saexpr a :assignment) :rat-err
  (match e
    (:rational ...)
    (:var ...)
    (:usaexpr
     (('- x) ...)
     ...)
    ...))
```

Definition Induction

$$\frac{P(\text{(rationalp e)}) \quad P(\text{(varp e)}) \quad P(\text{(usaexprp e), e = ('- x)}) \quad P(\text{(usaexprp e), e = ...}) \quad P(\text{(bsaexprp e), e = ...}) \quad \ldots}{[P(\text{e})]}$$

Slides by Pete Manolios for CS4820

Induction in ACL2s

- Given a function definition of the form `(definec f (x1 :d1 … xn :dn) :df  :ic ic :oc oc  body)`
- Expand all macros
- *Terminal:* expression occurrence in body with no `if`'s in it & which is not a subexpression of the test of any `if`
- A terminal is *maximal* if it is not contained in any other terminal
- For every terminal, there is a corresponding *condition* that must hold for execution to reach the terminal
- The set of *recursive calls* of a terminal t contains all calls to `f` that must be executed in order to execute t
- If the set of recursive calls of a terminal is empty, then the terminal is *basic*; otherwise it is *recursive*
- Let $\langle t_1, \ldots, t_m \rangle$ be a sequence containing `f`'s maximal terminals with corresponding conditions $\langle c_1, \ldots, c_m \rangle$
- with recursive calls $\langle r_1, \ldots, r_m \rangle$, where r_i is $\{$`(f x1 ... xn)`$|_{\sigma_i^j} : 1 \leq j \leq |r_i|\}$ (the σ_i^j's are substitutions; r_i is {} if t_i is basic)

```
(not (or (g x) (or (not (f (1- x))) (f (- x 2)))))
```

= {expand macros}

```
(not (if (g x) (g x) (if (not (f (- x 1))) (not (f (- x 1))) (f (- x 2)))))
```

Recursive Calls for Maximal Terminals: `{}`, `{(f (- x 1))}`, `{{(f (- x 1)), (f (- x 2))}`

$\sigma_2^1 =$ `((x (- x 1)))`, $\sigma_3^1 =$ `((x (- x 1))`, $\sigma_3^2 =$ `((x (- x 2))`

Maximal Terminals:

Conditions:

Slides by Pete Manolios for CS4820

Induction in ACL2s

- Given a function definition of the form `(definec f (x1 :d1 … xn :dn) :df  :ic ic :oc oc  body)`
- Expand all macros
- *Terminal:* expression occurrence in body with no `if`'s's in it & which is not a subexpression of the test of any `if`
- A terminal is *maximal* if it is not contained in any other terminal
- For every terminal, there is a corresponding *condition* that must hold for execution to reach the terminal
- The set of *recursive calls* of a terminal t contains all calls to `f` that must be executed in order to execute t
- If the set of recursive calls of a terminal is empty, then the terminal is *basic*; otherwise it is *recursive*
- Let $\langle t_1, \ldots, t_m \rangle$ be a sequence containing `f`'s maximal terminals with corresponding conditions $\langle c_1, \ldots, c_m \rangle$
- with recursive calls $\langle r_1, \ldots, r_m \rangle$, where r_i is $\{\texttt{(f x1 ... xn)}|_{\sigma_i^j} : 1 \leq j \leq |r_i|\}$ (the σ_i^j's are substitutions; r_i is {} if t_i is basic)
- The function `f` gives rise to an *induction scheme* that is parameterized by a formula ϕ
- To prove ϕ, you can instead prove, where $C = \texttt{ic} \wedge \bigwedge_{1 \leq i \leq n} \texttt{(d}i\ \texttt{x}i\texttt{)}$:

    1. $\neg C \Rightarrow \phi$
    2. For all t_i that are basic terminals: $C \wedge c_i \Rightarrow \phi$
    3. For all t_i that are recursive terminals: $C \wedge c_i \wedge \bigwedge_{1 \leq j \leq |r_i|} \phi|_{\sigma_i^j} \Rightarrow \phi$

Professional Method

```
(definec ap (a :tl b :tl) :tl
  (if (endp a)
      b
    (cons (car a)
          (ap (cdr a) b))))
```

```
(definec rv (x :tl) :tl
  (if (endp x)
      nil
    (ap (rv (cdr x))
        (list (car x)))))
```

Prove: (rv (rv x)) = x No quite right, why?

Prove: (tlp x) ⇒ (rv (rv x)) = x Contract completion!

Professional Method: use abbreviations, discover induction scheme

We'll induct on (... x). Base case is trivial, so go to induction step

```
          (R (R x))
= {Def R} (R (A (R (cdr x)) (L (car x))))
= {L1}    (A (R (L (car x))) (R (R (cdr x))))
= {IH}    (A (R (L (car x))) (cdr x))
= {Def R} (A (L (car x)) (cdr x))
= {Def A} x
```

Hm, to use IH, need lemma

Now I can use IH

Just equational reasoning

L1.(R (A x y)) = (A (R y) (R x))

What Induction scheme?

(tlp x) or (rev x): minor differences

Professional Method

```
(definec ap (a :tl b :tl) :tl
  (if (endp a)
      b
    (cons (car a)
          (ap (cdr a) b))))
```

```
(definec rv (x :tl) :tl
  (if (endp x)
      nil
    (ap (rv (cdr x))
        (list (car x)))))
```

Prove: (tlp x) ∧ (tlp y) ⇒ (R (A x y)) = (A (R y) (R x))

Professional Method: induct on?   x controls both LHS, RHS, so probably x

Start with induction step

```
          (R (A x y))
= {Def A} (R (cons (car x) (A (cdr x) y)))
= {Def R} (A (R (A (cdr x) y)) (L (car x)))
= {IH}    (A (A (R y) (R (cdr x))) (L (car x)))
= {Ass A} (A (R y) (A (R (cdr x)) (L (car x))))
= {Def R} (A (R y) (R x))
```

Ass A: (A (A x y) z) = (A x (A y z))
What Induction scheme?
(tlp x) or (rev x): minor differences

Base case?

```
          (R (A x y))
= {Def A} (R y)

          (A (R y) (R x))
= {Def R} (A (R y) nil)
= {L2!}   (R y)
```

L2: (A x nil) = x
Needs proof by induction!

DEMO

ACL2 is . . .

- **A programming language:**
  - Applicative, functional subset of Lisp
  - Compilable and executable
  - Untyped, first-order
- **A mathematical logic:**
  - First-order predicate calculus
  - With equality, induction, recursive definitions
  - Ordinals up to ϵ_0 (termination & induction)
- ***A mechanical theorem prover:***
  - *Integrated system of ad hoc proof techniques*
  - *Heavy use of term rewriting*
  - *Largely written in ACL2*

Next Time

Questions?