Induction and Recursion

Prof. Clarkson
Fall 2016

Today’s music: *Dream within a Dream*
from the soundtrack to *Inception* by Hans Zimmer
Review

Previously in 3110:

• Behavioral equivalence
• Proofs of correctness by induction on naturals

Today:

• Induction on lists
• Induction on trees
Review: Induction on natural numbers

Theorem:
for all natural numbers \( n \), \( P(n) \).

Proof: by induction on \( n \)

Case: \( n = 0 \)
Show: \( P(0) \)

Case: \( n = k+1 \)
IH: \( P(k) \)
Show: \( P(k+1) \)

QED
Induction principle

for all properties $P$ of natural numbers, if $P(0)$ and (for all $n$,
  $P(n)$ implies $P(n+1)$) then (for all $n$, $P(n)$)
Induction principle

for all properties $P$ of lists,
if $P([])$
and (for all $x$ and $xs$,
$P(xs)$ implies $P(x::xs)$)
then (for all $xs$, $P(xs)$)
Induction on lists

Theorem:
for all lists \( \text{lst} \), \( P(\text{lst}) \).

Proof: by induction on \( \text{lst} \)

Case: \( \text{lst} = [] \)
Show: \( P([]) \)

Case: \( \text{lst} = h::t \)
IH: \( P(t) \)
Show: \( P(h::t) \)

QED
Append

let rec length = function
    | [] -> 0
    | _::xs -> 1 + length xs

let rec append xs1 xs2 = match xs1 with
    | [] -> xs2
    | h::t -> h :: append t xs2

Theorem.
for all lists xs and ys,
    length (append xs ys) ~ length xs + length ys.
Append

Theorem.
for all lists \(xs\) and \(ys\),

\[
\text{length (append } xs \ ys \text{)} \sim \text{length } xs + \text{length } ys.
\]

Proof: by induction on \(xs\)

Case: \(xs = []\)
Show: for all \(ys\),

\[
\text{length (append } [] \ ys \text{)} \sim \text{length } [] + \text{length } ys
\]

\[
\text{length (append } [] \ ys \text{)} \sim \text{length } ys \quad \text{(eval)}
\]
\[
\sim 0 + \text{length } ys \quad \text{(math)}
\]
\[
\sim \text{length } [] + \text{length } ys \quad \text{(eval,symm.)}
\]
Append

Theorem.
for all lists xs and ys,
length (append xs ys) ~ length xs + length ys.

Proof: by induction on xs

Case: xs = h::t
Show: for all ys, length (append (h::t) ys)
      ~ length (h::t) + length ys
IH:   ??
If we're trying to prove
for all lists \(xs\) and \(ys\),

\[
\text{length } (\text{append } xs \ ye) \sim \text{length } xs + \text{length } ys.
\]

by induction on \(xs\), in the case where \(xs = h::t\), what is the inductive hypothesis?

A. for all \(ys\),

\[
\text{length } (\text{append } xs \ ye) \sim \text{length } xs + \text{length } ys
\]

B. for all \(ys\),

\[
\text{length } (\text{append } t \ ye) \sim \text{length } t + \text{length } ys
\]

C. for all \(ys\),

\[
\text{length } (\text{append } (h::t) \ ye) \\
~ \text{length } (h::t) + \text{length } ys
\]

D. for all \(h'\) and \(t'\),

\[
\text{length } (\text{append } (h::t) (h'::t')) \\
~ \text{length } (h::t) + \text{length } (h'::t')
\]

E. for all \(xs\),

\[
\text{length } (\text{append } xs \ t) \sim \text{length } xs + \text{length } t
\]
Question

If we're trying to prove
for all lists xs and ys,
  \[ \text{length} \left( \text{append} \; \text{xs} \; \text{ys} \right) \sim \text{length} \; \text{xs} + \text{length} \; \text{ys} \].
by induction on xs, in the case where \( \text{xs} = h::t \), what is the inductive hypothesis?

A. for all ys,
  \[ \text{length} \left( \text{append} \; \text{xs} \; \text{ys} \right) \sim \text{length} \; \text{xs} + \text{length} \; \text{ys} \]

B. for all ys,
  \[ \text{length} \left( \text{append} \; t \; \text{ys} \right) \sim \text{length} \; t + \text{length} \; \text{ys} \]

C. for all ys,
  \[ \text{length} \left( \text{append} \; (h::t) \; \text{ys} \right) \sim \text{length} \; (h::t) + \text{length} \; \text{ys} \]

D. for all \( h' \) and \( t' \),
  \[ \text{length} \left( \text{append} \; (h::t) \; (h':t') \right) \sim \text{length} \; (h::t) + \text{length} \; (h':t') \]

E. for all xs,
  \[ \text{length} \left( \text{append} \; \text{xs} \; t \right) \sim \text{length} \; \text{xs} + \text{length} \; t \]
Append

Theorem.
for all lists \(xs\) and \(ys\),

\[
\text{length (append } xs\ \text{ ys)} \sim \text{length } xs + \text{length } ys.
\]

Proof: by induction on \(xs\)

Case: \(xs = h::t\)
Show: for all \(ys\), length (append (h::t) \(ys\))

\[
\sim \text{length (h::t) + length } ys
\]

IH: for all \(ys\), length (append \(t\) \(ys\))

\[
\sim \text{length } t + \text{length } ys
\]
Append

Case: \(xs\) is \(h::t\)
Show: for all \(ys\), \(\text{length} (\text{append} (h::t) \ ys) \sim \text{length} (h::t) + \text{length} \ ys\)
IH: for all \(ys\), \(\text{length} (\text{append} \ t \ ys) \sim \text{length} \ t + \text{length} \ ys\)

\[
\text{length} (\text{append} (h::t) \ ys) \\
\sim \text{length} (h :: \text{append} \ t \ ys) \quad \text{(eval)} \\
\sim 1 + \text{length} (\text{append} \ t \ ys) \quad \text{(eval)} \\
\sim 1 + \text{length} \ t + \text{length} \ ys \quad \text{(IH, congr.)} \\
\sim \text{length} (h::t) + \text{length} \ ys \quad \text{(eval, symm., congr.)}
\]
QED

From now on, omit many uses of symm., trans., congr.
Higher-order functions

Proofs about higher-order functions sometimes need an additional axiom:

**Extensionality:**

if (for all \( x \), \((f \ x) \sim (g \ x)\))

then \( f \sim g \)
**Compose**

```ocaml
let (@@) f g x = f (g x)
let map = List.map
```

**Theorem:**
for all functions f and g,

\[(\text{map } f) \@\!\@ \ (\text{map } g) \sim \text{map } (f \@\!\@ g).\]

**Proof:**
By extensionality, we need to show that for all \(xs\),

\[(\text{map } f) \@\!\@ (\text{map } g) \; \text{xs} \sim \text{map } (f \@\!\@ g) \; \text{xs}.\]

By eval, \[(\text{map } f) \@\!\@ (\text{map } g) \; \text{xs} \sim \text{map } f \; (\text{map } g \; \text{xs}).\]
So by transitivity, it suffices to show that

\[\text{map } f \; (\text{map } g \; \text{xs}) \sim \text{map } (f \@\!\@ g) \; \text{xs}.\]
Compose

Show: map f (map g xs) ~ map (f @@ g) xs.

Proof: by induction on xs

Case: xs = []
Show: map f (map g []) ~ map (f @@ g) []

    map f (map g [])
~ []                    (eval)
~ map (f @@ g) []      (eval)
Compose

Show: \( \text{map } f (\text{map } g \; x) \sim \text{map } (f \circ g) \; x. \)

Proof: by induction on \( x \)

Case: \( x = h::t \)

Show: \( \text{map } f (\text{map } g \; (h::t)) \sim \text{map } (f \circ g) \; (h::t) \)

IH: \( \text{map } f (\text{map } g \; t) \sim \text{map } (f \circ g) \; t \)

\[
\begin{align*}
\text{map } f (\text{map } g \; (h::t)) & \sim \text{map } f ((g \; h)::\text{map } g \; t) \quad \text{(eval \; map)} \\
& \sim (f \; (g \; h))::\text{map } f (\text{map } g \; t) \quad \text{(eval \; map)} \\
& \sim ((f \circ g) \; h)::\text{map } f (\text{map } g \; t) \quad \text{(eval \; \circ)} \\
& \sim ((f \circ g) \; h)::\text{map } (f \circ g) \; t \quad \text{(IH)} \\
& \sim \text{map } (f \circ g) \; (h::t) \quad \text{(eval \; map)}
\end{align*}
\]
## Compose

\[
\text{let } (\@\@) \ f \ g \ x = f (g \ x) \\
\text{let } \text{map} = \text{List.map}
\]

**Theorem:**
for all functions \( f \) and \( g \),
\[
(\text{map } f) \@\@ (\text{map } g) \sim \text{map } (f \@\@ g).
\]

**Proof:**
By extensionality, we need to show that for all \( xs \),
\[
((\text{map } f) \@\@ (\text{map } g)) \ xs \sim \text{map } (f \@\@ g) \ xs.
\]
By eval, \((\text{map } f) \@\@ (\text{map } g)) \ xs \sim \text{map } f (\text{map } g \ xs)\).
So by transitivity, it suffices to show that
\[
\text{map } f (\text{map } g \ xs) \sim \text{map } (f \@\@ g) \ xs. \quad \text{We have.}
\]
QED.
Compos

let ( @@ ) f g x = f (g x)
let map = List.map

Theorem:
for all functions f and g,
   (map f) @@ (map g) ~ map (f @@ g).

Comment: this theorem would be the basis for a nice compiler optimization in a pure language. Replace an operation that processes list twice with an operation that processes list only once.
Trees

type 'a tree =
  | Leaf
  | Branch of 'a * 'a tree * 'a tree

let rec reflect = function
  | Leaf -> Leaf
  | Branch(x,l,r) -> Branch(x, reflect r, reflect l)
Trees

reflection of

```
    1
   / \ 
  2   3
 / \ / \ 
4   5   6   7
```
is

```
    1
   / \ 
  3   2
 / \ / \ 
7   6 5 4
```
Trees

```ocaml
type 'a tree =
  | Leaf
  | Branch of 'a * 'a tree * 'a tree

let rec reflect = function
  | Leaf -> Leaf
  | Branch(x, l, r) -> Branch(x, reflect r, reflect l)
```

Theorem: for all trees t, reflect(reflect t) ~ t.

Proof: by induction on t.
Induction principle

for all properties P of trees, if P(Leaf)
and (for all x and l and r,
    P(l) and P(r) implies P(Branch(x,l,r))
then (for all t, P(t))
Induction on trees

Theorem:  
for all trees $t$, $P(t)$.

Proof: by induction on $t$

Case:  $n = \text{Leaf}$  
Show: $P(\text{Leaf})$

Case:  $n = \text{Branch}(x,l,r)$  
IH: $P(l)$ and $P(r)$  
Show: $P(\text{Branch}(x,l,r))$

QED
Trees

Theorem: for all trees \( t \), \( \text{reflect} (\text{reflect} \ t) \sim t \).

Proof: by induction on \( t \).

Case: \( t = \text{Leaf} \)
Show: \( \text{reflect} (\text{reflect} \ \text{Leaf}) \sim \text{Leaf} \)

\[
\text{reflect} (\text{reflect} \ \text{Leaf})
\sim \text{Leaf} \quad \text{(eval)}
\]
Trees

Theorem: for all trees $t$, $\text{reflect}(\text{reflect } t) \sim t$.

Proof: by induction on $t$.

Case: $t = \text{Branch}(x, l, r)$
Show:

$\text{reflect}(\text{reflect}(\text{Branch}(x, l, r))) \sim \text{Branch}(x, l, r)$

IH: ???
Question

How many formulas in inductive hypothesis—i.e., how many inductive hypotheses?

A. 1 (for the Branch constructor)
B. 2 (for the two subtrees)
C. 3 (for the two subtrees and the node's label)
Question

How many formulas in inductive hypothesis—i.e., how many inductive hypotheses?

A. 1 (for the Branch constructor)

B. 2 (for the two subtrees)

C. 3 (for the two subtrees and the node's label)
Trees

Theorem: for all trees t, reflect(reflect t) ~ t.

Proof: by induction on t.

Case: t = Branch(x,l,r)
Show:
    reflect(reflect(Branch(x,l,r))) ~ Branch(x,l,r)
IH:
    1. reflect(reflect l) ~ l
    2. reflect(reflect r) ~ r
Trees

Show:

\[ \text{reflect(\text{reflect(\text{Branch}(x,l,r)))} \sim \text{Branch}(x,l,r)} \]

IH:

1. \[ \text{reflect(\text{reflect } l)} \sim l \]
2. \[ \text{reflect(\text{reflect } r)} \sim r \]

\[
\begin{align*}
\text{reflect(\text{reflect(\text{Branch}(x,l,r)))} & \sim \text{reflect(\text{Branch}(x, \text{reflect } r, \text{reflect } l))} & \text{(eval)} \\
& \sim \text{Branch}(x, \text{reflect(\text{reflect } l}), \text{reflect(\text{reflect } r)}) & \text{(eval)} \\
& \sim \text{Branch}(x, l, \text{reflect(\text{reflect } r)}) & \text{(IH 1)} \\
& \sim \text{Branch}(x, l, r) & \text{(IH 2)}
\end{align*}
\]

QED
Inductive proofs on variants

type \( t = C_1 \text{ of } t_1 \mid \ldots \mid C_n \text{ of } t_n \)

Theorem: for all \( x : t \), \( P(x) \)
Proof: by induction on \( x \)

... 

Case: \( x = C_i y \)
IH: \( P(v) \) for any components \( v : t \) of \( y \)
Show: \( P(C_i y) \)

... 

QED
General induction principle

for all properties P of t,
    if
        (for all Ci,
            (for all y,
                (for all components z:t of y, P(z))
                implies P(Ci y)
            )
        )
    then
        (for all t, P(t))
Naturals

(* unary representation *)

\texttt{type nat = Z | S of nat}

Theorem: 
for all \( n : \text{nat} \), \( P(n) \)
Proof: by induction on \( n \)

Case: \( n = Z \)
Show: \( P(Z) \)

Case: \( n = S \ k \)
IH: \( P(k) \)
Show: \( P(S \ k) \)

QED
Induction

- The kind of induction we've done today is called structural induction
  - Induct on the structure of a data type
  - Widely used in programming languages theory
- When naturals are coded up as variants, weak induction becomes structural induction
- Both structural induction and weak induction (and strong induction) are instances of a very general kind of induction called well-founded induction
Induction and recursion

• Intense similarity between inductive proofs and recursive functions on variants
  – In proofs: one case per constructor
  – In functions: one pattern-matching branch per constructor
  – In proofs: uses IH on "smaller" value
  – In functions: uses recursive call on "smaller" value

• Inductive proofs truly are a kind of recursive programming (see Curry-Howard isomorphism)
Upcoming events

• [next Wednesday] MS1 due

This is inductive.

THIS IS 3110