





**COMPUTER SCIENCE 51**  
 Spring 2009  
 cs51.seas.harvard.edu

**Prof. Greg Morrisett**  
**Prof. Ramin Zabih**

computer science 51  



## Induction

- Computers are discrete & finite
  - But we don't want to argue based on their current hardware
  - Want our programs to run correctly and efficiently on arbitrary inputs
- Prove an infinite set of statements?
  - P[1] = "my factorial program is correct when the input is 1"
  - P[2], P[3], ...
    - Gets old fast!

## Induction recipe

1. What variable  $n$  are you doing induction on, and what is its type?
2. What statement  $P[n]$  will you prove?
3. Prove  $P[1]$  (first element, can be 0)
4. For an **arbitrary**  $n$ , prove  $P[n+1]$  follows from  $P[n]$ .
  - *Induction Hypothesis* (must be used!)



 

## Recipe use (math)

1. Variable is positive integer  $n$
2. Statement:  $\forall n \in \{1, 2, \dots\} 2^n > \sum_{j=0}^{n-1} 2^j$
3. Proof of  $P[1]$   $2^1 > \sum_{j=0}^{1-1} 2^j = 2^0$  😊
4. Want to show (via IH!):
 
$$2^{n+1} > \sum_{j=0}^{n+1-1} 2^j = 2^n + \sum_{j=0}^{n-1} 2^j$$



$$2^{n+1} = 2^n + 2^n > 2^n + \sum_{j=0}^{n-1} 2^j$$

$$2^n > \sum_{j=0}^{n-1} 2^j$$
 😊



## English ambiguity in IH

- Need: for all  $n$ ,  $P[n]$  implies  $P[n+1]$ 
  - We pick an  $n$  and assume  $P[n]$  (IH)
  - Then derive  $P[n+1]$
- It's always true that
  - For all  $n$ ,  $P[n]$  *implies*
  - For all  $n$ ,  $P[n+1]$
- Math notation helps
  - $\forall n ( P[n] \Rightarrow P[n+1] )$  versus
  - $\forall n ( P[n] ) \Rightarrow \forall n ( P[n+1] )$

## Example notes

- IH versus statement
- This theorem is very easy in binary
- Related to many exponentially growing functions
  - "Everything is at the leaves"
- Explains iterative deepening
  - The repeated work doesn't matter much

## Recipe use (Scheme)

- Start with simple definition
  - Note: more complex definitions make proofs much harder (bad on exams!)

```
(define (fact n)
  (if (= n 1)
      1
      (* n (fact (- n 1)))))
```

## Recipe on fact

- 1. Variable is  $n$ , the input, whose type is a positive integer
  - Can have multiple arguments
  - Note the type gives us a smallest element
    - Usually 0 or 1
- 2.  $P[n]$  = "the value of (fact  $n$ ) is  $n!$ "
  - Mathematical statement about a program, under the evaluation rules ("semantics") of Scheme

## Recipe on fact (cont.)

- 3. Prove  $P[1]$  = "value of (fact 1) is 1!"
  - By the evaluation rules, (fact 1) is the body with the variable  $n$  replaced by 1
 

```
(if (= 1 1)
      1
      (* 1 (fact (- 1 1))))
```
  - The predicate evaluates to #t, and so the value of (fact 1) is 1, which is 1!

## Recipe on fact (cont.)

- For arbitrary  $n$ , prove  $P[n+1]$  from  $P[n]$ 
  - Prove "value of (fact  $n+1$ ) is  $n+1!$ " follows from "value of (fact  $n$ ) is  $n!$ "
  - By the evaluation rules, (fact  $n+1$ ) is
 

```
(if (= n+1 1)
      1
      (* n+1 (fact (- n+1 1))))
```
  - Predicate is #f
    - Why is this true?

## End of proof

- Need to show  $n+1!$  is value of
 

```
(* n+1 (fact (- n+1 1)))
```
- By evaluation rules this is
 

```
(* n+1 (fact n))
```
- Use induction hypothesis!
  - $P[n]$  is "value of (fact  $n$ ) is  $n!$ "
- So this is
 

```
(* n+1 n!)
```

## How to lose points

- Don't follow the recipe
  - If we can't find all the parts clearly labeled, you aren't entitled to 100%
  - The first two parts are actually important, especially on harder examples
- Don't use the IH
  - No IH  $\Rightarrow$  No induction  $\Rightarrow$  No points
- Don't use the evaluation rule
  - No evaluation  $\Rightarrow$  No Scheme  $\Rightarrow$  No points

## An exam question

- Write the recursive factorial in Scheme
  - Or maybe something a bit harder...
- Prove it is correct using induction and the evaluation rules
- Write the recurrence relation for the running time, and solve it
  - $T(n) = c + T(n-1)$ , which is  $O(n)$
  - Be careful: not all Scheme primitives take constant time!

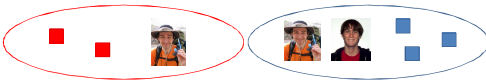
## Another example

- More interesting example
 

```
(define (times a b)
  (if (= b 0)
      0
      (+ a (times a (- b 1)))))
```
- What are we doing induction on, and what statement are we proving?

## Beer induction problem

- All TF's like beer equally
  - By induction on the number of TF's
- $P[n]$  = "Any group of TF's of size  $n$  likes beer equally"
- $P[1]$  is clearly true
- To prove  $P[n+1]$ , divide the set into two smaller overlapping groups



## Induction variant

- Consider a recursively-defined datastructure, like a list
  - The empty list, or
  - $\text{CONS}(x,L)$  where  $L$  is a list
- Let's think of the dullest function:

```
(define (copy lst)
  (if (empty? lst)
      lst
      (cons (first lst)
            (copy (rest lst)))))
```

## Correctness proof

- Can do proof by induction
  - On what?
- But this is sort of un-natural, since the induction is really on the list
  - And just like the natural numbers, lists are recursively defined
    - Base element (0, or empty-list)
    - Successor function (1+, or cons)
  - "Inductively defined set"

## Structural induction recipe

- In lecture, will do lists
- 1. Variable is a list, call it  $lst$
- 2.  $P[lst]$  is "(copy  $lst$ ) is the same as  $lst$ "
- 3. Prove  $P[\text{empty-list}]$
- 4. Prove  $P[(\text{cons } x \text{ } lst)]$ , for any  $x, lst$ 
  - Using  $P[lst]$

## More interesting example

- Prove correctness of:
 

```
(define (member x lst)
  (cond [(empty? lst) #f]
        [(= (first lst) x) #t]
        [else (member x (rest lst))]))
```
- What is the statement we are proving?
- Can do this either by regular induction or by structural induction



## Induction on graphs

- A clique in an undirected graph with  $n$  nodes has  $n(n-1)/2$  edges.
- Induction on the number of nodes
- Induction step is a counting argument, much as in the math example
- As in the math example, can get to this result directly as well!



## Strong induction

- IH is  $P[n], P[n-1], P[n-2]$ , etc.
- Great for structural induction!
- A tree with  $n$  vertices has  $n-1$  edges
- Removing an edge creates two subtrees, both of which have fewer vertices

