15-150 Fall 2013

Stephen Brookes

Lecture 4

Using induction

Our plan

- Introduce induction
  - templates to help write accurately
  - learn when applicable
- Focus on *examples*
- Specifications will involve *equality* and *evaluation*

Simple induction

- To prove a property of the form

  P(n), for all non-negative integers n

- First, prove P(0). *base case*
- Then show that, for all $k \geq 0$, P(k+1) follows logically from P(k).

inductive step

Example

```
fun f(x:int):int =
      if x=0 then 1 else f(x-1) + 1
(* REQUIRES x≥0        *)
(* ENSURES f(x) = x+1 *)
```

- To prove:

For all values x:int
such that $x \geq 0$, $f(x) = x+1$

Proof by simple induction

- Let P(n) be f(n) = n+1
- Base case: we prove P(0), i.e. f(0) = 0+1

f 0 = (fn x => if x=0 then 1 else f(x-1)+1) 0
= ⟦x:0⟧(if x=0 then 1 else f(x-1)+1)
= **if** 0=0 **then** 1 **else** f(0-1) + 1
= **if true then** 1 **else** f(0-1) + 1
= 1

0+1 = 1

So f(0) = 0+1

Proof by simple induction

- Let P(n) be f(n) = n+1
- Inductive step:
  let k≥0, assume P(k), prove P(k+1).
  Let v be the value of k+1.

f(k+1) = **if** v=0 **then** 1 **else** f(v-1) + 1
= **if false then** 1 **else** f(v-1) + 1
= f(v-1) + 1
= f(k) + 1 since v=k+1
= (k + 1) + 1 by assumption P(k)

So P(k+1) follows from P(k)

Using simple induction

- Q: When can I use *simple* induction to prove a property of a recursive function f ?
- A: When there is a *non-negative* measure of *argument size* and f(x) only makes recursive calls of form f(y) with *size*(y) = *size*(x)-1

Example

```
fun eval ([ ]:int list) : int = 0
 |    eval (d::L) = d + 10 * (eval L);
```

(size = length of argument list,
decreases by 1 in recursive call)

To prove:

For all values L:int list
there is an integer n such that
eval L =>* n

Exercise

- Prove the termination property for eval
- It's easy using simple induction on the length of the argument list

When it doesn't work

You cannot use
simple induction
for

```
fun decimal (n:int) : int list =
   if n<10 then [n]
          else (n mod 10) :: decimal (n div 10)
```

Why not?

Strong induction

- To prove a property of the form

  P(n), for all non-negative integers n

- Show that, for all k ≥ 0, P(k) follows logically from {P(0), ... , P(k-1)}.

you can use any of these to establish P(k)

Why it works

- P(0) gets a direct proof WHY?
- P(1) follows from P(0)
- P(2) follows from P(0), P(1)
- P(3) follows from P(0), P(1), P(2)
- For k>0 at the k^{th} step we've already shown P(0),..., P(k-1), and P(k) follows

Using strong induction

- Q: When can I use *strong* induction to prove a property of a recursive function f ?
- A: When there is a *non-negative* measure of *argument size* and f(x) only makes recursive calls of form f(y) with *size*(y) < *size*(x)

Example

```
fun decimal (n:int) : int list =
   if n<10 then [n]
            else (n mod 10) :: decimal (n div 10);
```

(when $n \geq 10$, $0 \leq n \text{ div } 10 < n$)

To prove:

For all values n:int such that $n \geq 0$,
eval(decimal n) = n

Proof by strong induction

- For $0 \leq n < 10$, show directly that eval(decimal n) = n
- For $n \geq 10$, assume that

  For each m such that $0 \leq m < n$,
  eval(decimal m) = m

  Show that eval(decimal n) =n

Proof sketch

- For n ≥ 10 let r = n mod 10, q = n div 10.

  eval(decimal n)

  = eval ((n mod 10) :: decimal(n div 10))

  = eval (r :: decimal q)

- Since 0 ≤ q < n it follows from IH that

  eval(decimal q) = q

- Hence there is a list value Q such that

  decimal(q) = Q

  And eval (r :: decimal q) = eval (r::Q)

  = r + 10 * eval(Q)

  = r + 10 * q = n

Notes

- We proved that for all values n $\geq$ 0, eval(decimal n) *evaluates* to n
- It follows that for all expressions e:int, if e =>* n and n $\geq$ 0, then eval(decimal e) =>* n
- Also possible to use induction based on *evaluational* reasoning to prove these results