15-150 Fall 2018

Stephen Brookes

Lecture 4

Recursion and induction
Last time

• Specification format for a function $F$
  • $\textit{type}$
  • $\textit{assumption}$ (REQUIRES)
  • $\textit{guarantee}$ (ENSURES)

For all (properly typed) $x$ satisfying the $\textit{assumption}$, $F \times$ satisfies the $\textit{guarantee}$
Today

Showing that a specification is valid

• **Proofs** by *induction*
  • templates to help with accuracy

• **Focus on** *examples*

program structure guides proof
What is a proof?

A proof is a connected series of statements intended to establish a proposition.

No it isn’t!

Yes it is!

It’s not just contradiction.
It CAN be.

No it ISN'T!
What is a proof?

- A proof is a sequence of steps.
- Each step must follow logically from *math facts* or the results of *earlier steps*. 
Simple induction

- To prove a property of the form $P(n)$, for all non-negative integers $n$
- First, prove $P(0)$.
- Then show that, for all $k \geq 0$, $P(k+1)$ follows logically from $P(k)$.
Why this works

- P(0) gets a direct proof  \textit{base}
- P(0) implies P(1)  \textit{step}
- P(1) implies P(2)  \textit{step}
- ...  

For each \( n \geq 0 \) we can establish P(n)  
(follows from \textit{base} after \( n \) uses of \textit{step})*
Example

fun f(x:int):int = if x=0 then 1 else f(x-1)

(* REQUIRES  n≥0       *)
(* ENSURES     f(n) = 1   *)

• To prove:

For all n:int such that n≥0, f(n) = 1

(same as: for all n≥0, for all e such that e=n, f(e) = 1)
proof (part 1)

fun f(x:int):int = if x=0 then 1 else f(x-1)

Let P(n) be “f(n) = 1”

Theorem: ∀n≥0. P(n)

Proof: By simple induction on n.

• **Base**: we prove P(0). Here’s a proof:

\[ f(0) = (\text{fn } x \Rightarrow \text{if } x=0 \text{ then } 1 \text{ else } f(x-1)) \, 0 \]
\[ = \text{if } 0=0 \text{ then } 1 \text{ else } f(0-1) \]
\[ = \text{if true then } 1 \text{ else } f(0-1) \]
\[ = 1 \]

So f(0) = 1. That’s P(0).
proof (part 2)

fun f(x:int):int = if x=0 then 1 else f(x-1)

• Inductive step:
  Let \( k \geq 0 \) and assume \( P(k) \), \( f(k) = 1 \).
  We prove \( P(k+1) \), \( f(k+1) = 1 \).

• Let \( v \) be the value of \( k+1 \), so \( v = k + 1 \).
  \[
  f(k+1) = (\text{fn } x \Rightarrow \text{if } x=0 \text{ then } 1 \text{ else } f(x-1))(k+1)
  = (\text{fn } x \Rightarrow \text{if } x=0 \text{ then } 1 \text{ else } f(x-1))(v)
  = \text{if } v=0 \text{ then } 1 \text{ else } f(v-1)
  = \text{if false then } 1 \text{ else } f(v-1)
  = f(v-1)
  = f(k) \quad \text{since } v = k + 1
  = 1 \quad \text{by assumption } P(k)
  
So \( P(k+1) \) holds.
Notes

• State the *induction hypothesis* clearly

• Use induction hypothesis only when *justified*

• Use equations and rules only when *justified*

• Use math and logic accurately

• Give explanation for non-trivial steps
NOT a proof

\[
\begin{align*}
(\text{fn } x \Rightarrow \text{ if } x=0 \text{ then } 1 \text{ else } f(x-1)) \ 0 &= 1 \\
\text{if } 0=0 \text{ then } 1 \text{ else } f(0-1) &= 1 \\
\text{if true then } 1 \text{ else } f(0-1) &= 1 \\
1 &= 1 \\
\text{true}
\end{align*}
\]

Why is this not a proof?

- A proof is a sequence of steps.
- Each step must follow logically from math facts or the results of earlier steps.
backwards = wrong

\[
\begin{align*}
0 &= 1 \\
1 &= 0 & \text{by symmetry} \\
0+1 &= 1+0 & \text{by adding} \\
1 &= 1 & \text{by arithmetic} \\
\text{true}
\end{align*}
\]

A “proof” that 0 = 1
Comments

• The spec and proof for $\forall n \geq 0. f(n) = 1$ used *equational* reasoning

• We could have worked with *evaluational* reasoning, but the details would be different

(let’s do it!)
Example
(using evaluational reasoning)

fun f(x:int):int = if x=0 then 1 else f(x-1)

(* REQUIRES e =>* n for some value n≥0 *)
(* ENSURES f(e) =>* 1 *)

- To prove:
  For all n≥0,
  for all e:int such that e =>* n, f(e) =>* 1
Proof by simple induction

fun f(x:int):int = \( \text{if } x=0 \text{ then } 1 \text{ else } f(x-1) \)

Let \( P(n) \) be “for all \( e:\text{int} \) such that \( e \Rightarrow^* n, f(e) \Rightarrow^* 1 \)”

To prove: \( \forall n \geq 0. P(n) \)

- **Base**: prove \( P(0) \). Suppose \( e \Rightarrow^* 0 \).

\[
\begin{align*}
f(e) & \Rightarrow (fn \ x \Rightarrow \text{if } x=0 \text{ then } 1 \text{ else } f(x-1))(e) \\
& \Rightarrow^* (fn \ x \Rightarrow \text{if } x=0 \text{ then } 1 \text{ else } f(x-1)) \ 0 \\
& \Rightarrow \text{if } 0=0 \text{ then } 1 \text{ else } f(0-1) \\
& \Rightarrow \text{if true then } 1 \text{ else } f(0-1) \\
& \Rightarrow 1
\end{align*}
\]

So \( f(e) \Rightarrow^* 1 \). This establishes \( P(0) \).
Proof by simple induction

fun f(x:int):int = if x=0 then 1 else f(x-1)

• **Inductive step:**
  Let \( k \geq 0 \) and assume \( \text{P}(k) \). Then prove \( \text{P}(k+1) \).

• Let \( v = k+1 \) and suppose \( e \Rightarrow^* v \).

\[
\begin{align*}
  f(e) & \Rightarrow (\text{fn } x \Rightarrow \text{if } x=0 \text{ then } 1 \text{ else } f(x-1))(e) \\
  & \Rightarrow^* (\text{fn } x \Rightarrow \text{if } x=0 \text{ then } 1 \text{ else } f(x-1))(v) \\
  & \Rightarrow \text{if } v=0 \text{ then } 1 \text{ else } f(v-1) \quad \text{since } v > 0 \\
  & \Rightarrow \text{if false then } 1 \text{ else } f(v-1) \\
  & \Rightarrow f(v-1) \quad \text{by } \text{P}(k), \text{since } v-1 \Rightarrow^* k \\
  & \Rightarrow^* 1
\end{align*}
\]

So \( \text{P}(k+1) \) holds.
Proof by simple induction

fun f(x:int):int = if x=0 then 1 else f(x-1)

P(n) is “for all e:int such that e =>* n, f(e) =>* 1”

Conclusion

• The base analysis proved P(0).

• The inductive analysis showed that for k≥0, P(k) implies P(k+1).

• Hence for all n≥0, P(n) holds.
Remarks

• In *equational* reasoning we don’t always have to mimic *evaluation* order

• Sometimes we can do *parallel* analysis steps that don’t reflect actual evaluation of code

• This may yield a shorter proof

fun f(x:int):int = if x=0 then 1 else f(x-1) + f(x-1)

For all n:int such that n≥0, f(n) = 2^n

For all n:int such that n≥0, f(n) =>* 2^n
Using simple induction

- **Q:** When can I use *simple* induction to prove a property of a recursive function $f$?
- **A:** When we can find a *non-negative* measure of *argument size* and show that if $f(x)$ calls $f(y)$ then $size(y) = size(x) - 1$

pick a notion of size appropriate for $f$
fun fact (x : int) : int = if x=0 then 1 else x * fact(x-1)

fun sum (L : int list) : int = case L of
    [] => 0
    | (x::R) => x + sum R

Which of these can be proven by simple induction?

For all n ≥ 0, fact n > n

fact is total

For all n > 1, fact n > n

For all n ≥ 0, fact n evaluates to an integer value

sum is total
Example

fun eval [ ] = 0
| eval (d::L) = d + 10 * (eval L)

(The length of the argument list decreases in the recursive call)

To prove:

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

• Prove the specification for `eval`

• It’s easy using simple induction on the length of the argument list

(this proof shows that

`eval : int list -> int`

is a `total` function)
Life’s not always simple

You cannot use simple induction on n 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 $k \geq 0$,

$P(k)$ follows logically from $P(0), \ldots, P(k-1)$.

You can use any, all, or none to establish $P(k)$
Why this works

- P(0) gets a direct proof
- P(0) implies P(1)
- P(0), P(1) imply P(2)
- P(0), P(1), P(2) imply P(3)

For each $k \geq 0$ we can establish P(k) with $k$ uses of step
Using strong induction

• Q: When can I use strong induction to prove a property of a recursive function $f$ ?

• A: When we can find a non-negative measure of argument size and show that if $f(x)$ calls $f(y)$ then $\text{size}(y) < \text{size}(x)$
Notes

• Sometimes, even for *simple* induction, it’s convenient to handle several “base” case argument values at the same time

• A proof using *strong* induction may not need a separate “base” case analysis
  • can sometimes handle *all* possible arguments in the “inductive step”
Example

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

( when n≥10, we get 0 ≤ n div 10 < n,
so the argument value decreases
in the recursive call )

To prove:

For all values n≥0,
eval(decimal n) = n
Proof by strong induction

• For $0 \leq n < 10$, show directly that $\text{eval}($decimal $n) = n$

• For $n \geq 10$, assume that
  
  For each $m$ such that $0 \leq m < n$, $\text{eval}($decimal $m) = m$

  Then show that $\text{eval}($decimal $n) = n$

use inductive analysis for cases that make a recursive call

multiple base cases handled together
Reminder

fun eval [ ] = 0
| eval (d::L) = d + 10 * (eval L)

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

For all values $n \geq 0$,
$$\text{eval(decimal n)} = n$$

Proof: by strong induction on $n$
Proof sketch
(the base cases)

• For $0 \leq n < 10$ we have
  \[
  \text{eval(\text{decimal n})}
  \]
  \[
  = \text{eval } [n]
  \]
  \[
  = n
  \]

(that was easy!)
Proof sketch
(the inductive part)

- For $n \geq 10$ let $r = n \mod 10$, $q = n \div 10$.
  
  $\text{eval}(\text{decimal}\ n)$
  
  $\quad = \text{eval}\ ((n \mod 10) :: \text{decimal}(n \div 10))$
  
  $\quad = \text{eval}\ (r :: \text{decimal}\ q)$

- Since $0 \leq q < n$ it follows from IH that
  
  $\text{eval}(\text{decimal}\ q) = q$

- Hence there is a list value $Q$ such that
  
  $\text{decimal}(q) = Q$ and $\text{eval}\ Q = q$
  
  So
  
  $\text{eval}\ (r :: \text{decimal}\ q) = \text{eval}\ (r :: Q)$
  
  $\quad = r + 10 \times \text{eval}(Q)$
  
  $\quad = r + 10 \times q = n$

This shows that $\text{eval}(\text{decimal}\ n) = n$
Proof sketch
(conclusion)

Let $P(n)$ be “eval(decimal n) = n”

• The base analysis shows $P(0), P(1), \ldots, P(9)$

• The inductive analysis shows that for $n \geq 10$, $P(n)$ follows from $\{P(0), \ldots, P(n-1)\}$

• Hence, for all $n \geq 0$, $P(n)$ holds
• We used equational reasoning to show that for all values \( n \geq 0 \), \( \text{eval(decimal n)} = n \)

• It follows that for all expressions \( e : \text{int} \), if \( e \Rightarrow^* n \) and \( n \geq 0 \), then \( \text{eval(decimal e)} \Rightarrow^* n \)

• It’s also possible to use evaluational reasoning to prove this result, inductively.
So far

• Simple and strong induction
• Examples of their use
• Just the beginning…

Next

• Another example
• What would you do?
Example

fun log(x:int):int = 
  if x=1 then 0 else 1 + log(x div 2)

(* log : int -> int *)

(* REQUIRES n > 0 *)

(* ENSURES log n keeps dividing n by 2 
   * until it gets to 1 
   *)

too vague... doesn’t describe the result 
too operational... talks about internal details
Example

fun log(x:int):int = 
    if x=1 then 0 
    else 1 + log(x div 2)

(* log : int -> int *)

(* REQUIRES n > 0 *)

(* ENSURES log n evaluates to an integer k *)

(* such that \(2^k \leq n < 2^{k+1}\) *)

describes the key properties of the result value
Exercise

• Show that for each integer $n > 0$, there is a unique integer $k$ such that $2^k \leq n < 2^{k+1}$
  • this $k$ is called the logarithm (base 2) of $n$
• Prove the spec for $\log$

This shows that $\log$ computes logarithms (base 2)
Exercise

• When $b > 1$ and $n > 0$ there is a unique integer $k$ such that $b^k \leq n < b^{k+1}$
  • this is called the logarithm (base $b$) of $n$

• Define a recursive ML function $\log : \text{int} \times \text{int} \rightarrow \text{int}$ such that $\log(n,b)$ returns the logarithm (base $b$) of $n$ computing base $b$ logarithms