



CS 3110

Proofs about Programs

Prof. Clarkson

Fall 2019

Today's music: Suite from *Downton Abbey* by John Lunn

CLICKER QUESTION 1

Review

Previously in 3110:

- Functional programming
- Modular programming
- Efficiency
- Interpreters

Next unit of course: [proofs about programs](#)

Today:

- Equational reasoning
- Proving correctness of recursive functions



Approaches to validation [lec 10]

- Social
 - Code reviews
 - Extreme/Pair programming
- Methodological
 - Design patterns
 - Test-driven development
 - Version control
 - Bug tracking
- Technological
 - Static analysis (“lint” tools, FindBugs, ...)
 - Fuzzers
- Mathematical
 - Sound type systems
 - “Formal” verification



Less formal: Techniques may miss problems in programs

All of these methods should be used!

Even the most formal can still have holes:

- did you prove the right thing?
- do your assumptions match reality?

More formal: eliminate *with certainty* as many problems as possible.

Verification

- In the 1970s, scaled to about tens of LOC
- Now, research projects scale to real software:
 - CompCert: verified C compiler
 - seL4: verified microkernel OS
 - Ynot: verified DBMS, web services
 - Four color theorem
 - Project Everest: verified HTTPS stack [in progress]
 - Etc.
- In another 40 years?

Our goals

- Write **small, pure functional programs**
 - no side effects, mutability, I/O; always terminating
 - integers, lists, options, trees
- Prove **correctness theorems**
 - CS 2800 mathematics: induction, logic
- Be **rigorous** but not completely formal

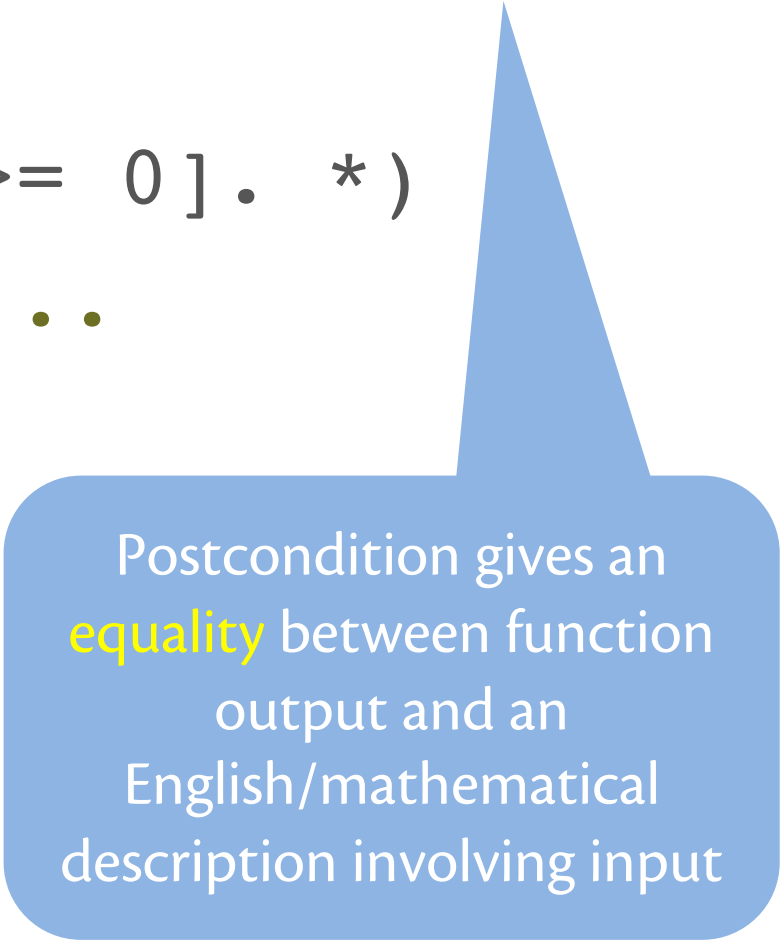
CORRECTNESS

Specifications

```
(** [fact n] is [n] factorial,  
    i.e., [n!].
```

```
    Requires: [n >= 0]. *)
```

```
let rec fact n = ...
```



Postcondition gives an **equality** between function output and an English/mathematical description involving input

Correctness proofs

- Based on equality between expressions
- When does $e = e'$?
 - Not asking about OCaml Boolean equality
 - Asking whether two pieces of code are equal...

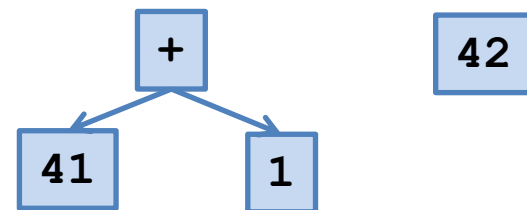
Equality of expressions

$$41 + 1 \stackrel{?}{=} 42$$

Semantically: **yes**

$$\begin{aligned} 41 + 1 &\rightarrow^* 42 \\ 42 &\rightarrow^* 42 \end{aligned}$$

Syntactically: **no**



Equality of expressions

fun $x \rightarrow x \stackrel{?}{=} \text{fun } y \rightarrow y$

Extensionality

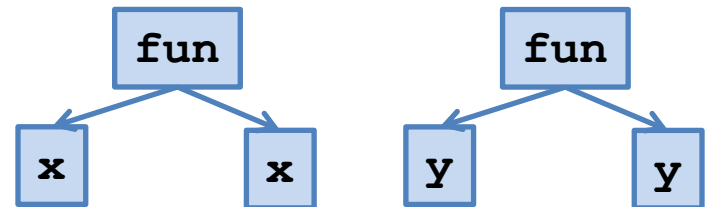
Semantically: **yes**

Syntactically: **no**

for all v ,

(**fun** $x \rightarrow x$) $v \rightarrow^* v$

(**fun** $y \rightarrow y$) $v \rightarrow^* v$



Equality of expressions

$$e = e'$$

if e and e' evaluate to the same value

must be well typed, pure, total

EQUATIONAL REASONING

Example 1

let twice f x = f (f x)

let compose f g x = f (g x)

twice h x = h (h x) (by evaluation)

compose h h x = h (h x) (by evaluation)

so

twice h x = compose h h x (by transitivity)

Example 1

let twice f x = f (f x)

let compose f g x = f (g x)

twice h x
= {evaluation}
h (h x)
= {evaluation}
compose h h x

Please use this
proof format

Example 2

let ($<<$) = compose

Theorem: composition is associative.

$$(f << g) << h = f << (g << h)$$

Proof: by extensionality, we need to show:

$$\text{forall } x, ((f << g) << h) x = (f << (g << h)) x$$

```
let compose f1 f2 x = f1 (f2 x)
```

Example 2

$$((f \ll g) \ll h) x = (f \ll (g \ll h)) x$$

$$\begin{aligned} & ((f \ll g) \ll h) x \\ &= \{ \text{evaluation} \} \\ & (f \ll g) (h x) \\ &= \{ \text{evaluation} \} \\ & f (g (h x)) \end{aligned}$$

$$\begin{aligned} & (f \ll (g \ll h)) x \\ &= \{ \text{evaluation} \} \\ & f ((g \ll h) x) \\ &= \{ \text{evaluation} \} \\ & f (g (h x)) \end{aligned}$$

QED

PROOFS WITH RECURSION

Example 1: Summation

$$\sum_{i=0}^n i = \frac{n(n+1)}{2}$$

```
let rec sumto n =  
  if n = 0 then 0  
  else n + sumto (n - 1)
```

sumto n ? n * (n + 1) / 2

Induction on natural numbers

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

Proof: by induction on n

Base case: $n = 0$

Show: $P(0)$

Inductive case: $n = k+1$

IH: $P(k)$

Show: $P(k+1)$

QED

Please use this
proof format

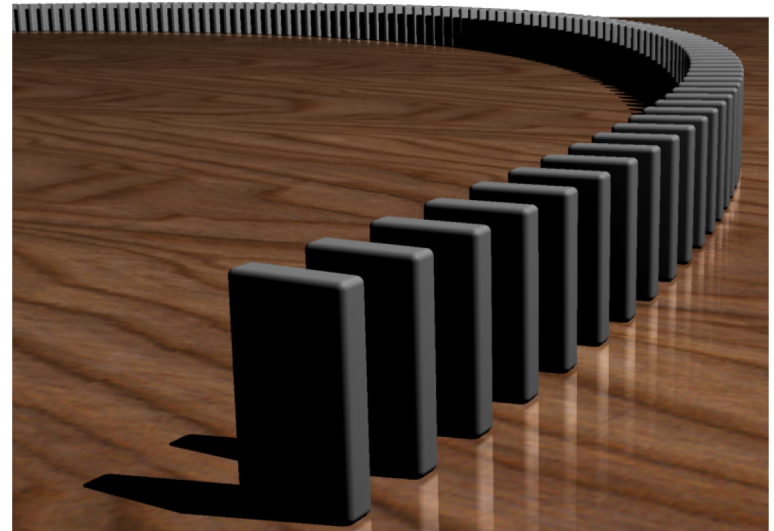
Induction principle on naturals

forall properties P ,

if $P(0)$

and (forall n , $P(n)$ implies $P(n+1)$)

then (forall n , $P(n)$)



Summation: proof structure

Claim: for all n , $\text{sum to } n = n * (n + 1) / 2$

Proof: by induction on n .

$$P(n) = \text{sum to } n = n * (n + 1) / 2$$

Base case: $n = 0$

Show: $\text{sum to } 0 = 0 * (0 + 1) / 2$

Inductive case: $n = k + 1$

IH: ???

CLICKER QUESTION 2

Summation: proof structure

Claim: for all n , $\text{sumto } n = n * (n + 1) / 2$

Proof: by induction on n .

$$P(n) = \text{sumto } n = n * (n + 1) / 2$$

Base case: $n = 0$

Show: $\text{sumto } 0 = 0 * (0 + 1) / 2$

Inductive case: $n = k + 1$

IH: $\text{sumto } k = k * (k + 1) / 2$

Show: $\text{sumto } (k + 1) = (k + 1) * ((k + 1) + 1) / 2$

Summation: base case

Base case: $n = 0$

Show: $\text{sumto } 0 = 0 * (0 + 1) / 2$

$\text{sumto } 0$

$= \{ \text{evaluation} \}$

0

$= \{ \text{algebra (or evaluation)} \}$

$0 * (0 + 1) / 2$

```
let rec sumto n =  
  if n = 0 then 0  
  else n + sumto (n - 1)
```

Summation: inductive case

Inductive case: $n = k + 1$

IH: $\text{sumto } k = k * (k + 1) / 2$

Show: $\text{sumto } (k + 1) = (k + 1) * ((k + 1) + 1) / 2$

$$\begin{aligned} & \text{sumto } (k + 1) \\ &= \{ \text{evaluation} \} \\ & \quad k + 1 + \text{sumto } k \\ &= \{ \text{IH} \} \\ & \quad k + 1 + k * (k + 1) / 2 \\ &= \{ \text{algebra} \} \\ & \quad (k + 1) * ((k + 1) + 1) / 2 \end{aligned}$$

QED

```
let rec sumto n =  
  if n = 0 then 0  
  else n + sumto (n - 1)
```

Example 2: Factorial

```
let rec fact n =  
  if n = 0 then 1  
  else n * fact (n - 1)
```

“i” suggests
iterative

```
let rec facti acc n =  
  if n = 0 then acc  
  else facti (acc * n) (n - 1)
```

```
let fact_tr n = facti 1 n
```

Factorial: correctness

Claim: for all n , $\text{fact } n = \text{facti } 1\ n$

Proof: by induction on n .

$P(n) = \text{fact } n = \text{facti } 1\ n$

Base case: $n = 0$

Show: $\text{fact } 0 = \text{facti } 1\ 0$

Inductive case: $n = k + 1$

IH: $\text{fact } k = \text{facti } 1\ k$

Show: $\text{fact } (k + 1) = \text{facti } 1\ (k + 1)$

Factorial: inductive case

Inductive case: $n = k + 1$

IH: $\text{fact } k = \text{facti } 1 \ k$

Show: $\text{fact } (k + 1) = \text{facti } 1 \ (k + 1)$

$\text{fact } (k + 1)$
 $= \{ \text{evaluation} \}$
 $(k + 1) * \text{fact } k$
 $= \{ \text{IH} \}$

$(k + 1) * \text{facti } 1 \ k$

STUCK

want to move $(k + 1)$ into accumulator:

$\text{facti } (k + 1) \ k$

but how?

$\text{facti } 1 \ (k + 1)$
 $= \{ \text{evaluation} \}$
 $\text{facti } (k + 1) \ k$

```
let rec fact n =  
  if n = 0 then 1  
  else n * fact (n - 1)
```

```
let rec facti acc n =  
  if n = 0 then acc  
  else facti (acc * n) (n - 1)
```

Strengthened IH

What we have from IH:

$$(k + 1) * \text{fact } k = (k + 1) * \text{facti } 1 \ k$$

What we want:

$$(k + 1) * \text{fact } k = \text{facti } (k + 1) \ k$$

want to multiply
(k + 1) into acc

So, strengthen $P(n)$ to give us what we want:

$$\text{forall } p, p * \text{fact } n = \text{facti } p \ n$$

can multiply
anything into acc

Factorial: correctness, take 2

Lemma: $\text{forall } n, \text{forall } p, p * \text{fact } n = \text{facti } p \ n$

Proof: by induction on n .

$P(n) = \text{forall } p, p * \text{fact } n = \text{facti } p \ n$

Base case: $n = 0$

Show: $\text{forall } p, p * \text{fact } 0 = \text{facti } p \ 0$

Inductive case: $n = k + 1$

IH: $\text{forall } p, p * \text{fact } k = \text{facti } p \ k$

Show: $\text{forall } p, p * \text{fact } (k + 1) = \text{facti } p \ (k + 1)$

Factorial: strengthened ind. case

Inductive case: $n = k + 1$

IH: forall p , $p * \text{fact } k = \text{facti } p \ k$

Show: forall p , $p * \text{fact } (k + 1) = \text{facti } p \ (k + 1)$

$p * \text{fact } (k + 1)$	$\text{facti } p \ (k + 1)$
$= \{ \text{evaluation} \}$	$= \{ \text{evaluation} \}$
$(p * (k + 1)) * \text{fact } k$	$\text{facti } (p * (k + 1)) \ k$
$= \{ \text{IH with } p := p * (k + 1) \}$	
$\text{facti } (p * (k + 1)) \ k$	

QED

```
let rec fact n =  
  if n = 0 then 1  
  else n * fact (n - 1)  
  
let rec facti acc n =  
  if n = 0 then acc  
  else facti (acc * n) (n - 1)
```

Factorial: correctness

Claim: forall n, fact n = fact_tr n

Proof:

fact_tr n
= { evaluation }
facti 1 n
= { previous lemma with p := 1 }
fact n

QED

Upcoming events

- [Wed] A6 due

This is rigorous.

THIS IS 3110