15-150 Fall 2020

Stephen Brookes

Lecture 4

Proving correctness

Last time

- Specification format for a function F
  - ***type***
  - ***assumption*** (REQUIRES)
  - ***guarantee*** (ENSURES)

For all (properly *typed*)
x satisfying the *assumption*,
F x satisfies the *guarantee*

Remember...

- Can use ***equivalence*** (a.k.a. equality, written =) to specify *applicative behavior* of functions
- Equality is *compositional*
- Equality is *defined* in terms of ***evaluation***
- $\Longrightarrow^*$ is *consistent* with = and ML evaluation

Example

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

f : int -> int

REQUIRES $x \geq 0$

ENSURES $f\ x = 1$

$f\ x = 1$

means the same as

$f\ x \Longrightarrow^{*} 1$

eval **spec**

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

eval : int list -> int

REQUIRES
  L is a list of decimal digits

ENSURES
  (eval L) $\Longrightarrow^*$ a non-negative integer

eval **spec**

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

eval : int list -> int

a *sufficient* REQUIRES
for the given ENSURES,
but not the most general

REQUIRES
  L is a list of decimal digits

ENSURES
  (eval L) $\Longrightarrow^*$ a non-negative integer

eval **spec**

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

eval : int list -> int

REQUIRES
  L is a list of decimal digits

ENSURES
  (eval L) $\Longrightarrow^*$ a non-negative integer

eval **spec**

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

eval : int list -> int

REQUIRES

L is a list of ~~decimal digits~~ non-negative integers

ENSURES

(eval L) $\Longrightarrow^*$ a non-negative integer

decimal **spec**

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

decimal : int -> int list

REQUIRES n >= 0

ENSURES

decimal n =

a list L of decimal digits

such that (eval L) = n

decimal **spec**

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

decimal : int -> int list

REQUIRES n >= 0

This REQUIRES property is just right, for the given ENSURES

ENSURES

decimal n =

a list L of decimal digits

such that (eval L) = n

Problem

How to show that a spec for a recursive function is valid

- Solution: Use ***induction*** to prove it
    - we offer templates to help with accuracy
- We focus on ***examples***...

program structure *guides* proof

But first, what's a **proof**?

What is a proof?

A proof is a logical sequence of steps, leading to a conclusion.

- Each step must *follow* logically
    from *math facts*,
        or the results of *earlier steps*.

Simple induction

- To prove a property of the form

$$\forall n \geq 0.\ P(n)$$

- First, prove $P(0)$.

base

- Then show that, for $k \geq 0$,
  $P(k+1)$ follows logically from $P(k)$.

inductive step

Why this works

- P(0) gets a direct proof *base*
- P(0) implies P(1) *step* *(with k=0)*
- P(1) implies P(2) *step* *(with k=1)*
- …

For each $n \geq 0$ we can establish P(n)

(follows from *base* after n uses of *step*)

Example

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

REQUIRES $n \geq 0$
ENSURES $f(n) = 1$

- To prove:

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

type REQUIRES ENSURES

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: $\forall n \geq 0.\ P(n)$

Proof: By simple induction on n.

- **Base**: we prove P(0). Here's a proof:

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: $\forall n \geq 0.\ P(n)$

Proof: By simple induction on n.

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

  f 0

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: $\forall n \geq 0.$ P(n)

Proof: By simple induction on n.

- **Base**: we prove P(0). Here's a proof:

  f 0 = (**fn** x => **if** x=0 **then** 1 **else** f(x-1)) 0

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: $\forall n \geq 0.$ P(n)

Proof: By simple induction on n.

- **Base**: we prove P(0). Here's a proof:

  f 0 = (**fn** x => **if** x=0 **then** 1 **else** f(x-1)) 0

  = **if** 0=0 **then** 1 **else** f(0-1)

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: $\forall n \geq 0$. P(n)

Proof: By simple induction on n.

- **Base**: we prove P(0). Here's a proof:

  f 0 = (**fn** x => **if** x=0 **then** 1 **else** f(x-1)) 0

  = **if** 0=0 **then** 1 **else** f(0-1)

  = **if true then** 1 **else** f(0-1)

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 = (**fn** x => **if** x=0 **then** 1 **else** f(x-1)) 0

  = **if** 0=0 **then** 1 **else** f(0-1)

  = **if true then** 1 **else** f(0-1)

  = 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: $\forall n \geq 0$. P(n)

Proof: By simple induction on n.

- **Base**: we prove P(0). Here's a proof:

  f 0 = (**fn** x => **if** x=0 **then** 1 **else** f(x-1)) 0

  = **if** 0=0 **then** 1 **else** f(0-1)

  = **if true then** 1 **else** f(0-1)

  = 1

  So f(0) = 1. That's P(0).

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: $\forall n \geq 0.\ P(n)$

Proof: By simple induction on n.

- **Base**: we prove P(0). Here's a proof:

  f 0 = (**fn** x => **if** x=0 **then** 1 **else** f(x-1)) 0

  = **if** 0=0 **then** 1 **else** f(0-1)

  justifications?

  = **if true then** 1 **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)

proof (part 2)

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

- **Inductive step**:
  Let k≥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) = (**fn** x => **if** x=0 **then** 1 **else** f(x-1))(k+1)

proof (part 2)

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

- **Inductive step**:

  Let k≥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) = (**fn** x => **if** x=0 **then** 1 **else** f(x-1))(k+1)

  = (**fn** x => **if** x=0 **then** 1 **else** f(x-1))(v)

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) = (**fn** x => **if** x=0 **then** 1 **else** f(x-1))(k+1)
  = (**fn** x => **if** x=0 **then** 1 **else** f(x-1))(v)
  = **if** v=0 **then** 1 **else** f(v-1)

proof (part 2)

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

- **Inductive step**:

  Let k≥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) = (**fn** x => **if** x=0 **then** 1 **else** f(x-1))(k+1)
  = (**fn** x => **if** x=0 **then** 1 **else** f(x-1))(v)
  = **if** v=0 **then** 1 **else** f(v-1)
  = **if false then** 1 **else** f(v-1)

proof (part 2)

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

- **Inductive step**:
  Let k≥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) = (**fn** x => **if** x=0 **then** 1 **else** f(x-1))(k+1)
  = (**fn** x => **if** x=0 **then** 1 **else** f(x-1))(v)
  = **if** v=0 **then** 1 **else** f(v-1)
  = **if false then** 1 **else** f(v-1)
  = f(v-1)

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) = (**fn** x => **if** x=0 **then** 1 **else** f(x-1))(k+1)
  = (**fn** x => **if** x=0 **then** 1 **else** f(x-1))(v)
  = **if** v=0 **then** 1 **else** f(v-1)
  = **if false then** 1 **else** f(v-1)
  = f(v-1)
  = f(k)

proof (part 2)

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

- **Inductive step**:
  Let k≥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) = (**fn** x => **if** x=0 **then** 1 **else** f(x-1))(k+1)
  = (**fn** x => **if** x=0 **then** 1 **else** f(x-1))(v)
  = **if** v=0 **then** 1 **else** f(v-1)
  = **if false then** 1 **else** f(v-1)
  = f(v-1)
  = f(k) since v=k+1

proof (part 2)

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

- **Inductive step**:
  Let k≥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) = (**fn** x => **if** x=0 **then** 1 **else** f(x-1))(k+1)
= (**fn** x => **if** x=0 **then** 1 **else** f(x-1))(v)
= **if** v=0 **then** 1 **else** f(v-1)
= **if false then** 1 **else** f(v-1)
= f(v-1)
= f(k) since v=k+1
= 1 by assumption P(k)

proof (part 2)

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

- **Inductive step**:
  Let k≥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) = (**fn** x => **if** x=0 **then** 1 **else** f(x-1))(k+1)
  = (**fn** x => **if** x=0 **then** 1 **else** f(x-1))(v)
  = **if** v=0 **then** 1 **else** f(v-1)
  = **if false then** 1 **else** f(v-1)
  = f(v-1)
  = f(k) since v=k+1
  = 1 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

Warning

- It's easy to write ***bogus*** proofs
- We want you to learn how to write ***excellent*** proofs
- Here are some bad examples, not to be copied…

Is this a proof of f 0 = 1?

Is this a proof of f 0 = 1?

f 0 = 1

Is this a proof of f 0 = 1?

f 0 = 1

(**fn** x => **if** x=0 **then** 1 **else** f(x-1)) 0 = 1

Is this a proof of f 0 = 1?

f 0 = 1

(**fn** x => **if** x=0 **then** 1 **else** f(x-1)) 0 = 1

if 0=0 **then** 1 **else** f(0-1) = 1

Is this a proof of f 0 = 1**?**

f 0 = 1

(**fn** x => **if** x=0 **then** 1 **else** f(x-1)) 0 = 1

if 0=0 **then** 1 **else** f(0-1) = 1

if true then 1 **else** f(0-1) = 1

Is this a proof of f 0 = 1**?**

f 0 = 1

(**fn** x => **if** x=0 **then** 1 **else** f(x-1)) 0 = 1

if 0=0 **then** 1 **else** f(0-1) = 1

if true then 1 **else** f(0-1) = 1

1 = 1

Is this a proof of f 0 = 1?

f 0 = 1

(**fn** x => **if** x=0 **then** 1 **else** f(x-1)) 0 = 1

if 0=0 **then** 1 **else** f(0-1) = 1

if true then 1 **else** f(0-1) = 1

1 = 1

true

Is this a proof of f 0 = 1**?**

f 0 = 1

(**fn** x => **if** x=0 **then** 1 **else** f(x-1)) 0 = 1

if 0=0 **then** 1 **else** f(0-1) = 1

if true then 1 **else** f(0-1) = 1

1 = 1

true

Is this a proof of f 0 = 1?

f 0 = 1

(**fn** x => **if** x=0 **then** 1 **else** f(x-1)) 0 = 1

if 0=0 **then** 1 **else** f(0-1) = 1

if true then 1 **else** f(0-1) = 1

1 = 1

true

No, this just shows that

“*if* f 0 = 1 *then* true *is true***”**

Is this a proof of f 0 = 1**?**

f 0 = 1

(**fn** x => **if** x=0 **then** 1 **else** f(x-1)) 0 = 1

if 0=0 **then** 1 **else** f(0-1) = 1

if true then 1 **else** f(0-1) = 1

1 = 1

true

No, this just shows that

"*if* f 0 = 1 *then* true *is true***"**

The first line in this "proof" isn't (yet) a math fact!

is this a proof?

$2 = 1$	
$1 = 2$	by symmetry
$2+1 = 1+2$	by adding
$3=3$	by arithmetic
true	

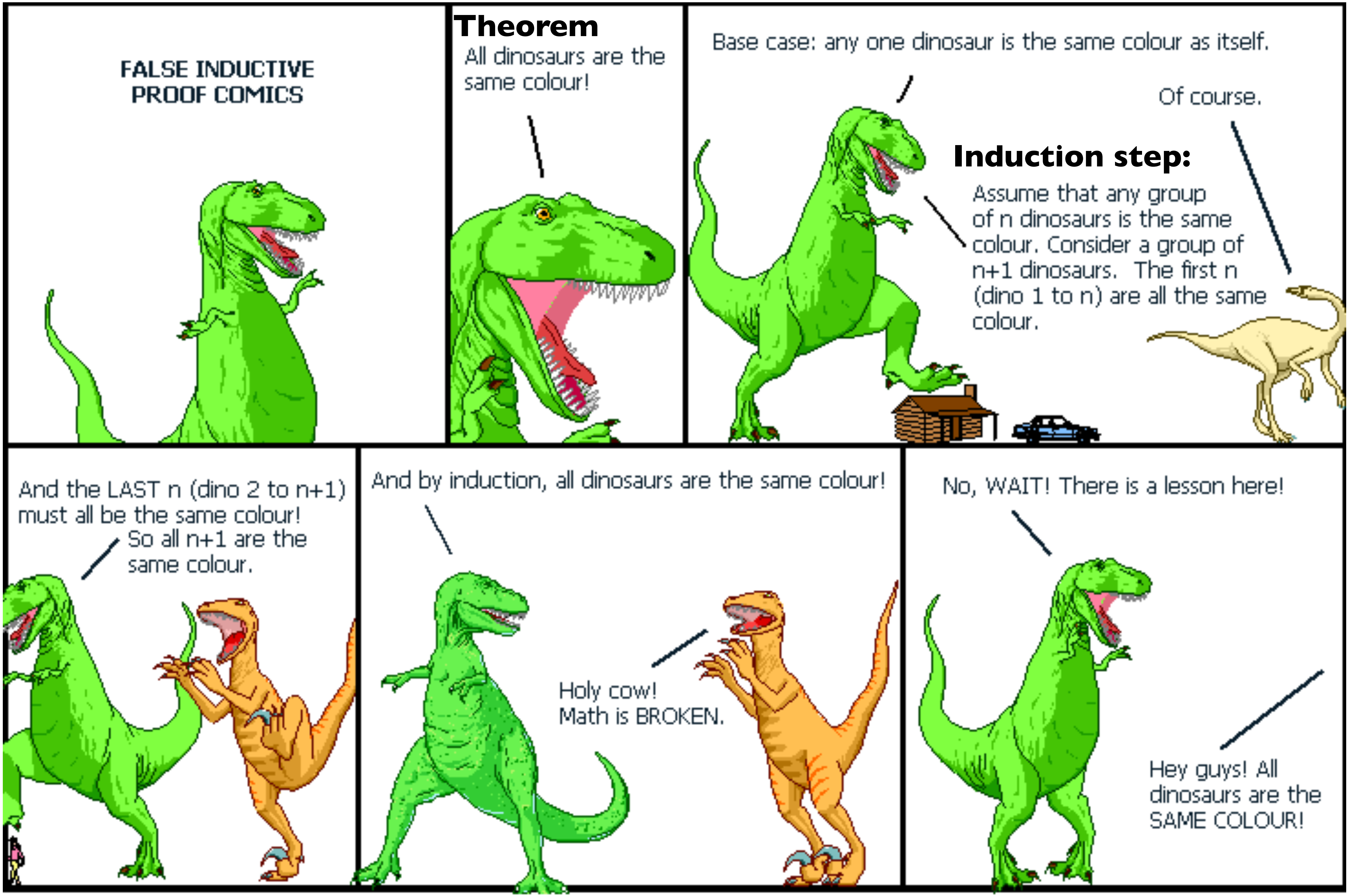
Is this a proof that $2 = 1$?

is this a proof?

2 = 1	
1 = 2	by symmetry
2+1 = 1+2	by adding
3=3	by arithmetic
true	

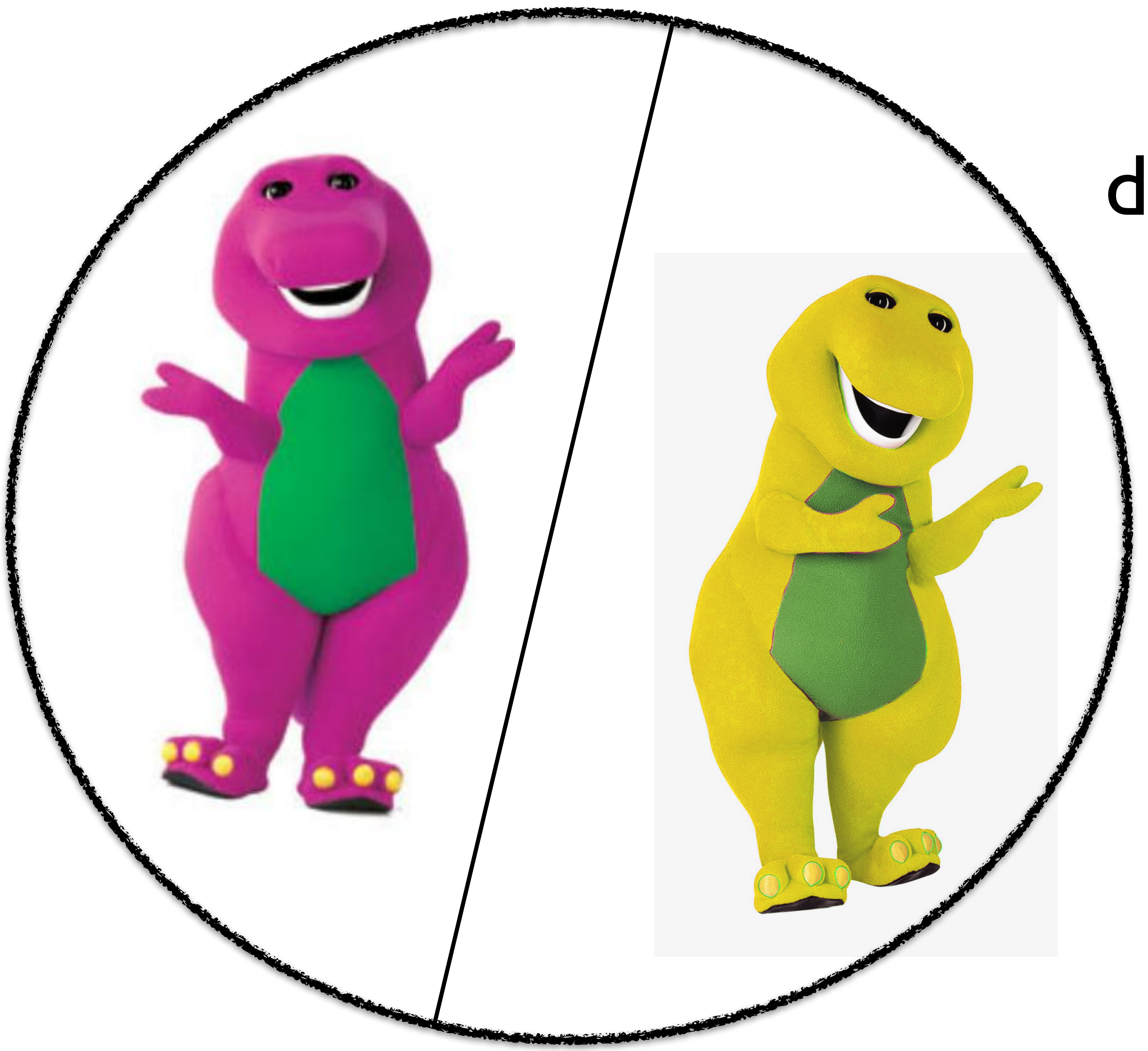
Is this a proof that 2 = 1?

Every non-empty set of dinosaurs has the same colo(u)r.



The proof is wrong

- The ***inductive step*** is inaccurate



P(2)

does not follow from

P(1)

Is this a proof?

```
fun silly(x:int):int = silly(x)
fun hitchhiker(n:int):int = 42
```

Claim

For all values x : int, hitchhiker(silly x) = 42.

Proof

hitchhiker(silly x) = (**fn** n => 42) (silly x)

= [(silly x)/n] 42

= 42 … QED

Is this a proof?

```
fun silly(x:int):int = silly(x)
fun hitchhiker(n:int):int = 42
```

Claim

For all values x : int, hitchhiker(silly x) = 42.

Proof

hitchhiker(silly x) = (**fn** n => 42) (silly x)

= [(silly x)/n] 42

= 42 … QED

No! The substitution step isn't justified, because (silly x) is not a value.

What is a proof?

A proof is a logical sequence of steps, leading to a conclusion.

- Each step must *follow* logically
  from *math facts*,
  or the results of *earlier steps*.

*An **excellent** proof has a true conclusion*

*A **bogus** proof can have a false conclusion*

(again)

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

Examples

```
fun fact (x:int) : int = if x=0 then 1 else x * fact(x-1)

fun sum [ ] = 0
  |   sum (x::R) = x + sum R
```

Examples

```
fun fact (x:int) : int = if x=0 then 1 else x * fact(x-1)

fun sum [ ] = 0
  |   sum (x::R) = x + sum R
```

Which of the following can be proven
by simple induction?

Examples

```
fun fact (x:int) : int = if x=0 then 1 else x * fact(x-1)

fun sum [ ] = 0
  |   sum (x::R) = x + sum R
```

Which of the following can be proven
by simple induction?

fact is total

Examples

```
fun fact (x:int) : int = if x=0 then 1 else x * fact(x-1)

fun sum [ ] = 0
  |   sum (x::R) = x + sum R
```

Which of the following can be proven
by simple induction?

Examples

```
fun fact (x:int) : int = if x=0 then 1 else x * fact(x-1)

fun sum [ ] = 0
  |   sum (x::R) = x + sum R
```

Which of the following can be proven
by simple induction?

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

Examples

```
fun fact (x:int) : int = if x=0 then 1 else x * fact(x-1)

fun sum [ ] = 0
  |    sum (x::R) = x + sum R
```

Which of the following can be proven
by simple induction?

Examples

```
fun fact (x:int) : int = if x=0 then 1 else x * fact(x-1)

fun sum [ ] = 0
  |    sum (x::R) = x + sum R
```

Which of the following can be proven
by simple induction?

sum is total

Examples

```
fun fact (x:int) : int = if x=0 then 1 else x * fact(x-1)

fun sum [ ] = 0
  |   sum (x::R) = x + sum R
```

Which of the following can be proven
by simple induction?

Examples

```
fun fact (x:int) : int = if x=0 then 1 else x * fact(x-1)

fun sum [ ] = 0
  |   sum (x::R) = x + sum R
```

Which of the following can be proven
by simple induction?

For all $n \geq 0$, fact n > n

Examples

```
fun fact (x:int) : int = if x=0 then 1 else x * fact(x-1)

fun sum [ ] = 0
  |   sum (x::R) = x + sum R
```

Which of the following can be proven
by simple induction?

Examples

```
fun fact (x:int) : int = if x=0 then 1 else x * fact(x-1)

fun sum [ ] = 0
  |   sum (x::R) = x + sum R
```

Which of the following can be proven by simple induction?

For all n>1, fact n > n

eval again

eval :int list -> int

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

To prove:

For all integer lists L
there is an integer n such that
eval L $\Longrightarrow^*$ n

eval again

eval :int list -> int

```
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 integer lists L
there is an integer n such that
eval L $\Longrightarrow^*$ n

Exercise

- Prove the specification for eval
- It's a simple induction on list length

This shows that
eval : int list -> int
is a **total** function.

For all values L : int list,
eval L evaluates to a value.

Life's not 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?

We need a *stronger* form of induction…

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).

inductive step

you can use any, all, or none to establish P(k)

Why this works

- P(0) gets a direct proof **WHY?**
- P(0) implies P(1) *step*
- P(0), P(1) imply P(2) *step*
- P(0), P(1), P(2) imply P(3) *step*

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 *size*(y) < *size*(x)

Notes

- Sometimes, even for *simple* induction,
  it's convenient to handle several "base" cases
  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)
```

To prove:

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

Example

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

When $n \geq 10$, we get $0 \leq n$ **div** $10 < n$

To prove:

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

Example

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

When $n \geq 10$, we get $0 \leq n \textbf{ div } 10 < n$

so the *argument value* decreases,
stays non-negative,
in the recursive call

To prove:

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

Proof by strong induction

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

multiple base cases handled together

- 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

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)
```

We want to prove:

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

Proof: will be by strong induction on n

Proof sketch

(the base cases)

- For $0 \leq n < 10$ we have

$$\begin{aligned} &\text{eval(decimal n)} \\ &\quad = \text{eval [n]} \\ &\quad = n \end{aligned}$$

(That was easy!)

(We used the function definitions!)

Proof sketch

(the inductive part)

Proof sketch

(the inductive part)

- For $n \geq 10$ let $r = n$ **mod** 10, $q = n$ **div** 10.

Proof sketch

(the inductive part)

- For $n \geq 10$ let $r = n$ **mod** 10, $q = n$ **div** 10.

  eval(decimal n)

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

  = eval (r :: decimal q)

Proof sketch

(the inductive part)

- For $n \geq 10$ 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 $0 \leq q < n$ it follows from IH that

Proof sketch

(the inductive part)

- For $n \geq 10$ 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 $0 \leq q < n$ it follows from IH that

  eval(decimal q) = q

Proof sketch

(the inductive part)

- For $n \geq 10$ let $r = n \textbf{ mod } 10$, $q = n \textbf{ div } 10$.

  eval(decimal n)

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

  = eval (r :: decimal q)

- Since $0 \leq q < n$ it follows from IH that

  eval(decimal q) = q

- Hence there is a list value Q such that

Proof sketch

(the inductive part)

- For $n \geq 10$ 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 $0 \leq q < n$ it follows from IH that

  eval(decimal q) = q

- Hence there is a list value Q such that

  decimal q = Q and eval Q = q

Proof sketch

(the inductive part)

- For $n \geq 10$ 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 $0 \leq q < n$ it follows from IH that

  eval(decimal q) = q

- Hence there is a list value Q such that

  decimal q = Q and eval Q = q

  So

Proof sketch

(the inductive part)

- For $n \geq 10$ 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 $0 \leq q < n$ it follows from IH that

  eval(decimal q) = q

- Hence there is a list value Q such that

  decimal q = Q and eval Q = q

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

  = r + 10 * (eval Q)

  = r + 10 * q = n

Proof sketch

(the inductive part)

- For $n \geq 10$ 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 $0 \leq q < n$ it follows from IH that

  eval(decimal q) = q

- Hence there is a list value Q such that

  decimal q = Q and eval Q = q

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

  = r + 10 * (eval Q)

  = r + 10 * q = n

This shows that eval(decimal n) = n

Proof sketch

(conclusion)

Let P(n) be "eval(decimal n) = n"

- The base analysis shows P(0), P(1),…, P(9)

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

- Hence, for all $n \geq 0$, P(n) holds

Notes

- We used equational reasoning to show that for all ***values*** $n \geq 0$, eval(decimal n) = n
- It follows that for all ***expressions*** e : int, if $e \Longrightarrow^* n$ and $n \geq 0$, then eval(decimal e) $\Longrightarrow^*$ n

So far

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

Next

- What would ***you*** do?

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

log

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

log

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

log : int -> int

log

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

log : int -> int

REQUIRES n > 0

log

```
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

log

```
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

log

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

log : int -> int

REQUIRES n > 0

log

```
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

log

```
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 $2^k \leq n < 2^{k+1}$

log

```
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

log computes *logarithms* (base 2)