Recitation 23: Proofs about Programs

Correctness as defined by spec

```
(** [fact n] is n factorial,
    or [n!] *)
let fact n = ...
```

Validation

- PP
- TDD
- Static analysis
- Mathematical "Formal verification"

Informal $\leftrightarrow$ Formal

Equality of expressions

$e = e'$

$e \xrightarrow{*} v \xleftarrow{*} e'$

$41 + 1 = 42$

$41 + 1 \xrightarrow{1} 42 \xleftarrow{0} 42$

OCaml =

Functions?

- $\text{fun } x \to x \stackrel{?}{=} \text{fun } y \to y$

Define $f = g$ if for all inputs v, $f\ v = g\ v$

"extensionality"

$(\text{fun } x \to x)\ v = (\text{fun } y \to y)\ v$ for all v

$(\text{fun } x \to x)\ v \xrightarrow{1} v \xleftarrow{1} (\text{fun } y \to y)\ v$

Gotchas to ask:

$e \stackrel{?}{=} e'$

e, e' must be

- well-typed (they do eval to a value)
- pure (no refs, I/O)
- total (no ∞ loops, no exceptions)

Example

```
let twice f x = f (f x)
let compose f g x = f (g x)
```

Show twice f x = compose f f x

twice h x = h (h x)

compose h h x = h (h x)

standard format:

twice h x

= {eval twice}

h (h x)

= {eval compose}

compose h h x

Technically this is Equational Reasoning

- transitivity
- substitution of equals for equals
- high school algebra!

Induction (2800)

Example:

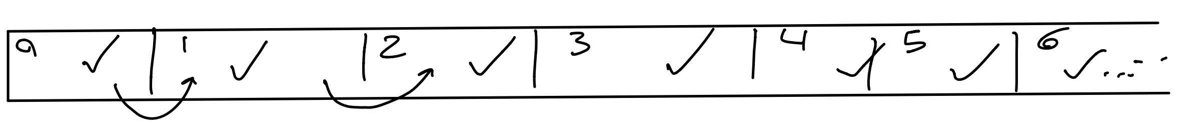
```
let rec sumto n =
  if n=0 then 0 else n + sumto (n-1)
```

show: sumto n = ((n − 1) * n) / 2

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

Recursive function → inductive proof

Induction on natural numbers



P(0)

if $P(k)$ then $P(k+1)$

Proof by induction on n

Base case: $n = 0$
Show $\quad$ sumto $0 = (0(1+0))/2$

sumto 0

$= \{\text{eval}\}$

0

$= \{\text{arithmetic}\}$

$(0(1+0))/2$

Inductive case

IH: sumto $k = (k * (k+1))/2$
Show: sumto $(k+1) = ((k+1) * ((k+1)+1))/2$

sumto $(k+1)$

$= \{\text{eval}\}$

$(k+1) +$ sumto k

$= \{\text{IH}\}$

$(k+1) + (k * (k+1))/2$

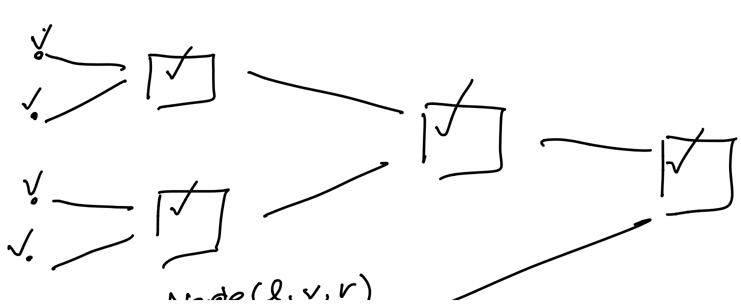
$= \{\text{algebra}\}$

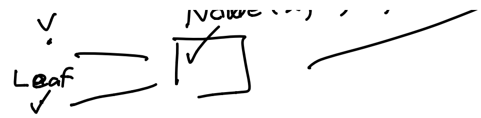
$((k+1) * ((k+1)+1))/2$

$\square$

Structural induction

Trees





P(Leaf)

if P(l) and P(r) then P(Node (l, v, r))

Example:

```
let rec nodes = function
  | Leaf -> 0
  | Node (l, _, r) -> 1 + nodes l + nodes r

let rec leaves = function
  | Leaf -> 1
  | Node (l, _, r) -> leaves l + leaves r
```

Show: leaves t = 1 + nodes t

Prove by induction on t

Base case : Leaf
Show : leaves Leaf = 1 + nodes Leaf

Inductive Case:

IH: leaves l = 1 + nodes l
leaves r = 1 + nodes r

Show: leaves(Node (l _, r)) = 1 + nodes(Node (l, _, r))

leaves (Node (l, _, r))

= {eval}

leaves l + leaves r

= {IH}

(1 + nodes l) + (1 + nodes r)

= {algebra}

1 + (1 + nodes l + nodes r)

= {eval}
  1 + nodes (Node (l, _, r))