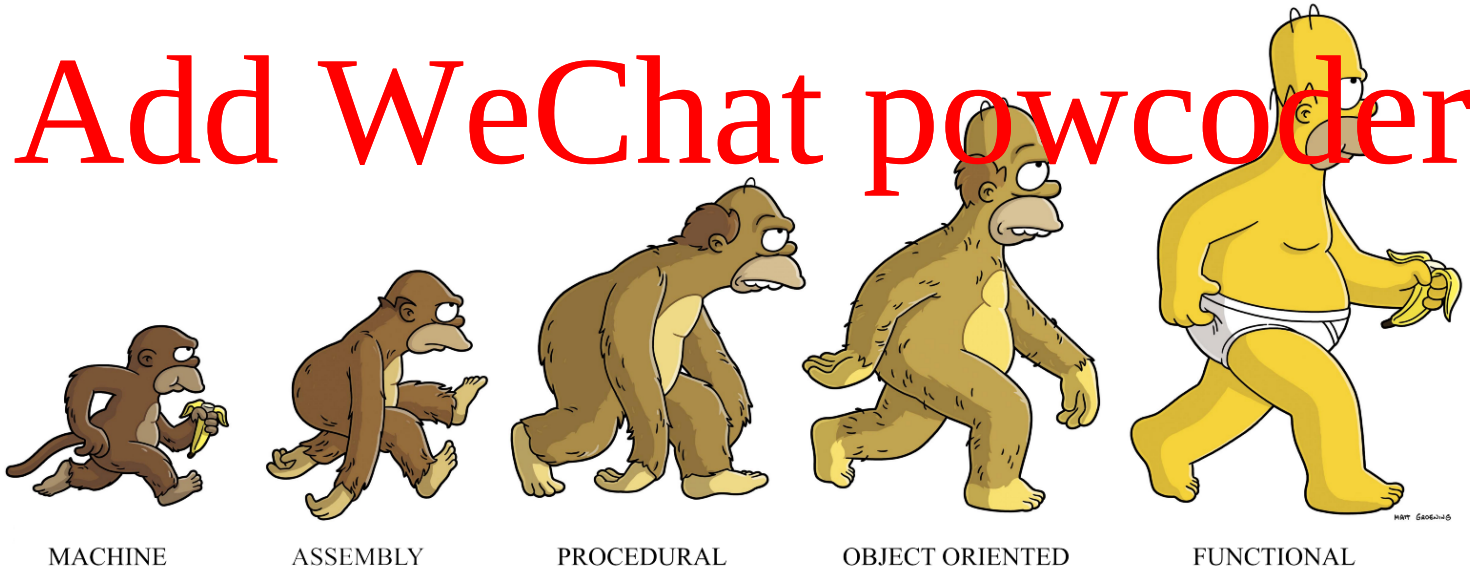
COMP302: Programming Languages and Paradigms

Prof. Brigitte Pientka (Sec 01)
bpientka@cs.mcgill.ca

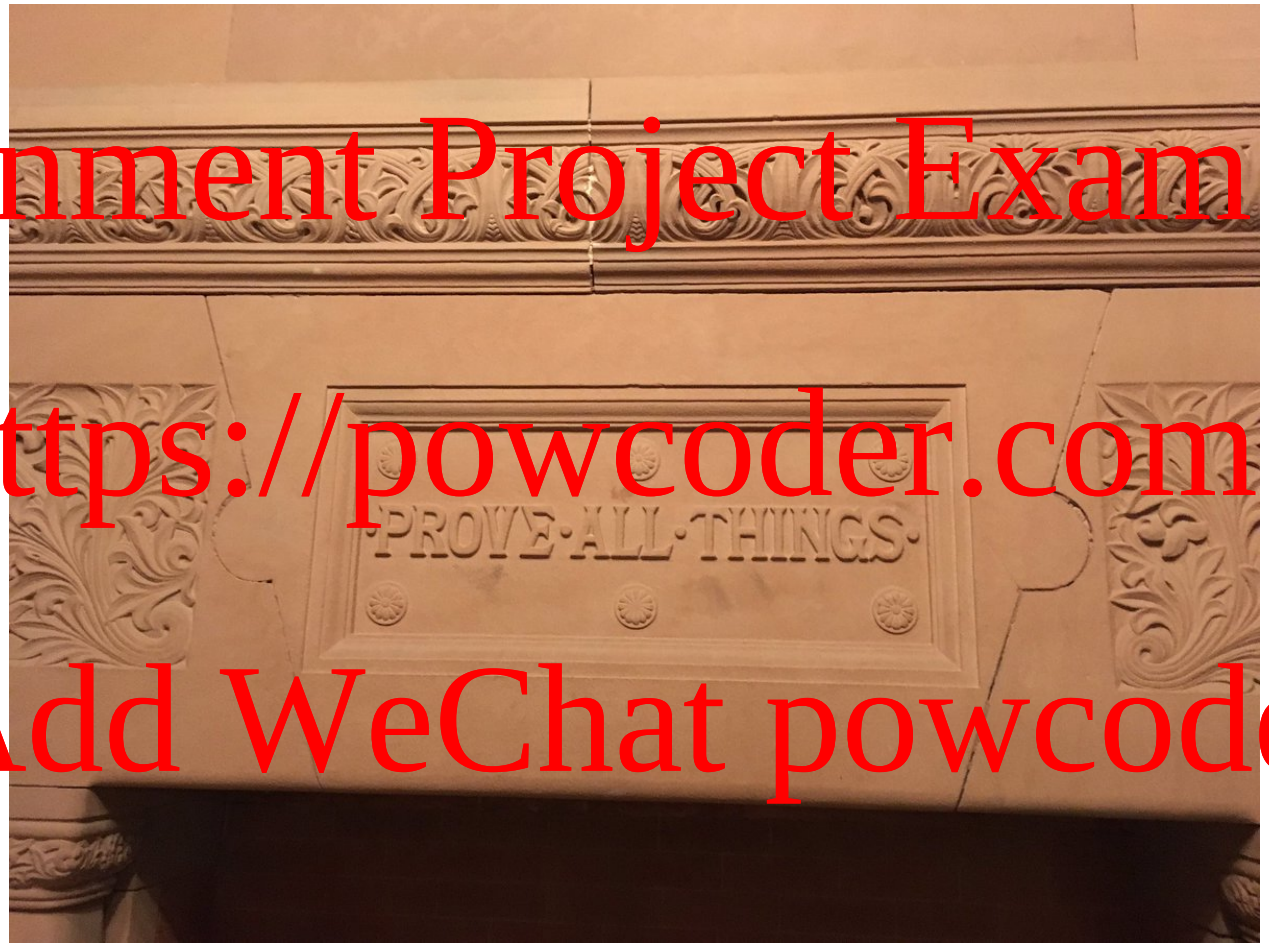
Francisco Ferreira (Sec 02)
fferre8@cs.mcgill.ca

School of Computer Science
McGill University

Week 2-1, Fall 2017



Functional Tidbit: Prove All Things!



Anybody knows where this is on campus?

Topic

Let's prove some stuff!

Warm Up: Lookup

Write a function `lookup: 'a -> ('a * 'b) list -> 'b option`. Given a key `k` of type `'a` and a list `l` of key-value pairs, return the corresponding value `v` in `l` (if it exists).

Warm Up:

Write a function `insert` which given a key `k` and a value `v` and an ordered list `l` of type `('a * 'b) list` it inserts the key-value pair `(k,v)` into the list `l` preserving the order.

Example:

```
insert (3,"a") [(2,"b") ; (7, "c")]
```

$\Longrightarrow^*$ `[(2, "b"); (3, "a"); (7, "c")]`

Warm Up:

Write a function `insert` which given a key `k` and a value `v` and an ordered list `l` of type `('a * 'b) list` it inserts the key-value pair `(k,v)` into the list `l` preserving the order.

Example:

```
insert (3,"a") [(2,"b") ; (7, "c")]
```

$\Longrightarrow^*$ `[(2, "b"); (3, "a"); (7, "c")]`

What is the relationship between `lookup` and `insert`?

Warm Up:

Write a function `insert` which given a key `k` and a value `v` and an ordered list `l` of type `('a * 'b) list` it inserts the key-value pair `(k,v)` into the list `l` preserving the order.

Example:

```
insert (3,"a") [(2,"b") ; (7, "c")]
```

$\Longrightarrow^*$ `[(2, "b"); (3, "a"); (7, "c")]`

What is the relationship between `lookup` and `insert`?

`lookup k (insert k v l)` returns `Some v`

How to prove it?

Step 1: We need to understand how programs are executed (operational semantics)

How to prove it?

Step 1: We need to understand how programs are executed
(operational semantics)

$e \Downarrow v$ expression e evaluates in multiple steps to the value v.
(Big-Step)

$e \Rightarrow e'$ expression e evaluates in one steps to expression e'.
(Small-Step (single))

$e \Longrightarrow^* e'$ expression e evaluates in multiple steps to expression e'
(Small-Step (multiple)).

How to prove it?

Step 1: We need to understand how programs are executed
(operational semantics)

$e \Downarrow v$ expression e evaluates in multiple steps to the value v.
(Big-Step)

$e \Rightarrow e'$ expression e evaluates in one step to expression e'.
(Small-Step (single))

$e \Longrightarrow^* e'$ expression e evaluates in multiple steps to expression e'
(Small-Step (multiple)).

For all `l`, `v`, `k`, `lookup k (insert k v l)` $\Longrightarrow^*$ `Some v`

How to prove it?

Step 2: How to reason inductively about lists?

How to prove it?

Step 2: How to reason inductively about lists?

- Analyze their structure

How to prove it?

Step 2: How to reason inductively about lists?

Analyze their structure

The recipe ...

To prove a property `P(l)` holds about a list `l`

Base Case: `l = []`

Show `P([])` holds

Step Case: `l = x::xs`

IH `P(xs)` — Assume the property `P` holds for lists smaller than `l`.

Show `P(x::xs)` holds — Show the property `P` holds for the original list `l`.

Let's prove something

```
let rec lookup k l = match l with
  | [] -> None
  | (k',v)::t ->
      if k = k' then Some v
      else lookup k t

let rec insert (k,v) l = match l with
  | [] -> [(k,v)]
  | ((k',v') as h) :: t ->
      if k = k' then (k,v)::t
        else
          if k < k' then (k,v)::l
            else h::insert (k,v) t
```


Theorem: For all `l`, `v`, `k`, `lookup k (insert (k,v) l)` $\Longrightarrow^*$ `Some v`

Let's prove something

```
let rec lookup k l = match l with
  | [] -> None
  | (k',v)::t ->
      if k = k' then Some v
      else lookup k t

let rec insert (k,v) l = match l with
  | [] -> [(k,v)]
  | ((k',v') as h) :: t ->
      if k = k' then (k,v)::t
        else
          if k < k' then (k,v)::l
            else h::insert (k,v) t
```


Theorem: For all `l`, `v`, `k`, `lookup k (insert (k,v) l)` $\Longrightarrow^*$ `Some v`

Lessons to take away

- State what you are doing induction on.

Proof by structural induction in the list `l`.

Lessons to take away

- State what you are doing induction on.

Proof by structural induction in the list `l`.

- Consider the different cases!

For lists, there are two cases – either `l = []` or `l = h::t`

Lessons to take away

- State what you are doing induction on.

Proof by structural induction in the list `l`.

- Consider the different cases!

For lists, there are two cases – either `l = []` or `l = h::t`

- State your induction hypothesis!

IH: For all `v`, `k`, `lookup k (insert (k, v) t) = Some v`

Lessons to take away

- State what you are doing induction on.

  Proof by structural induction in the list `l`.

- Consider the different cases!

  For lists, there are two cases – either `l = []` or `l = h::t`

- State your induction hypothesis!

  IH: For all `v`, `k`, `lookup k (insert (k, v) t) = Some v`

- Justify your evaluation / reasoning steps by
  - Referring to evaluation of a given program
  - The induction hypothesis
  - Lemmas / Properties (such as associativity, commutativity, etc.)