Accumulators

17 November 2020
Accumulators: Because sometimes recursion is slow.™
What’s fast? What’s slow?
Fast: Does one thing
Fast: Does one thing

Slow: Looks at every thing in list
Fast: Does one thing

Slow: Looks at every thing in list

Fast or slow?
*Fast*: Does one thing  
*Slow*: Looks at every thing in list

*Fast or slow?*

map
Fast: Does one thing

Slow: Looks at every thing in list

Fast or slow?
Fast: Does one thing

Slow: Looks at every thing in list

Fast or slow?

map

cons
Fast: Does one thing

Slow: Looks at every thing in list

Fast or slow?

cons

map
Fast: Does one thing

Slow: Looks at every thing in list

Fast or slow?

append
Fast: Does one thing

Slow: Looks at every thing in list

Fast or slow?
Fast: Does one thing

Slow: Looks at every thing in list

- cons
- map
- append

Fast or slow?

filter
Fast: Does one thing

Slow: Looks at every thing in list

cons

map
append
filter

Fast or slow?
Fast: Does one thing

Slow: Looks at every thing in list

Fast or slow?

first
Fast: Does one thing

Slow: Looks at every thing in list

Fast or slow?
**Fast**: Does one thing

- cons
- first

**Slow**: Looks at every thing in list

- map
- append
- filter

**Fast or slow?**

- rest
Fast: Does one thing

Slow: Looks at every thing in list

- cons
- first
- rest

- map
- append
- filter

Fast or slow?
Accumulators
A helper *accumulator function* can save a running total – the *accumulator* – so we can do fast things instead of slow things.
**Accumulator function**

Defined in **local**
Tracks a running total over multiple function calls

**Accumulator**

Parameter of accumulator function
The value accumulates over multiple function calls
Accumulator structure

(define (foo x y)
  (local [;; Signature
           ;; Purpose statement
           ;; ACCUMULATOR: DESCRIBE ME
           (define (foo/a x y acc)
                ...)]
     (foo/a x y acc0)))
Accumulator structure

\[
\text{(define (foo x y)} \\
\text{  (local [;; Signature} \\
\text{    ;; Purpose statement} \\
\text{    ;; ACCUMULATOR: DESCRIBE ME} \\
\text{    (define (foo/a x y acc)} \\
\text{      ...)]} \\
\text{  (foo/a x y acc0)))}
\]
Accumulator structure

(define (foo x y)
  (local [;; Signature
           ;; Purpose statement
           ;; ACCUMULATOR: DESCRIBE ME
           (define (foo/a x y acc)
             ...)]
    (foo/a x y acc0)))

Accumulator function
Accumulator structure

(define (foo x y)
  (local [;;;; Signature
          ;;;; Purpose statement
          ;;;; ACCUMULATOR: DESCRIBE ME
           (define (foo/a x y acc)
                ...
           )]
  (foo/a x y acc0)))
Accumulator structure

(define (foo x y)
  (local ;; Signature
      ;; Purpose statement
      ;; ACCUMULATOR: DESCRIBE ME
      (define (foo/a x y acc)
          ...
        ]
    (foo/a x y acc0))))
Accumulator structure

(define (foo x y)
  (local [;; Signature
           ;; Purpose statement
           ;; ACCUMULATOR: DESCRIBE ME
           (define (foo/a x y acc)
                ...)]
    (foo/a x y acc0)))

Good practice: Describe the accumulator (running total you’re saving)
Accumulator structure

(define (foo x y)
  (local [;; Signature
           ;; Purpose statement
           ;; ACCUMULATOR: DESCRIBE ME
           (define (foo/a x y acc)
             ...)]
  (foo/a x y acc0)))
Accumulator structure

(define (foo x y)
  (local []
    ;; Signature
    ;; Purpose statement
    ;; ACCUMULATOR: DESCRIBE ME
    (define (foo/a x y acc)
      ...))]
  (foo/a x y acc0)))

Call the accumulator function with the initial value for the accumulator argument.
Example: Accumulate a value
Q: What years did the Winter Olympics take place?
Q: What years did the Winter Olympics take place?

A: Well, they used to be every four years with the Summer Olympics, but then they kinda switched.
Q: What years did the Winter Olympics take place?

A: Well, they used to be every four years with the Summer Olympics, but then they kinda switched:

(list 1988 4 2 4 4)
Q: What years did the Winter Olympics take place?

A: Well, they used to be every four years with the Summer Olympics, but then they kinda switched:

   (list 1988 4 2 4 4)

Q: Okay, but \textit{when} were they?
Q: What years did the Winter Olympics take place?

A: Well, they used to be every four years with the Summer Olympics, but then they kinda switched:

(list 1988 4 2 4 4)

Q: Okay, but *when* were they?

A:

(list 1988 ___ ___ ___ ___ ___)
Q: What years did the Winter Olympics take place?

A: Well, they used to be every four years with the Summer Olympics, but then they kinda switched:

(list 1988 4 2 4 4)

Q: Okay, but *when* were they?

A:

(list 1988 (+ 1988 4) ___ ___ ___ ___)
Q: What years did the Winter Olympics take place?

A: Well, they used to be every four years with the Summer Olympics, but then they kinda switched:

(list 1988 4 2 4 4)

Q: Okay, but *when* were they?

A:

(list 1988 1992 ___ ___ ___ ___)
Q: What years did the Winter Olympics take place?

A: Well, they used to be every four years with the Summer Olympics, but then they kinda switched:

   (list 1988 4 2 4 4)

Q: Okay, but when were they?

A:

   (list 1988 1992 (+ 1992 2) ____ ___)
Q: What years did the Winter Olympics take place?

A: Well, they used to be every four years with the Summer Olympics, but then they kinda switched:
   (list 1988 4 2 4 4)

Q: Okay, but when were they?

A:  
   (list 1988 1992 1994 ____ ____)

Q: What years did the Winter Olympics take place?

A: Well, they used to be every four years with the Summer Olympics, but then they kinda switched:

(list 1988 4 2 4 4)

Q: Okay, but *when* were they?

A:

Q: What years did the Winter Olympics take place?

A: Well, they used to be every four years with the Summer Olympics, but then they kinda switched:

(list 1988 4 2 4 4)

Q: Okay, but *when* were they?

A:

Q: What years did the Winter Olympics take place?

A: Well, they used to be every four years with the Summer Olympics, but then they kinda switched:

   (list 1988 4 2 4 4)

Q: Okay, but *when* were they?

A:

Q: What years did the Winter Olympics take place?

A: Well, they used to be every four years with the Summer Olympics, but then they kinda switched:

   (list 1988 4 2 4 4)

Q: Okay, but *when* were they?

A:

The problem to solve

*Input*: List of starting point and relative values (Numbers)

*Output*: List of actual years
[Code: relative->abs]
The problem to solve

*Input*: List of starting point and relative values (Numbers)

*Output*: List of actual years

*Approach:*

  *Without accumulator*: Fine, but slow

  *With accumulator*: Better!
Example: Accumulate a list
Consider the humble conga line:
Consider the humble conga line:

“Conga, conga, conga!”
Consider the humble conga line:

“Conga, conga, conga!”
We can represent a *CongaLine* as *List-of String* with length $\geq 1$:

\[
\begin{align*}
\text{(define CONGA1 (list "Smithers"))} \\
\text{(define CONGA5 (list "Smithers" "Mr Burns" "Bart" "Lisa" "Homer"))}
\end{align*}
\]
What if the conga line is going to crash into a wall?

If everyone turns around, the line is reversed, and they can keep on conga-ing.
We can write a function that will reverse any *CongaLine* (or, you know, any less ridiculous list):

```
(define CONGA1 (list "Smithers"))
(define CONGA5 (list "Smithers" "Mr Burns" "Bart" "Lisa" "Homer"))

(reverse-list CONGA1)  →  (list "Smithers")
(reverse-list CONGA5)  →  (list "Homer" "Lisa" "Bart" "Mr Burns" "Smithers")
```
One way to reverse elements in a list would be to use the built-in `append` function:

```scheme
(define (reverse-list-v1 lst)
  (cond [(empty? lst) '()]
        [(cons? lst)
         (append (reverse-list-v1 (rest lst))
                 (list (first lst)))]))
```
But the way `append` works is it walks through the first list until it gets to the end, then it points to the second list.
> (append CONGA2 (list BART))
(list "Smithers" "Mr Burns" "Bart")
Racket needs to make a new *copy* of the first argument to *append*.
This makes for a very inefficient `reverse-list` function since we call `append` as many times as there are elements in the list we’re reversing.

A more efficient approach is – again – to use an accumulator!
[Code: reverse-list]
Trace of evaluation of new reverse-list procedure

(reverse-list (list 1 2 3))
(reverse-list/a (list 1 2 3) (list))
(reverse-list/a (list 2 3) (list 1))
(reverse-list/a (list 3) (list 2 1))
(reverse-list/a (list) (list 3 2 1))
(list 3 2 1)
The direction of information
With normal, structural recursion, information from deeper in the structure is returned to computation shallower in the structure.

For a list, that means all results are built from the end; the \textit{first} item in the list is added to the result \textit{last}. 
(require racket/trace)

(define (count-to-n n)
  (if (= n 0)
      0
      (+ 1 (count-to-n (- n 1))))

(trace count-to-n)

(count-to-n 5)
(count-to-n 5)
(+ 1 (count-to-n 4))
(+ 1 (+ 1 (count-to-n 3)))
(+ 1 (+ 1 (+ 1 (count-to-n 2))))
(+ 1 (+ 1 (+ 1 (+ 1 (count-to-n 1)))))
(+ 1 (+ 1 (+ 1 (+ 1 (count-to-n 0)))))
(+ 1 (+ 1 (+ 1 (+ 1 0))))
(+ 1 (+ 1 (+ 1 1)))
(+ 1 (+ 1 2))
(+ 1 (+ 1 3))
(+ 1 4)
5
An accumulator sends information the other way – from shallower in the structure to deeper.

For a list, that means we're building the result starting with the first item; the last item in the list is added to the result last.
(require racket/trace)

(define (count-to-n-acc n acc)  
  (if (= n 0)  
      acc  
      (count-to-n-acc (- n 1)  
                       (+ acc 1))))

(trace count-to-n-acc)

(count-to-n-acc 5)
When we use an accumulator, there’s nothing waiting for a recursive call to finish before it can be evaluated.

The style of recursion accumulators allow is called *tail recursion*, and it allows us to write recursive functions that don’t need us to keep a growing stack of function calls.
Acknowledgments

This lecture incorporates material from:

Laney Strange