CS 135 — L10: Final Recursion Rules & Accumulators

CS 135 Notes • 2025-10-21

Website: student.cs.uwaterloo.ca/~cs135

Final Rules

- Always move toward a base case.
- **Nat**: `zero?` / `sub1`. **List**: `empty?` / `rest`.
- Parameters not involved in the descent can change (accumulators).

Nat & List in Lockstep

```
(define (index n lst)    ; return element or empty if OOB
  (cond [(empty? lst) empty]
        [(zero? n) (first lst)]
        [else (index (sub1 n) (rest lst))]))
```

Slices:

```
(define (first-n n lst)
  (cond [(or (zero? n) (empty? lst)) empty]
        [else (cons (first lst) (first-n (sub1 n) (rest lst)))]))

(define (rest-n n lst)
  (cond [(or (zero? n) (empty? lst)) lst]
        [else (rest-n (sub1 n) (rest lst))]))
```

Two Lists in Lockstep

```
(define (dot-product xs ys)
  (cond [(or (empty? xs) (empty? ys)) 0]
        [else (+ (* (first xs) (first ys))
                 (dot-product (rest xs) (rest ys)))]))
```

```
(define (merge xs ys)
  (cond [(empty? xs) ys]
        [(empty? ys) xs]
        [(< (first xs) (first ys))
         (cons (first xs) (merge (rest xs) ys))]
        [(> (first xs) (first ys))
         (cons (first ys) (merge xs (rest ys)))]
        [else
         (cons (first xs)
               (cons (first ys) (merge (rest xs) (rest ys))))]))
```

Accumulators (tail recursion)

Reverse:

```
(define (rev/acc lst acc)
  (cond [(empty? lst) acc]
        [else (rev/acc (rest lst) (cons (first lst) acc))]))
(define (rev lst) (rev/acc lst empty))
```

Sum:

```
(define (sum/acc lst acc)
  (cond [(empty? lst) acc]
        [else (sum/acc (rest lst) (+ (first lst) acc))]))
(define (sum lst) (sum/acc lst 0))
```

Filter via accumulator (preserve order by one final reverse):

```
(define (filter/acc p? lst acc)
  (cond [(empty? lst) (rev acc)]
        [(p? (first lst)) (filter/acc p? (rest lst) (cons (first lst) acc))]
        [else (filter/acc p? (rest lst) acc)]))
```

> 🐋 **Info** — Accumulators convert “build-up at the end” (quadratic) into “build-to the front then reverse once” (linear).