Module 10: Accumulators

CPSC 110

Peyton Seigo

2018-11-13

Module 10: Accumulators

Learning goals

Structural recursion (on its own) doesn't let us see (1) where we've been in the traversal or (2) the work remaining to be done.

- Identify when a function design requires the use of accumulator.
- Work with the accumulator design recipe to design such functions.
- Understand and explain the concepts of tail position, tail call and tail recursion.

Accumulators

Three types of accumulators:

1. Context preserving
2. Result so far
3. Worklist

Accumulator HtDF Recipe

Main Idea

1. Structural recursion template
2. Wrap function in outer function, local, and trampoline
3. Add additional accumulator parameter

Three steps when filling in accumulator

1. Initialize accumualtor
2. Use/exploit accumulator value
    - Assume `acc` comment on what the accumulator represents is correct
3. Update accumulator to preserve invariant
    - Ensure `acc` value keeps invariant true

Full Recipe

1. Signature, purpose, stub.
2. Examples wrapped in `check-expect`s.
3. Template and inventory.
    - Template as usual
    - Wrap in function with same name; rename outer param (eg. `lox0`)
    - Trampoline: call inner function with outer param name

    - Add param to inner function; **add to each . . .**
    - In calls to inner function: specify type, invariant, and examples of accumulator
4. Code function body
5. Test and debug until correct

Example template operating on a list:

```
(@template (listof X) encapsulated accumulator)
(define (skip1 lox0)
  ;; acc: Natural; 1-based index of (first lox) in lox0
  ;; (skip1 (list "a" "b" "c") 1)
  ;; (skip1 (list     "b" "c") 2)
  ;; (skip1 (list         "c") 3)
  (local [(define (skip1 lox acc)
            (cond [(empty? lox) (... acc)]
                  [else
                   (... acc
                        (first lox)
                        (skip1 (rest lox)
                               (... acc)))]))]

    (skip1 lox0 ...)))
```

`add1` updates the accumulator to **preserve the invariant**.

Terminology

- **Accumulator invariant**: something that is always true about the accumulator (even if the exact value varies); varying quantity about a fact which does not vary
    - First accumulator comment in function