Module 9a: Generative Recursion

CPSC 110

Peyton Seigo

2018-11-03

Module 9a: Generative Recursion

Learning goals

- Identify whether a recursive function (or a set of mutually recursive functions) uses structural or generative recursion.
- Formulate a termination argument for a recursive function (or a set of mutually recursive functions).
- Design functions that use generative recursion (algorithms).

Generative Recursion HtDF Recipe

1. Check-expects
    - Start with trivial/base case
    - Then, go one case up
2. Template + body body
    - `(@template genrec)`
    - Copy gen-rec template from course website
    - Rename function + recursive call
    - Add + rename parameters

`gen-rec` template:

```
(@template genrec)
(define (genrec-fn d)
  ;; base case: _________
  ;; reduction: _________
  ;; argument:  _________

  (cond [(trivial? d) (trivial-answer d)]
        [else
         (... d
              (genrec-fn (next-problem d)))]))
```

Generative vs. Structural Recursion

Why is structural recursion guaranteed to end? Why isn't generative guaranteed to end?

Functions that use structural recursion driven by data, specifically **well-formed self-referential types**. These types can be `one-of` two things:

1. A value (like `empty` or `0`), or
2. Self-referential data

Data of these types MUST end at some point. Because structural recursion takes **sub-pieces** of the current data, they eventually lead to the base case.

Generative recursion is not based on such types. As such, our previous proof **does not apply anymore**. To deal with this, we write a three-part **termination argument**.

1. Base case.
2. Reduction step.
3. Argument that repeated application of reduction step will eventually reach the base case.

This process helps us reason about whether a function will stop.

Terminology

- Structural recursion: each recursive call takes a sub-piece of the starting data
- Generative recrusion: each recursive call generates **entirely new** data
  - We must prove—separately, for each function we write—that the function will terminate