Binary Trees, Arbitrary Trees

29 October 2020
A Pre-Halloween Warning:

This classroom was constructed on the site of the creepy, haunted attic of Sanders Physics.

😱
Lab preview
More pipes

A pipeline has faucets (opened or closed, straight parts (copper or lead), and branches.
More pipes

A pipeline has faucets (opened or closed, straight parts (copper or lead), and branches.
More pipes

A pipeline has faucets (opened or closed, straight parts (copper or lead), and branches.
More pipes

A pipeline has faucets (opened or closed, straight parts (copper or lead), and branches.
More pipes

A pipeline has faucets (opened or closed, straight parts (copper or lead), and branches.
(define-struct straight [kind next])
(define-struct branch [next1 next2])

;; A Pipeline is either:
;; - Boolean
;; - (make-straight String Pipeline)
;; - (make-branch Pipeline Pipeline)
Example pipelines

;;; A Pipeline is either:
;;; ‒ Boolean
;;; ‒ (make-straight String Pipeline)
;;; ‒ (make-branch Pipeline Pipeline)

#false
Example pipelines

;; A Pipeline is either:
;; - Boolean
;; - (make-straight String Pipeline)
;; - (make-branch Pipeline Pipeline)

#true
Example pipelines

;;; A Pipeline is either:
;;;   - Boolean
;;;   - (make-straight String Pipeline)
;;;   - (make-branch Pipeline Pipeline)

(make-straight "copper" #false)
Example pipelines

;;; A Pipeline is either:
;;; - Boolean
;;; - (make-straight String Pipeline)
;;; - (make-branch Pipeline Pipeline)

(make-straight "copper"
    (make-straight "lead" #false)
Example pipelines

;; A Pipeline is either:
;;   - Boolean
;;   - (make-straight String Pipeline)
;;   - (make-branch Pipeline Pipeline)

(make-branch
  (make-branch (make-straight "copper" #true)
                #false)
  #false)

(make-branch #false #false)
Programming with pipelines

;;; A Pipeline is either:
;;;  - Boolean
;;;  - (make-straight String Pipeline)
;;;  - (make-branch Pipeline Pipeline)
Programming with pipelines

;; A Pipeline is either:
;; - Boolean
;; - (make-straight String Pipeline)
;; - (make-branch Pipeline Pipeline)

Self-reference × 3
Programming with pipelines

;; A Pipeline is either:
;;  - Boolean
;;  - (make-straight String Pipeline)
;;  - (make-branch Pipeline Pipeline)

#;
;; Pipeline -> ...
(define (pipeline-template pl)
  (cond [(boolean? pl) ...]
       [(straight? pl)
         (... (straight-kind pl)
              (pipeline-template (straight-next pl)))]
       [(branch? pl)
         (... (pipeline-template (branch-next1 pl))
              (pipeline-template (branch-next2 pl))))])
Programming with pipelines

A Pipeline is either:
- Boolean
- (make-straight String Pipeline)
- (make-branch Pipeline Pipeline)

(define (pipeline-template pl)
  (cond [(boolean? pl) ...]
        [(straight? pl)
          (... (straight-kind pl)
               (pipeline-template (straight-next pl)))]
        [(branch? pl)
          (... (pipeline-template (branch-next1 pl))
               (pipeline-template (branch-next2 pl)))]))
Pipeline example

Design the function `water-running?` that takes a pipeline and determines whether any faucets are open.
A more realistic rumor mill
(make-gossip)
  (make-gossip '() '())
  (make-gossip)
    '()
    (make-gossip '() '())
    (make-gossip '() '())
Programming with rumors

;; A RumorMill is either
;; - '()
;; - (make-gossip Image RumorMill RumorMill)

#;
;; RumorMill -> ...
(define (rumor-mill-template rm)
  (cond [(empty? rm) ...]
        [(gossip? rm)
         (... (gossip-who rm)
              (... (gossip-next1 rm)
                    (rumor-mill-template (gossip-next1 rm)))
              (... (gossip-next2 rm)))
        ]))

Self-reference × 2

Natural recursion × 2
In our rumor mill, we restricted each person to spread gossip to at most two other people.

This isn’t very realistic; some gossips talk to lots of people!
Let each gossip talk to any number of people:

- Pansy
- Draco
- Cho
- Romilda
- Vincent
- Ginny
How do we represent an arbitrary number of gossip connections?
How do we represent an arbitrary number of gossip connections?

(define-struct gossip [who next])

;; A Gossip is
;; (make-gossip Image [List-of Gossip])
Programming with revised rumor mills

(define-struct gossip [who next])

;; A Gossip is
;; (make-gossip Image [List-of Gossip])

Remember what [List-of X] means?
Programming with revised rumor mills

(define-struct gossip [who next])

;; A Gossip is
;; (make-gossip Image ListOfGossip)

;; A ListOfGossip is either
;; - '()
;; - (cons Gossip ListOfGossip)

Writing out the full definition reminds us that there’s a self-reference in there!
Programming with revised rumor mills

(define-struct gossip [who next])

;; A Gossip is
;; (make-gossip Image ListOfGossip)

;; A ListOfGossip is either
;; - '()
;; - (cons Gossip ListOfGossip)
Programming with revised rumor mills

(define-struct gossip [who next])

;;; A Gossip is
;;; (make-gossip Image ListOfGossip)

;;; A ListOfGossip is either
;;; - ()
;;; - (cons Gossip ListOfGossip)
(define-struct gossip [who next])

;;; A Gossip is
;;; (make-gossip Image ListOfGossip)
;;; A ListOfGossip is either
;;; - ()
;;; - (cons Gossip ListOfGossip)

(define (gossip-template g)
  (... (gossip-who g) ...
       (log-template (gossip-next g)) ...))

(define (log-template l)
  (cond [(empty? l) ...]
        [(cons? l)
          (... (gossip-template (first l)) ...
               (log-template (rest l)) ...)]))
Design **count-gossips** which takes a gossip and returns the number of people informed by the gossip (including the starting person).
Acknowledgments

This lecture incorporates material from:

Matthias Felleisen
Robert Bruce Findler
Matthew Flatt
Shriram Krishnamurthi
Marc Smith