Intertwined Data

27 October 2020
Rumor mills
Ginny controls the rumor mill
Tracking rumors

Suppose we want to track gossip in a rumor mill.
Tracking rumors

Suppose we want to track gossip in a rumor mill.

Pansy
Tracking rumors

Suppose we want to track gossip in a rumor mill.

Pansy
Tracking rumors

Suppose we want to track gossip in a rumor mill.
Tracking rumors

Suppose we want to track gossip in a rumor mill.
Tracking rumors

Suppose we want to track gossip in a rumor mill.
Tracking rumors

Suppose we want to track gossip in a rumor mill.

Simplifying assumption: Each person tells at most two others.
Tracking rumors

Suppose we want to track gossip in a rumor mill.

Simplifying assumption: Each person tells at most two others.
Representing rumor mills

Is a rumor mill simply a list of people?
Representing rumor mills

Is a rumor mill simply a list of people? No, because there are relationships among the people.
Representing rumor mills

;;; A Person is
;;; (make-person Image Person Person Person)

How about this?
Representing rumor mills

;;; A Person is
;;; (make-person Image Person Person)

How about this? No, because some people don’t gossip to anyone else.
Representing rumor mills

(define-struct gossip [who next1 next2])
;; A RumorMill is either
;; - '()
;; - (make-gossip Image RumorMill RumorMill)

How about this?
Example rumor mills

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

'()
Example rumor mills

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

(make-gossip '() '())
Example rumor mills

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

(make-gossip
  '(')
  (make-gossip
    '(' '())

Romilda

Ginny
(make-gossip)
  (make-gossip  '() '())
    (make-gossip  '() '())
      (make-gossip  '() '())
        (make-gossip  '() '())
Example using constants:

```
(define GINNY-MILL
  (define DRACO-MILL
    (define ROMILDA-MILL
      (define CHO-MILL
        (define PANSY-MILL
          (definevincent-Mill
            (define ginny-Mill
              (make-gossip '() '()))
            (make-gossip '() '()))
            (make-gossip '() '()))
            (make-gossip '() '()))
            (make-gossip '() '()))
```
A *RumorMill* is a new kind of structure: a *tree*.

Each element in the tree is called a *node*.

The first node in the tree is called the *root*.

A node with no children is called a *leaf*.

Like a list, a tree is recursive: Every subtree is a tree.
Programming with rumors

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

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

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

For each element, there’s not just one “next” element; there are two!
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)
               (rumor-mill-template (gossip-next1 rm))
               (rumor-mill-template (gossip-next2 rm))))))
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)
                   (rumor-mill-template (gossip-next2 rm))))])

Self-reference × 2

Natural recursion × 2
Rumor program examples

Design the function `informed?` that takes a person image and a rumor mill and determines whether the person is part of the rumor mill.
Rumor program examples

Design the function `rumor-delay` that takes a rumor mill and determines the maximum number of days required for a rumor to reach everyone, assuming that each person waits a day before passing on a rumor.
Acknowledgments

This lecture incorporates material from:

Matthias Felleisen
Robert Bruce Findler
Matthew Flatt
Shriram Krishnamurthi
Marc Smith