CS 135 — L15: General Trees

CS 135 Notes • 2025-11-06

General Tree

Template of General Tree

```
(define (ntree-template nt)
  (cond
    [(number? nt)...]
    [(empty? (rest nt)) (... (ntree-template (first nt)))]
    [else (... (ntree-template (first nt)) (ntree-template (rest nt)))]))
```

Example:

```
(define nt-eg
  (list 6
    (list 8 9 3)
    (list 1
    (list 5 2) 7)4)
    )


;; A tree of numbers (NTree) is one of
;; * Num
;; * (cons NTree (listof NTree))

;; Predicate to check that something is an NTree
;; NTree?: Any -> Bool

(define (NTree? x)
  (cond
    [(number? x) true]
    [(or (not (list? x)) (empty? x)) false]
    [(empty? (rest x)) (NTree? (first x))]
    [else (and (NTree? (first x)) (NTree? (rest x)))]))

;; Add the numbers in a NTree
;; nt-add: NTree -> Num
(define (nt-add nt)
  (cond
    [(number? nt) nt]
    [(empty? (rest nt)) (nt-add (first nt))]
    [else (+ (nt-add (first nt)) (nt-add (rest nt)))]))
```

Potential Questions

1. Does an NTree contain a particular value?
2. Are all the values in an NTree positive?
3. How many times does a particular number appear?
4. How deep is an NTree?
5. How long is the longest list in an NTree?
6. Produce the equivalent NTree with all the values squared.
7. Are two trees “the same”?

Expression Trees

General Template

```
;; evaluate an arithmetic expression
;; eval: AExp -> Num
(define (eval exp)
  (cond
    [(number? exp) exp]
    [else (...) (first exp) (rest exp))]))
```

Atom

In Racket, and in related languages like Scheme and Lisp, any data that is not a list is sometimes called an `atom`.

Atoms include `Num`, `Bool`, `Sym`, and `Char`. A `Str` is considered to be an atom

We will use `Atom` in contracts to mean "anything not a list

Usage:

```
;; A (listof Any) is one of:
;; * empty
;; * (cons Atom (listof Any))
;; * (cons (listof Any) (listof Any))\
```

```
;; flatten an arbitrarily nested list to a list of atoms
;; flatten: (listof Any) -> (listof Atom)
(define (flatten lst)
  (cond
    [(empty? lst) lst]
    [(list? (first lst)) (append (flatten (first lst)) (flatten (rest lst)))]
    [else (cons (first lst) (flatten (rest lst)))]))
```

Quote

To use quote notation in Racket, simply prefix the list with an apostrophe—for example, write `'(a b c)` for `(list 'a 'b 'c)`. Notice there is just one apostrophe at the front of the list. Each symbol doesn't need an apostrophe.

Example:

```
'()
'(apple eggs bread apple milk bread)
'((1 1) (-1 1) (-1 -1) (-1 1))
'(6 (8 9 3) (1 (5 2) 7) 4)
'(#t (#f (#t (#f))))
'(+ (* 4 2) 3 (+ 5 1 2) 2)
```