Programming Paradigms CSI2120 – Winter 2018

Jochen Lang
EECS, University of Ottawa
Canada

Université d'Ottawa | University of Ottawa

uOttawa

L'Université canadienne
Canada's university

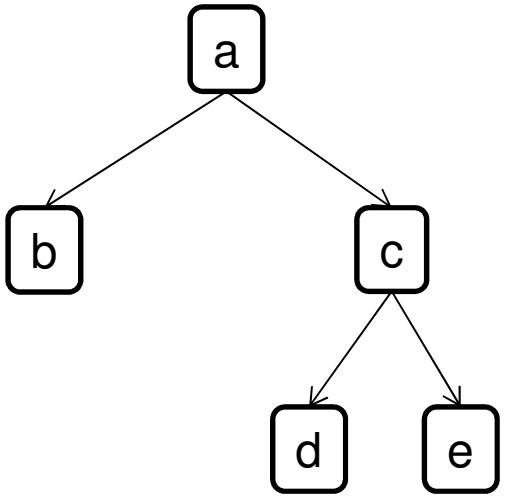
uOttawa.ca

Scheme: Functional Programming

- **Tree representations**
- **Binary search trees**

List Representation for Trees

- **A binary tree can be represented with nested lists**



(a b (c d e))

or

(a (b () ()) (c (d () ()) (e () ())))

or

(a b.(c d.e))

Test for Binary Tree

- **Test if a list confirms to the tree representation**

```
(define tree?
  (lambda (t)
    (cond
      ((not (list? t)) #f)
      ((null? t) #t)
      ((not (= (length t) 3)) #f)  ; node has 3 entries
      ((not (tree? (cadr t))) #f)  ; recurse left subtree
      ((not (tree? (caddr t))) #f) ; recurse right subtree
      (else #t)
)))
=> tree?
(tree? '(73 (31 (5 () ()) ()) (101 (83 () (97 () ())) ())))
=> #t
```

Inorder Traversal

- **Inorder traversal on a binary search tree will produce a sorted list**

```
(define (inorder t)
  (define traverse
    (lambda (t)
      (if (null? t) '()
      (append (traverse (cadr t)) (cons (car t)
                (traverse (caddr t))))
      )))
  (if
   (not (tree? t))
   (list 'not-a-tree t)
   (traverse t)
   ))
=> inorder
(inorder '(73 (31 (5 () ()) ()) (101 (83 () (97 () ()))
           ())))
=> (5 31 73 83 97 101)
```

Count the Type and Number of Elements in a Tree or List

- **Tree representation is a list**

```
(define (nsymbols tree)
  (if (pair? tree)
      (+ (nsymbols (car tree))
         (nsymbols (cdr tree)))
      (if (symbol? tree) 1 0)))
=> nsymbols
(nsymbols '(+ a (* b c)))
=> 5
```

- **Note the use of pair? instead of list?**
- **We could also use char? or number? for corresponding predicates**

Instead with Partial Tail Recursion

```
(define (nsymbols tree) (nsymbolst tree 0))
=> nsymbols
(define (nsymbolst tree n)
  (begin
    (display tree)(display " ")
    (display n)(newline) ; just to visualize
    (if (pair? tree)
      (nsymbolst (cdr tree)
                 (nsymbolst (car tree) n))
      (+ n (if (symbol? tree) 1 0)))))
=> nsymbolst
```

Tail Recursion

```
(nsymbols '(+ a (* b c)))
;;;;;;;;;;;;;;;;;
(+ a (* b c)) 0
(a (* b c)) 1
((* b c)) 2
(* b c) 2
(b c) 3
(c) 4
;;;;;;;;;;;;;;;;;
⇒ 5
```

The partial tail recursive version needs 6 tail recursive calls and 6 non tail recursive calls (compared to 12 with the double recursion)

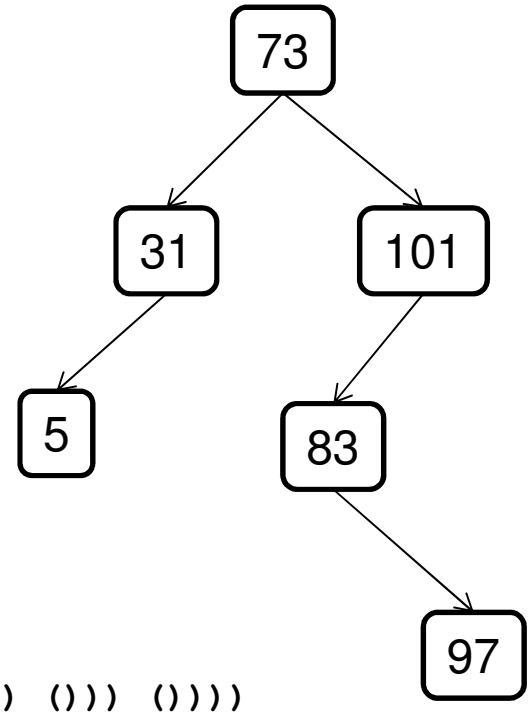
Conversion of a Tree into a List

```
(define (tree->list tree)
  (reverse (tree->list2 tree '())))

(define (tree->list2 tree lst)
  (if (pair? tree)
        (tree->list2 (cdr tree)
                     (tree->list2 (car tree) lst))
        (if (null? tree) lst (cons tree lst) )))
=> tree->list
(tree->list '(73 (31 (5 () ()) ()) (101 (83 ()
(97 () ())) ())))
=> (73 31 5 101 83 97)
```

Searching in a BST

```
(define search-BST
  (lambda (x t)
    (define search
      (lambda (x t)
          (cond
           ((null? t) #f)
           ((equal? x (car t)) #t)
           ((precedes? x (car t)) (search x (cadr t)))
           ((precedes? (car t) x) (search x (caddr t)))
           (else #f)
           )))
    (if
     (not (tree? t))
     (list 'not-a-tree t)
     (search x t)
     )))
=> search-BST
(define precedes? (lambda (x y) (< x y)))
=> precedes?
(search-BST 83 '(73 (31 (5 () ()) ()) (101 (83 () (97 () ())) ())))
=> #t
```



Insertion into a BST

```
(define (insert-BST tree value)
  (cond ((null? tree) (list value '() '()))
        ((< value (car tree))
         (list (car tree) (insert-BST (cadr tree) value)
               (caddr tree)))
        (else (list (car tree) (cadr tree)
                    (insert-BST (caddr tree) value)))))
=> insert-BST
(insert -BST '(73 (31 (5 () ()) ()) (101 (83 () (97 ()
())) ())) 86)
=> (73 (31 (5 () ()) ()) (101 (83 () (97 (86 () ())
())) ()))
```

Remove the Maximum from a BST

```
(define removemax-BST
  (lambda (t)
    (cond
     ((null? (caddr t)) (cons (cadr t) (car t)))
     (else
      (let ((r (removemax-BST (caddr t))))
       (cons (list (car t) (cadr t) (car r)) (cdr r))
        ))
     )))
=> removemax-BST
(removemax-BST '(73 (31 (5 () ()) ()) (101 (83 ()
(97 () ())) ())))
=> ((73 (31 (5 () ()) ()) (83 () (97 () ())) . 101)
```

Removal of a Node from a BST

```
(define delete
  (lambda (x t)
    (cond
     ((null? t) ())
     ((and (equal? x (car t)) (null? (cadr t))) (caddr t))
     ((and (equal? x (car t)) (null? (caddr t))) (cadr t))
     ((equal? x (car t))
      (let ((r (removemax-BST (cadr t))))
       (list (cdr r) (car r) (caddr t))
       ))
     ((precedes? x (car t)) (list (car t)
                     (delete x (cadr t)) (caddr t)))
     ((precedes? (car t) x) (list (car t) (cadr t)
                     (delete x (caddr t))))
     (else t)
    )))
```

Main Routine: Removal of a Node

```
(define delete-BST
  (lambda (x t)
    (if
      (not (tree? t))
      (list 'not-a-tree t)
      (delete x t)
)))
=> delete-BST
(delete-BST 101 '(73 (31 (5 () ()) ()) (101 (83
() (97 () ())) ())))
=> (73 (31 (5 () ()) ()) (83 () (97 () ())))
```