UNIVERSITY OF WATERLOO

David R. Cheriton School of Computer Science

CS 115: Introduction to Computer Science

Presented By Ahmed Ibrahim

Fall 2015

General Tree

- Binary trees can be used for a large variety of application areas. One limitation is the restriction on **the number of children**.
- What if there can be any number of children?
- How might we represent a node that can have up to three children?

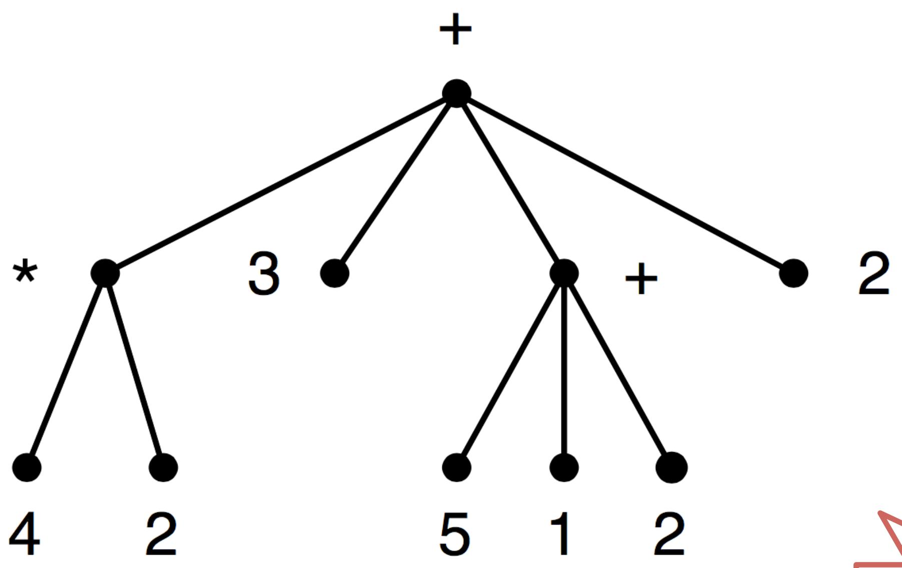
General arithmetic expressions

- For binary arithmetic expressions, we formed binary trees.
- Racket expressions using the functions + and ∗ can have an **unbounded number** of arguments.
- For **simplicity**, we will restrict the operations to + and ∗.

For example: (+ (∗ 4 2) 3 (+ 5 1 2) 2)

Visualizing the arithmetic expression

We can visualize an arithmetic expression as a general tree.



(+ (* 4 2) 3 (+ 5 1 2) 2)

There are labels on all the nodes of the tree

Structure definition of general arithmetic expressions

- For a <u>binary arithmetic expression</u>, we defined a structure with three fields: the **operation**, the **first argument**, and the **second argument**.
- For a <u>general arithmetic expression</u>, we define a structure with two fields: the **operation** and a **list of arguments** (which is a list of arithmetic expressions).
- We also need the data definition of a list of arithmetic expressions.

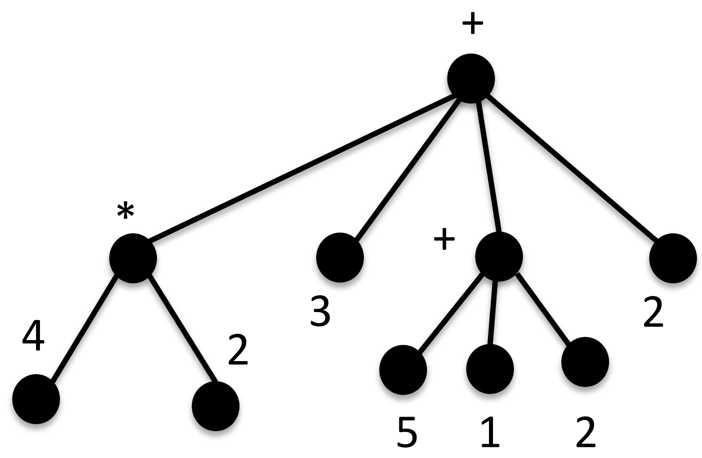
Structure definition of general arithmetic expressions (cont.)

```
;; Binary arithmetic expression:
(define-struct binode (op arg1 arg2))
```

```
(define-struct ainode (op args))
;;  An Arithmetic expression Internal Node (AINode) is a
;;  (make-ainode (anyof '* '+) (listof AExp))

;;  An Arithmetic Expression (AExp) is one of:
;;  * a Num
;;  * an AINode
```

General arithmetic expressions (cont.)



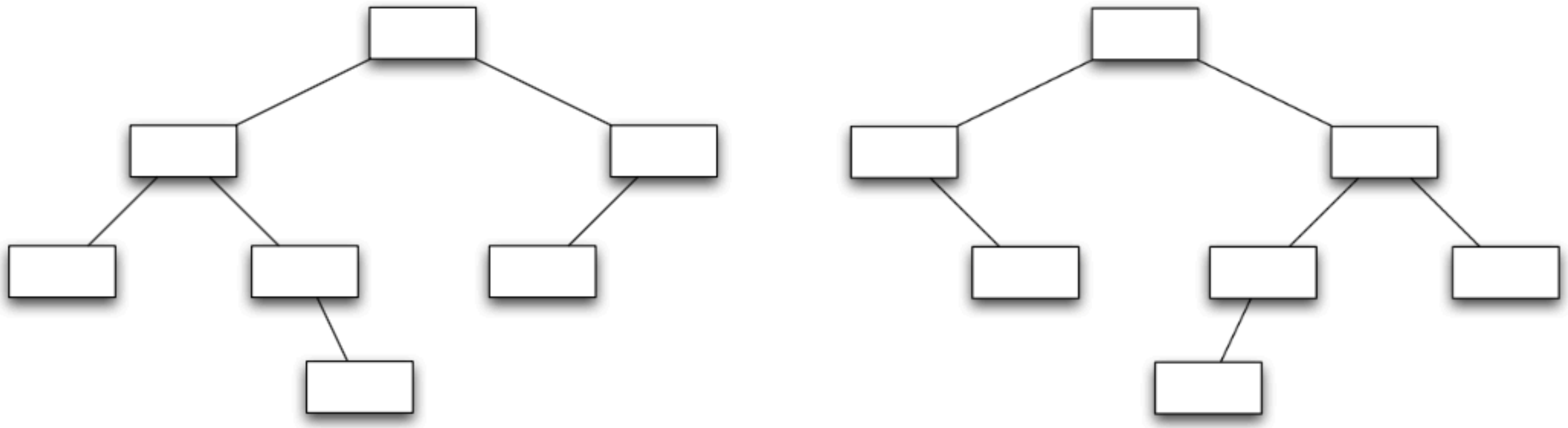
Examples of arithmetic expressions:

```
3
(make-ainode '+ (list 3 4))
(make-ainode '* (list 3 4))
(make-ainode '+
             (list (make-ainode '* (list 4 2))
                   3
                   (make-ainode '+ (list 5 1 2))
                   2))
```

It is also possible to <u>have an operation and an empty list.</u>

Additional Exercises

Place the keys 1, 2, 3, 4, 5, 6, and 7 into the following trees so that the resulting trees are *binary search trees*.



Mutual Recursion

- In computer science, **mutual recursion** is a form of recursion where two computational objects, such as functions or data types, are defined in terms of each other.
- Everything will be in pairs:

data definitions, templates, functions.

Mutual Recursion Example

```
(define (F n)
  (cond
    [(>= 0 n) 1]
    [else
     (- n (M (sub1 n)))]))
```

```
(define (M n)
  (cond
    [(>= 0 n) 0]
    [else
     (- n (F (sub1 n)))]))
```

Function remainder-n

Write a function remainder-n that consumes an AExp (in which all numbers are non-negative integers) and a positive integer n, and produces a new AExp in which all numbers have been replaced with their remainder when divided by n.

1. Check of the given AExp is number or not. [(number? ex) (remainder ex n)]
2. Start to create a new AExp by add the op into each node, then update the list. (make-ainode (ainode-op ex)…
3. Using update-args function will help to update args.

   (define (update-args exlist n)…

   1. Check if the list is empty or not. [(empty? exlist) empty]
   2. Start to construct a new list with an updated version of the args.

   (cons (remainder-n (first exlist) n) (update-args (rest exlist) n))

Templates for arithmetic expressions

```
;; (define (my-aexp-fun ex)
;; (cond
;;    [(number? ex) . . . ]
;;    [else . . . (ainode-op ex) . . .
;;           . . . (my-listof-aexp-fun (ainode-args ex)) . . . ]))

;; (define (my-listof-aexp-fun exlist)
;; (cond
;;    [(empty? exlist) . . . ]
;;    [else . . . (my-aexp-fun (first exlist)) . . .
;;           . . . (my-listof-aexp-fun (rest exlist)) . . . ]))
```

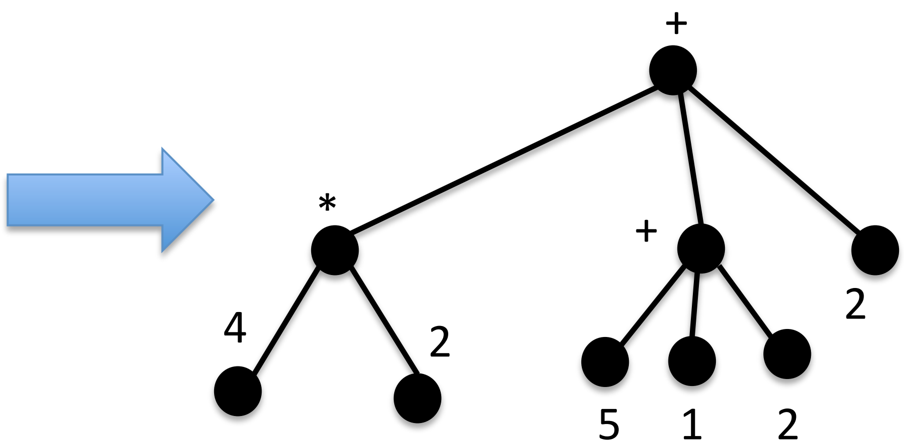
Alternate data definition

- In Module 6, we saw how a list could be used instead of a structure holding student information.
- Here we could use a similar idea to replace the structure AINode and the data definitions for AExp and (listof AExp).
- Each expression is a list consisting of a **symbol** (the operation) and a **list of expressions**.

Alternate data definition (cont.)

```
;; An Alternate arithmetic expression (AltAExp) is one of:
;; * a Num
;; * (cons (anyof '* '+) (listof AltAExp))
;; Examples:
3
(list '+ 3 4)
```

```
(list '+
        (list '* 4 2)
        (list '+ 5 1 2)
         2)
```



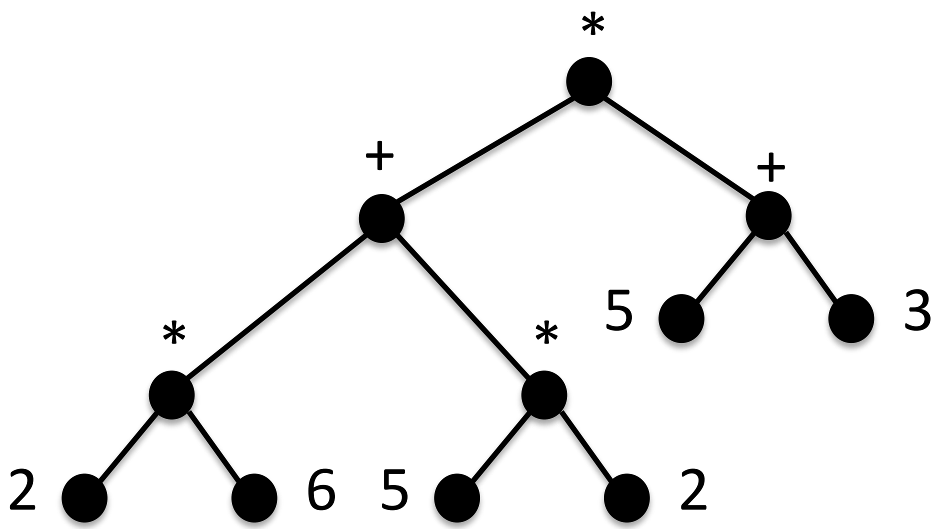
Templates for AltAExp and (listof AltAExp)

```
;; (define (my-aexp-fun ex)
;; (cond
;;       [(number? ex) . . . ]
;;       [else . . . (ainode-op ex) . . .
;;                  . . . (my-listof-aexp-fun (ainode-args ex)) . . . ]))
;; (define (my-listof-aexp-fun exlist)
;; (cond
;;       [(empty? exlist) . . . ]
;;       [else . . . (my-aexp-fun (first exlist)) . . .
;;              . . . (my-listof-aexp-fun (rest exlist)) . . . ]))
```

```
;; (define (my-altaexp-fun ex)
;; (cond
;; [(number? ex) . . . ]
;; [else . . .  (first ex) . . .
;;            . . . (my-listof-altaexp-fun (rest ex)) . . . ]))

;; (define (my-listof-altaexp-fun exlist)
;; (cond
;; [(empty? exlist) . . . ]
;; [else . . . (my-altaexp-fun (first exlist)) . . .
;;           . . . (my-listof-altaexp-fun(restexlist)) . . . ]))
```

An example

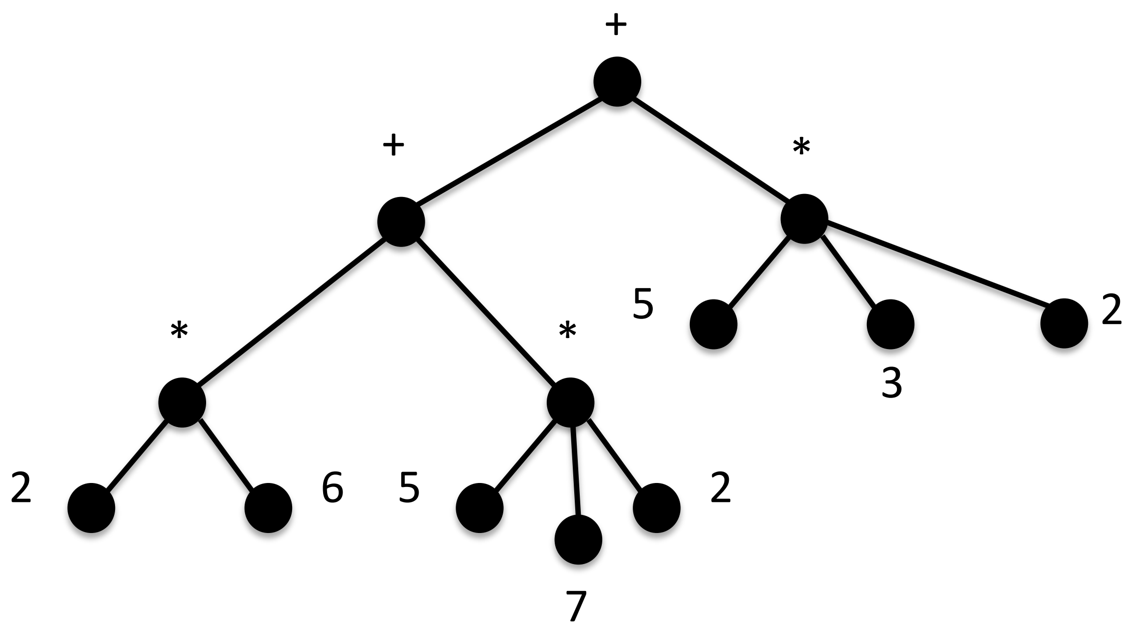


((2 * 6) + (5 * 2)) * (5 + 3)

```
(make-binode '*
    (make-binode '+
            (make-binode '* 2 6)
            (make-binode '* 5 2))
    (make-binode '+ 5 3))
```

```
(list '*
    (list '+
            (list '* 2 6)
            (list '* 5 2))
    (list '+ 5 3))
```

Another example



(list ‘+
    (list ‘+
        (list ‘* 2 6)
        (list ‘* 5 7 2))
    (list ‘* 5 3 2))

Some uses of general trees

The contents of organized text and web pages can be stored as a general list.

```
(list ’chapter
     (list ’section
          (list ’paragraph "This is the first sentence." "This is the second sentence.")
          (list ’paragraph "We can continue in this manner."))
     (list ’section . . . )
     ...
)
```

Some uses of general trees (cont.)

```
(list ’webpage
    (list ’title "CS 115: Introduction to Computer Science 1")
    (list ’paragraph "For a course description,"
        (list ’link "click here." "desc.html") "Enjoy the
                                                   course!")
    (list ’horizontal-line)
    (list ’paragraph "(Last modified yesterday.)"))
```

In lab, you will develop templates and write functions for general trees.

Additional Exercise

```
(define-struct node (key val left right))
;; A binary search tree (bst) is either:
;; - empty or
;; - a structure (make-node k v lft rgt) where
;; * k is an integer key,
;; * v is a string value, and ...
```

- Write the function *sum-leaves* that consumes an **BT** and produces the **sum of all the values in BT**. The sum of the leaves in an empty tree is 0.

- Write the function *min-key*. It consumes a non-empty *bst* and produces the **smallest key** it contains. Your function must not visit every node in the tree.