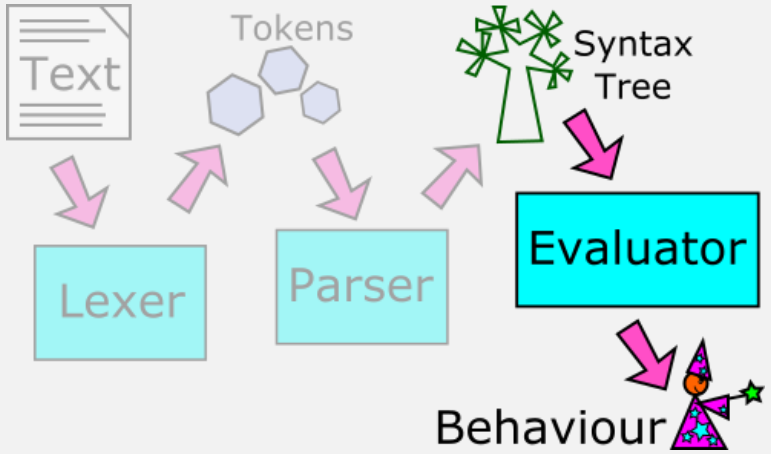
UMass Boston Computer Science

CS450 High Level Languages

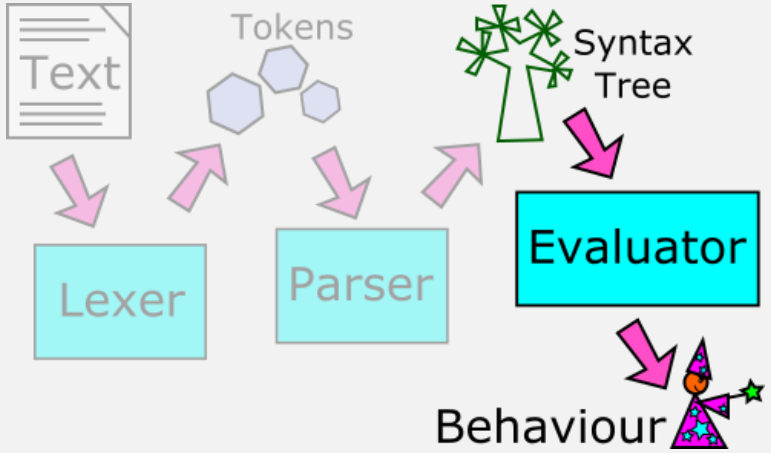
Interpreters and “eval”

Thursday, April 10, 2025



Logistics

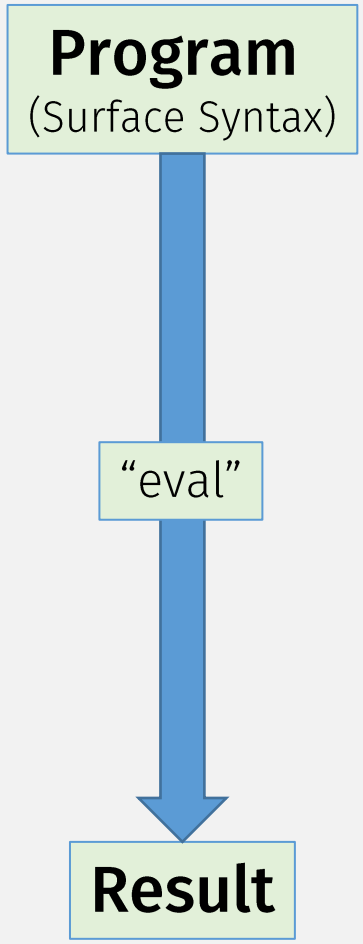
- HW 9 out
  - due: Tues 4/15, 11am EST



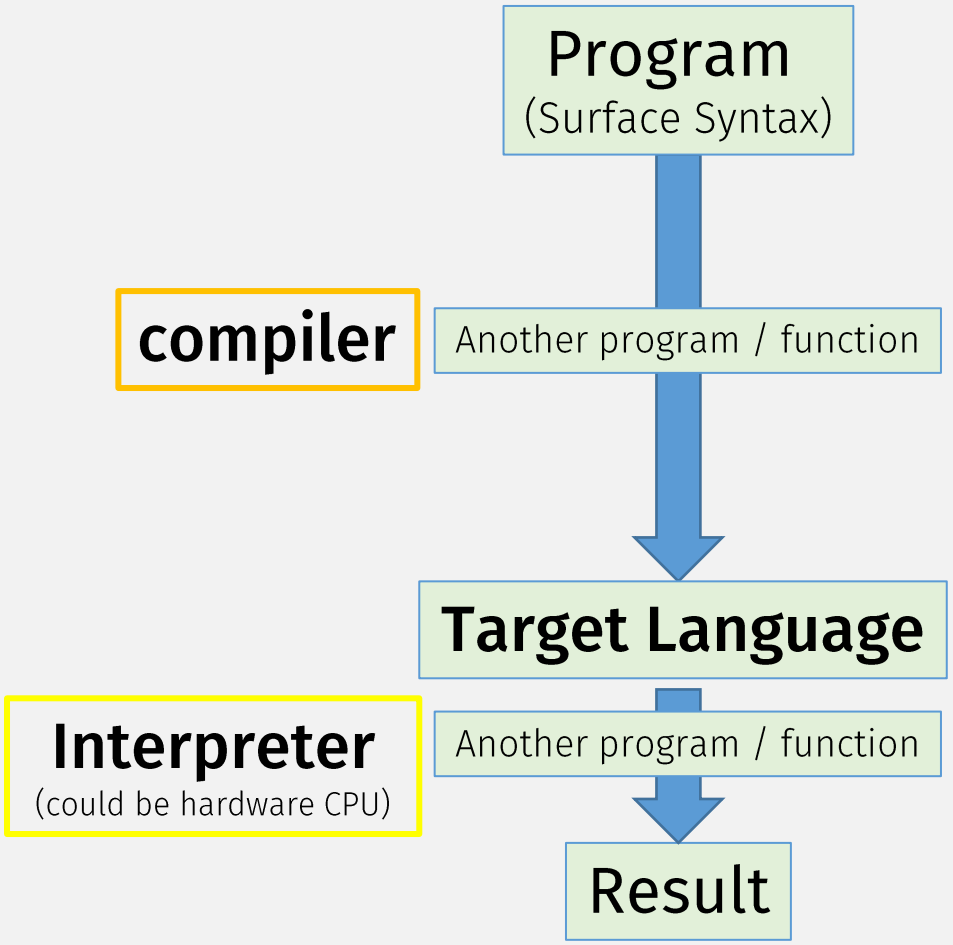
"How to Write a Programming Language" https://accu.org/journals/overload/26/146/balaam_2532/

Last Time

Giving Meaning to, i.e., Running, Programs



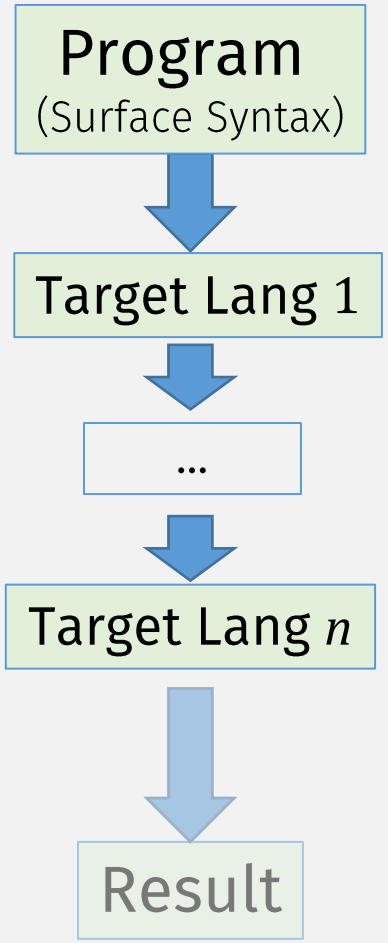
Last Time



More commonly, a **high-level program** is first **compiled** to a lower-level **target language** (and then **interpreted**)

Last Time

compiler



Compilers often have <u>multiple steps</u>

Parsing

Program that does this is called
Parser
(compiler's first step)

Program
(Surface Syntax)

Program
(tree)

...

Target Lang n

Result

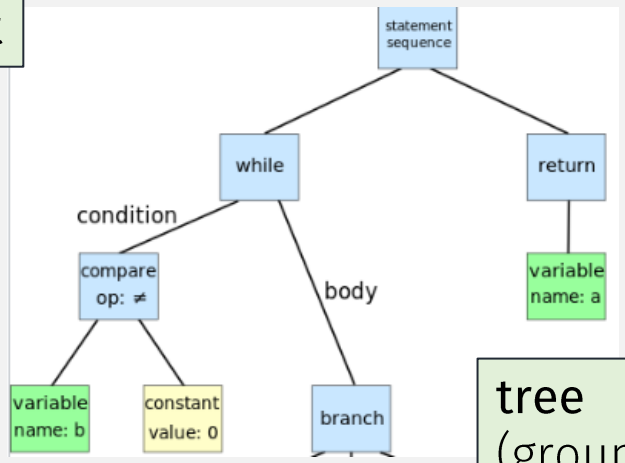
input

```
while b ≠ 0:
    if a > b:
        a := a - b
    else:
        b := b - a
return a
```

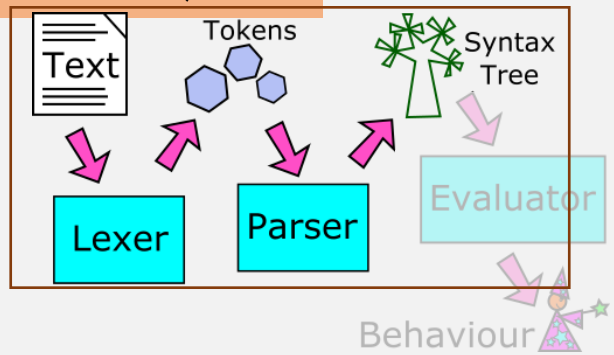
Typically, **string of chars**
(not a tree, not grouped by features)

Potentially really complicated
(take a compiler's course)

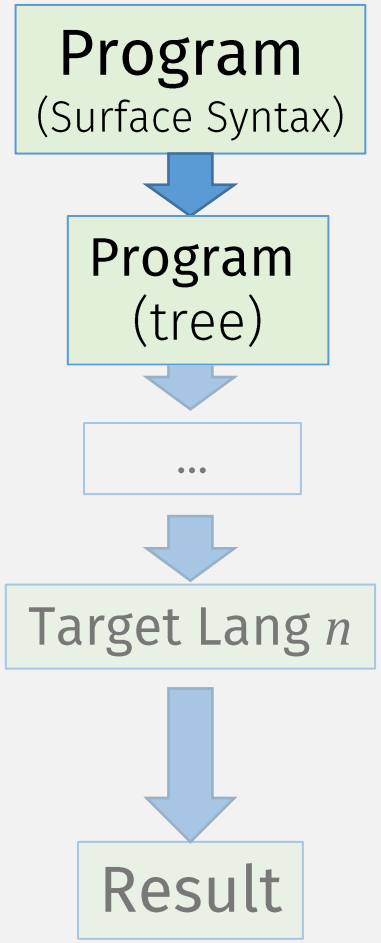
output



tree
(grouped by features)



Parsing – This Course




```
(* (+ 1 2)
   (- 4 3)
   (/ 10 5))
```

input

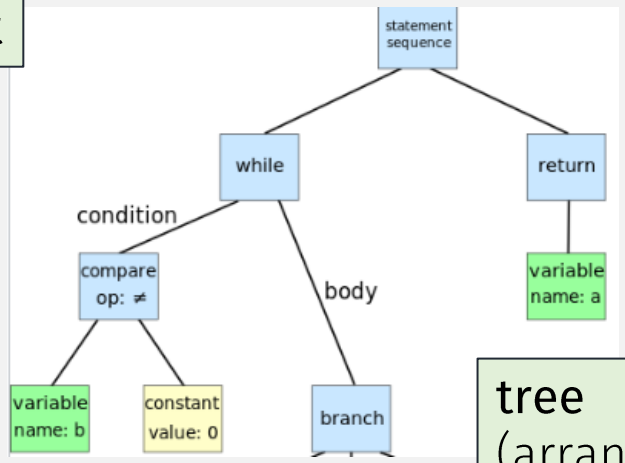
(very similar!)

output

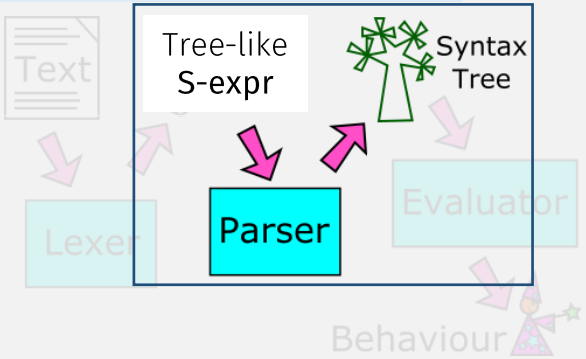
This course: **S-expressions** (already tree-like and grouped by features)

Data definition?

Much simpler
(save hard stuff for compilers course)



tree
(arranged by features)



```
;; A Program (Ssexpr) is a:
;; - Number
;; - `(+ ,Program ,Program)
;; - `(× ,Program ,Program)
```

```
(* (+ 1 2)
   (- 4 3)
   (/ 10 5))
```

This course: **S-expressions** (already tree-like and grouped by features)

Data definition?

```
(define (prog-fn p)
  (match p
    [(? number?) … ]
    [`(+ ,x ,y)
      … (prog-fn x) … (prog-fn y) … ]
    [`(× ,x ,y)
      … (prog-fn x) … (prog-fn y) … ]))
```

TEMPLATE?

```
;; A Program (
;; - Number
;; - `(+ ,Program ,Program)
;; - `(× ,
```

Program
(Surface Syntax)

Abstract Syntax
(Program) **Tree (AST)**

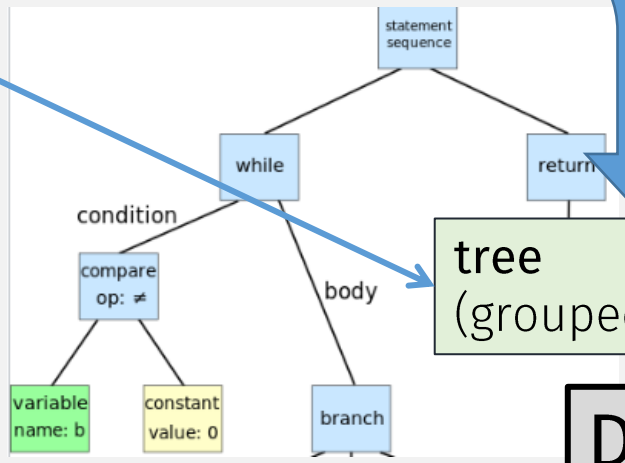
...

Target Lang n

Result

```
(* (+ 1 2)
   (- 4 3)
   (/ 10 5))
```

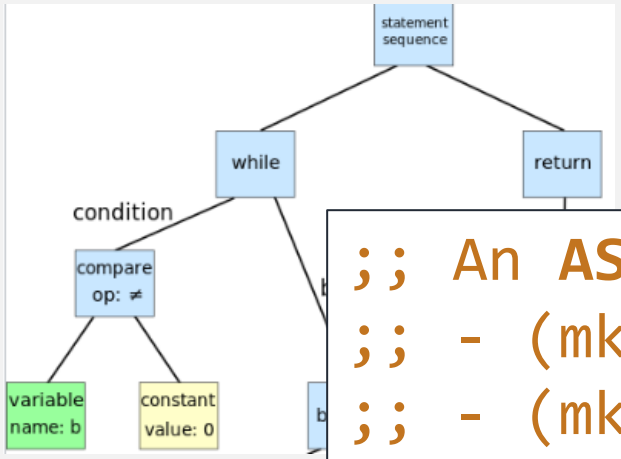
This course: **S-expressions** (already tree-like and grouped by features)



tree
(grouped by features)

Data definition?

Abstract Syntax
(Program) **Tree (AST)**




```
;; An AST is one of:
;; - (mk-num Number)
;; - (mk-add AST AST)
;; - (mk-mul AST AST)
(struct AST [])
(struct num AST [val])
(struct add AST [lft rgt])
(struct mul AST [lft rgt])
```

Program that does this is called

Parser

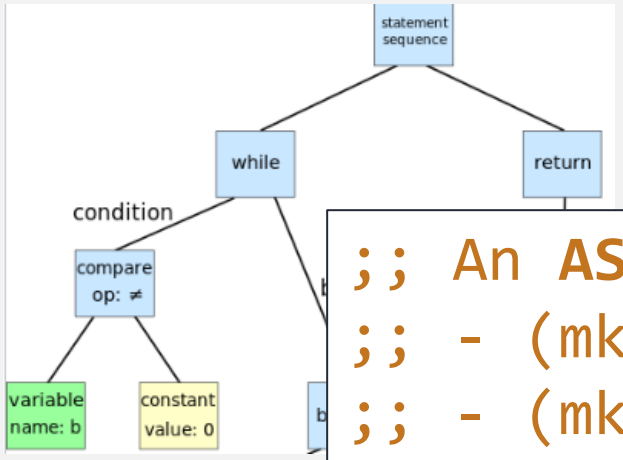
(compiler's first step)

Program

(Surface Syntax)

Abstract Syntax

(Program) **Tree (AST)**




```
;; An AST is one of:
;; - (mk-num Number)
;; - (mk-add AST AST)
;; - (mk-mul AST AST)
(struct AST [])
(struct num AST [val])
(struct add AST [lft rgt])
(struct mul AST [lft rgt])
```

```
;; A Program (Ssexpr) is a:
;; - Number
;; - `(+ ,Program ,Program)
;; - `(× ,Program ,Program)
```

```
;; parse: Program -> AST
;; Converts a Program to an AST
(define (parse p)
  (match p
    [(? number?) … ]
    [`(+ ,x ,y)
      … (parse x) … (parse y) … ]
    [`(× ,x ,y)
      … (parse x) … (parse y) … ]))
```

TEMPLATE

```
;; An AST is one of:
;; - (mk-num Number)
;; - (mk-add AST AST)
;; - (mk-mul AST AST)
(struct AST [])
(struct num AST [val])
(struct add AST [lft rgt])
(struct mul AST [lft rgt])
```

```
;; A Program (Ssexpr) is a:
;; - Number
;; - `(+ ,Program ,Program)
;; - `(× ,Program ,Program)
```

```
;; parse: Program -> AST
;; Converts a Program to an AST
(define (parse p)
  (match p
    [(? number?) (mk-num p)]
    [`(+ ,x ,y)
      … (parse x) … (parse y) … ]
    [`(× ,x ,y)
      … (parse x) … (parse y) … ]))
```

```
;; An AST is one of:
;; - (mk-num Number)
;; - (mk-add AST AST)
;; - (mk-mul AST AST)
(struct AST [])
(struct num AST [val])
(struct add AST [lft rgt])
(struct mul AST [lft rgt])
```

```
;; A Program (Ssexpr) is a:
;; - Number
;; - `(+ ,Program ,Program)
;; - `(× ,Program ,Program)
```

```
;; parse: Program -> AST
;; Converts a Program to an AST
(define (parse p)
  (match p
    [(? number?) (mk-num p)]
    [`(+ ,x ,y)
     (mk-add (parse x) (parse y))]
    [`(× ,x ,y)
     … (parse x) … (parse y) … ]))
```

```
;; An AST is one of:
;; - (mk-num Number)
;; - (mk-add AST AST)
;; - (mk-mul AST AST)
(struct AST [])
(struct num AST [val])
(struct add AST [lft rgt])
(struct mul AST [lft rgt])
```

```
;; A Program (Ssexpr) is a:
;; - Number
;; - `(+ ,Program ,Program)
;; - `(× ,Program ,Program)
```

```
;; parse: Program -> AST
;; Converts a Program to an AST
(define (parse p)
  (match p
    [(? number?) (mk-num p)]
    [`(+ ,x ,y)
     (mk-add (parse x) (parse y))]
    [`(× ,x ,y)
     (mk-mul (parse x) (parse y))]))
```

```
;; An AST is one of:
;; - (mk-num Number)
;; - (mk-add AST AST)
;; - (mk-mul AST AST)
(struct AST [])
(struct num AST [val])
(struct add AST [lft rgt])
(struct mul AST [lft rgt])
```

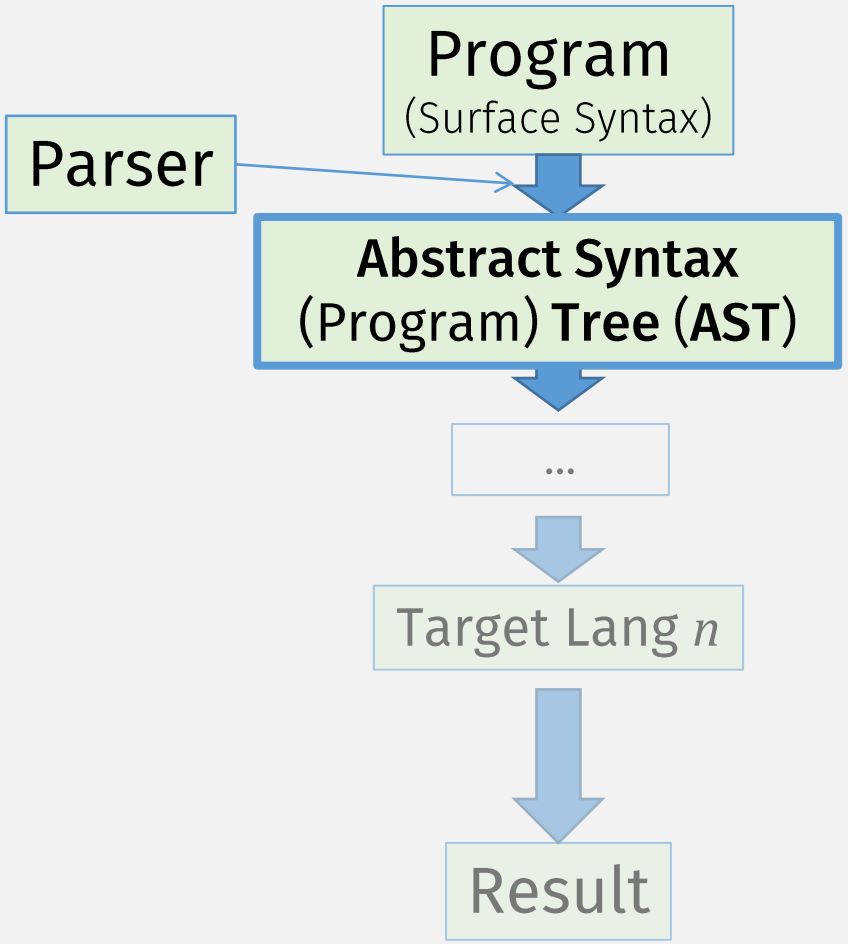
```
;; A Program (Ssexpr) is a:
;; - Number
;; - `(+ ,Program ,Program)
;; - `(× ,Program ,Program)
```

```
;; parse: Program -> AST
;; Converts a Program to an AST
(define (parse p)
  (match p
    [(? number?) (mk-num p)]
    [`(+ ,x ,y)
      (mk-add (parse x) (parse y))]
    [`(× ,x ,y)
      (mk-mul (parse x) (parse y))]))
```

TEMPLATE MAKES THIS EASY!

```
;; An AST is one of:
;; - (mk-num Number)
;; - (mk-add AST AST)
;; - (mk-mul AST AST)
(struct AST [])
(struct num AST [val])
(struct add AST [lft rgt])
(struct mul AST [lft rgt])
```

Previously

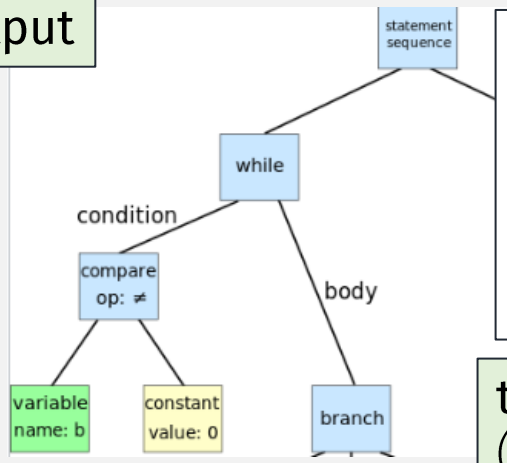


input

```
(* (+ 1 2)
   (- 4 3)
   (/ 10 5))
```

```
;; A Program (Ssexpr) is a:
;; - Number
;; - `(+ ,Program ,Program)
;; - `(× ,Program ,Program)
```

output




```
;; An AST is one of:
;; - (mk-num Number)
;; - (mk-add AST AST)
;; - (mk-mul AST AST)
```

tree
(grouped by features)

Program
(Surface Syntax)

Abstract Syntax
(Program) **Tree (AST)**

Interpreter
(could be hardware CPU)

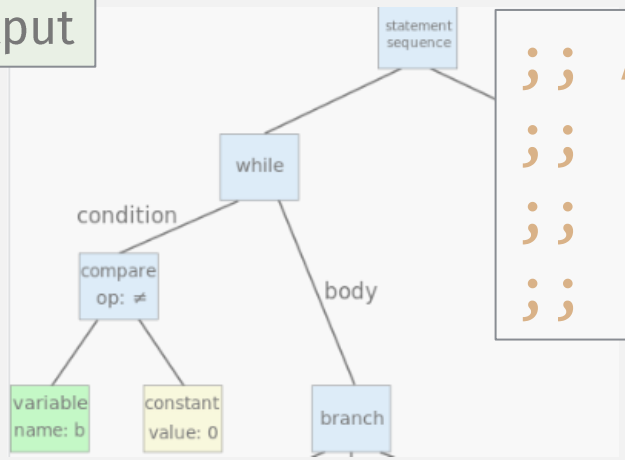
Another program / function

Result

input

```
(* (+ 1 2)
   (- 4 3)
   (/ 10 5))
```

output

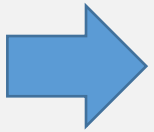



```
;; An AST is one of:
;; - (mk-num Number)
;; - (mk-add AST AST)
;; - (mk-mul AST AST)
```

```
;; A Result is a:
;; - ???
```

Data definition?

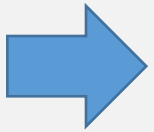
```
;; An AST is one of:
;; - (mk-num Number)
;; - (mk-add AST AST)
;; - (mk-mul AST AST)
```



```
;; A Result is a:
;; - Number
```

```
;; run: AST -> Result
;; Computes the Result of running the given program AST
```

```
;; An AST is one of:
(struct num [val])
(struct add [lft rgt])
(struct mul [lft rgt])
```



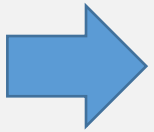
```
;; A Result is a:
;; - Number
```

```
;; run: AST -> Result
;; Computes the Result of running the given program AST
```

TEMPLATE

```
(define (run p)
  (match p
    [(num n) … ]
    [(add x y)  … (run x) …
                … (run y) … ]
    [(mul x y)  … (run x) …
                … (run y) … ])
```

```
;; An AST is one of:
(struct num [val])
(struct add [lft rgt])
(struct mul [lft rgt])
```



```
;; A Result is a:
;; - Number
```

```
;; run: AST -> Result
;; Computes the Result of running the given program AST
(define (run p)
  (match p
    [(num n) n]
    [(add x y) … (run x) …
               … (run y) … ]
    [(mul x y) … (run x) …
               … (run y) … ])
```

How to combine Results?

```
;; An AST is one of:
(struct num [val])
(struct add [lft rgt])
(struct mul [lft rgt])
```

```
;; A Result is a:
;; - Number
```

```
;; run: AST -> Result
;; Computes the Result of running the given program AST
(define (run p)
  (match p
    [(num n) n]
    [(add x y) (+ (run x)
                  (run y))]
    [(mul x y)  … (run x) …
                … (run y) … ])
```

Racket + gives semantics to our new language "+" operator

How to combine?

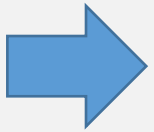
```
;; An AST is one of:
(struct num [val])
(struct add [lft rgt])
(struct mul [lft rgt])
```

```
;; A Result is a:
;; - Number
```

```
;; run: AST -> Result
;; Computes the Result of running the given program AST
(define (run p)
  (match p
    [(num n) n]
    [(add x y) (+ (run x)
                  (run y))]
    [(mul x y) (* (run x)
                  (run y))])
```

Racket * gives semantics to our new language "×" operator

```
;; An AST is one of:
(struct num [val])
(struct add [lft rgt])
(struct mul [lft rgt])
```



```
;; A Result is a:
;; - Number
```

```
;; run: AST -> Result
;; Computes the Result of running the given program AST
(define (run p)
  (match p
    [(num n) n]
    [(add x y) (+ (run x)
                  (run y))]
    [(mul x y) (* (run x)
                  (run y))]))
```

TEMPLATE MAKES THIS EASY!

The “CS 450 LANG” Programming Language

cs450lang.rkt

```
;; parse : Program -> AST
(define (parse p) … )
;; run: AST -> Result
(define (run t) … )

(define (eval450 p) (compose run parse))

;; tells Racket to use eval450 when “running” code
;; also, implicitly inserts quote “constructor”
(define-macro (module expr ...)
  (eval450 ‘expr) ...))
```

```
((compose f g) x) = (f (g x))
```

cs450lang-prog.rkt

```
#lang s-exp “cs450lang.rkt”

(+ 1 2) ; => 3
```

A program! written in “CS450 LANG”!

“CS450 Lang” Demo

The "CS450 Lang + Booleans" PL

```
;; A Program is a:
;; - Number
;; - `(+ ,Program ,Program)
;; - `(× ,Program ,Program)
```

Both represent code!
(before it's "run")

```
;; An AST is one of:
;; - (mk-num Number)
;; - (mk-add AST AST)
;; - (mk-mul AST AST)
(struct num [val])
```

Racket (bool) value

compiler internal (tree) representation of the program code

```
;; An AST is one of:
;; - (mk-num Number)
;; - (mk-boo Boolean)
;; - (mk-add AST AST)
;; - (mk-mul AST AST)
(struct num [val])
(struct boo [val])
(struct add [lft rgt])
(struct mul [lft rgt])
```

Code that the programmer writes

```
;; A Program is a:
;; - Number
;; - 450Bool
;; - `(+ ,Program ,Program)
;; - `(× ,Program ,Program)
```

```
;; A 450Bool is either:
;; - '450true
;; - '450false
```

Parsing "CS450 Lang + Bools" Programs

```
;; parse: Program -> AST
;; Converts a "450Lang" S-expression Program to AST
```

```
(define (parse p)
  (match p
   [(? number?) (mk-num p)]
   [(? 450bool?) (mk-boo ??? p)]
   [`(+ ,x ,y) (mk-add (parse x) (parse y))]
   [`(- ,x ,y) (mk-mul (parse x) (parse y))]))
```

```
;; A Program is a:
;; - Number
;; - 450Bool
;; - `(+ ,Program ,Program)
;; - `(× ,Program ,Program)
```

```
An AST is one of:
- (mk-num Number)
- (mk-boo Boolean)
;; - (mk-add AST AST)
;; - (mk-mul AST AST)
(struct num [val])
(struct boo [val])
(struct add [lft rgt])
(struct mul [lft rgt])
```

Parsing "CS450 Lang + Bools" Programs

```
;; parse: Program -> AST
;; Converts a "450Lang" S-expression Program to AST
```

```
(define (parse p)
  (match p
   [(? number?) (mk-num p)]
   [(? 450bool?) (mk-boo (450bool->bool p))]
   [`(+ ,x ,y) (mk-add (parse x) (parse y))]
   [`(- ,x ,y) (mk-mul (parse x) (parse y))]))
```

(could you write this function? Just follow the template!)

```
;; A Program is a:
;; - Number
;; - 450Bool
;; - `(+ ,Program ,Program)
;; - `(× ,Program ,Program)
```

```
An AST is one of:
- (mk-num Number)
- (mk-boo Boolean)
;; - (mk-add AST AST)
;; - (mk-mul AST AST)
(struct num [val])
(struct boo [val])
(struct add [lft rgt])
(struct mul [lft rgt])
```

Running “CS450 Lang + Bools” Programs

```
;; run: AST -> Result
;; computes the result of given program AST
```

```
;; A Result is a:
;; - Number
;; - Boolean
```

RACKET values

```
;; An AST is one of:
;;    …
;; - (mk-boo Boolean)
(struct num [val])
(struct boo [val])
(struct add [lft rgt])
(struct mul [lft rgt])
```

```
(define (run p)
  (match p
    [(num n) n]
    [(boo b) b]
    [(add x y) (???? (run x) (run y))]
    [(mul x y) (???? (run x) (run y))]))
```

Running “CS450 Lang + Bools” Programs

```
;; run: AST -> Result
;; computes the result of given program AST
```

```
;; A Result is a:
;; - Number
;; - Boolean
```

What should happen when two bools are added???

e.g., What is the “meaning” of `(+ 450true 450false)`

```
(define (run p)
  (match p
    [(num n) n]
    [(boo b) b]
    [(add x y) (???? (run x) (run y))]
    [(mul x y) (???? (run x) (run y))]))
```

Running “CS450 Lang + Bools” Programs

```
;; run: AST -> Result
;; computes the result of given program AST
```

```
;; A Result is a:
;; - Number
;; - Boolean
```

What should happen when two bools are added???

e.g., What is the “meaning” of `(+ 450true 450false)`

```
(define (run p)
  (match p
    [(num n) n]
    [(boo b) b]
    [(add x y) (450+ (run x) (run y))]
    [(mul x y) (???? (run x) (run y))]))
```

Running: “CS450LANG” Programs: “450+”

```
;; 450+: Result Result -> Result
;; “adds” two CS450Lang Result values together
```

```
;; A Result is either:
;; - Number
;; - Boolean
```

TEMPLATE

```
(define (450+ x y)
  (cond
    [(number? x) … ]
    [(boolean? x) … ]))
```

or

```
(define (450+ x y)
  (cond
    [(number? y) … ]
    [(boolean? y) … ]))
```

Two-Argument Templates

- Sometimes … a fn must **process <u>two</u> arguments simultaneously**
- **This template** should **combine templates of <u>both</u> args**
  - (This is only possible if the data defs are simple enough)

```
;; 450+: Result Result -> Result
;; "adds" two CS450Lang Result values together
```

(2-argument) TEMPLATE

```
(define (450+ x y)
  (cond
    [(and (number? x) (number? y))   … ]
    [(and (number? x) (boolean? y))  … ]
    [(and (boolean? x) (number? y))  … ]
    [(and (boolean? x) (boolean? y)) … ]))
```

(see why this is typically not recommended?)

```
;; 450+: Result Result -> Result
;; “adds” two CS450Lang Result values together
```

```
(define (450+ x y)
  (cond
    [(and (number? x) (number? y))   … ]
    [(and (number? x) (boolean? y))  … ]
    [(and (boolean? x) (number? y))  … ]
    [(and (boolean? x) (boolean? y)) … ]))
```

```
;; 450+: Result Result -> Result
;; “adds” two CS450Lang Result values together
```

```
(define (450+ x y)
  (cond
    [(and (number? x) (number? y)) (+ x y)]
    [(and (number? x) (boolean? y))  … ]
    [(and (boolean? x) (number? y))  … ]
    [(and (boolean? x) (boolean? y)) … ]))
```

Let’s look at other languages!

```
;; 450+: Result Result -> Result
;; “adds” two CS450Lang Result values together
```

```
(define (450+ x y)
  (cond
    [(and (number? x) (number? y)) (+ x y)]
    [(and (number? x) (boolean? y))  … ]
    [(and (boolean? x) (number? y))  … ]
    [(and (boolean? x) (boolean? y)) ???)]))
```

JavaScript Semantics Exploration: “plus”

- `repljs.com`

```
;; 450+: Result Result -> Result
;; “adds” two CS450Lang Result values together
;; (following js semantics!)
```

```
(define (450+ x y)
  (cond
    [(and (number? x) (number? y)) (+ x y)]
    [(and (number? x) (boolean? y)) … ]
    [(and (boolean? x) (number? y)) … ]
    [(and (boolean? x) (boolean? y)) ???)]))
```

```
;; 450+: Result Result -> Result
;; “adds” two CS450Lang Result values together
;; (following js semantics!)
```

```
(define (450+ x y)
  (cond
    [(and (number? x) (number? y)) (+ x y)]
    [(and (number? x) (boolean? y)) … ]
    [(and (boolean? x) (number? y)) … ]
    [(and (boolean? x) (boolean? y)) (+ (boo->num x) (boo->num y))])
```

```
;; 450+: Result Result -> Result
;; “adds” two CS450Lang Result values together
;; (following js semantics)
```

```
(define (450+ x y)
  (cond
    [(and (number? x) (number? y)) (+ x y)]
    [(and (number? x) (boolean? y)) ??? ]
    [(and (boolean? x) (number? y)) … ]
    [(and (boolean? x) (boolean? y)) (+ (boo->num x) (boo->num y))])
```

```
;; 450+: Result Result -> Result
;; “adds” two CS450Lang Result values together
;; (following js semantics)
```

```
(define (450+ x y)
  (cond
    [(and (number? x) (number? y)) (+ x y)]
    [(and (number? x) (boolean? y)) (+ x (boo->num y))]
    [(and (boolean? x) (number? y)) … ]
    [(and (boolean? x) (boolean? y)) (+ (boo->num x) (boo->num y))])
```

```
;; 450+: Result Result -> Result
;; “adds” two CS450Lang Result values together
;; (following js semantics)
```

```
(define (450+ x y)
  (cond
    [(and (number? x) (number? y)) (+ x y)]
    [(and (number? x) (boolean? y)) (+ x (boo->num y))]
    [(and (boolean? x) (number? y)) (+ (boo->num x) y)]
    [(and (boolean? x) (boolean? y)) (+ (boo->num x) (boo->num y))])
```

```
;; 450+: Result Result -> Result
;; “adds” two CS450Lang Result values together
;; (following js semantics)
```

```
(define (450+ x y)
  (cond
    [(and (number? x) (number? y))   (+ x              y)]
    [(and (number? x) (boolean? y))  (+ x              (boo->num y))]
    [(and (boolean? x) (number? y))  (+ (boo->num x) y)]
    [(and (boolean? x) (boolean? y)) (+ (boo->num x) (boo->num y))]))
```

(can any cond clauses be combined?)

```
;; 450+: Result Result -> Result
;; “adds” two CS450Lang Result values together
;; (following js semantics)
```

```
(define (450+ x y)
  (cond
    [(and (number? x) (number? y))   (+ (res->num x) (res->num y))]
    [(and (number? x) (boolean? y))  (+ (res->num x) (res->num y))]
    [(and (boolean? x) (number? y))  (+ (res->num x) (res->num y))]
    [(and (boolean? x) (boolean? y)) (+ (res->num x) (res->num y))])
```

(can any cond clauses be combined?)

```
;; 450+: Result Result -> Result
;; “adds” two CS450Lang Result values together
;; (following js semantics)
(define (450+ x y)
  (+ (res->num x) (res->num y)))
```

```
;; A Result is a:
;; - Number
;; - Boolean
```

```
;; res->num: Result -> Number
(define (res->num x)
  (cond
   [(number? x)  ... ]
   [(boolean? x) ... ]))
```

TEMPLATE

```
;; 450+: Result Result -> Result
;; “adds” two CS450Lang Result values together
;; (following js semantics)
(define (450+ x y)
  (+ (res->num x) (res->num y)))
```

```
;; res->num: Result -> Number
(define (res->num x)
  (cond
   [(number? x) x]
   [(boolean? x) (boo->num x)]))
```

Can we do the same thing for multiplication?

- Check repljs.com

```
;; 450times: Result Result -> Result
;; “multiplies” two 450Lang Result values
;; (following js semantics)
(define (450times x y)
  (* (res->num x) (res->num y)))
```

```
;; res->num: Result -> Number
(define (res->num x)
  (cond
   [(number? x) x]
   [(boolean? x) (boo->num x)]))
```

In-class Coding 4/10 (hw9): **put it all together!**

```
;; parse: Program -> AST
;; Parses “450 Lang” Program to AST
```

```
;; run: AST -> Result
;; Computes Result of running a program
```

```
;; An Program is one of:
;; - Number
;; - 450Bool
;; - `(+ ,Program ,Program)
;; - `(× ,Program ,Program)
```

Code that the programmer writes

```
;; An AST is one of:
;; - (mk-num Number)
;; - (mk-boo Boolean)
;; - (mk-add AST AST)
;; - (mk-mul AST AST)
(struct num [val])
(struct boo [val])
(struct add [lft rgt])
(struct mul [lft rgt])
```

Compiler internal code representation (before it’s run)

Use super struct?

```
;; A 450Bool is either
;; - ‘450true
;; - ‘450false
```

```
;; A Result is one of:
;; - Number
;; - Boolean
```

RACKET values

Result after running the code