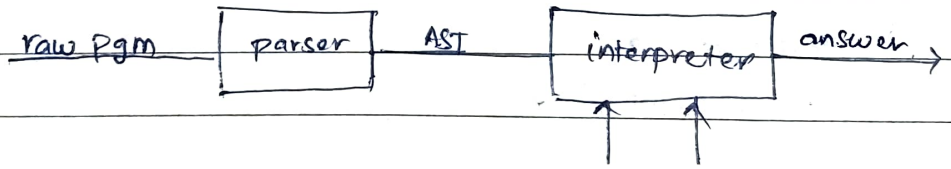
ARITHMETIC

→ next ALGEBRAIC

1. semantic domains
2. abstract syntax
3. interpreter & runtime
4. concrete syntax & parser.

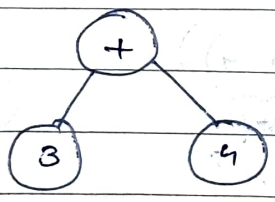
3 types of models to understand how programs run.

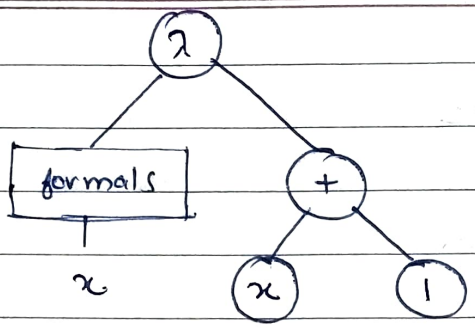
1. mathematical (operational semantics)
2. visual (AST annotations)
3. interpreters.
4. embeddings (libraries + syntactic extensions)



$e ::= n \mid +(e, e) \mid -(e, e)$

$+(3, 4)$





e::= n | e+e | e-e | e*e

(define-datatype ast ast?
  [num (n number?)]
  [plus (left ast?) (right ast?)]
  [minus (') (')]
  [mul (') (')])

(num 5)

~~→ num takes a~~

;;; num number? → ast?

(ast? (num 5)) ⇒ #t

mathematical representation

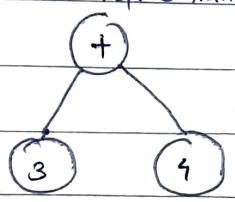
;;; 3+4

(plus (num 3) (num 4))

ast? ast? ast?

interpreter representation

visual representation



function will evaluate all arguments, then calls it.

keyword doesnt evaluate it.

Keywords so far:

define (if #t 5 ∞) ⇒ 5

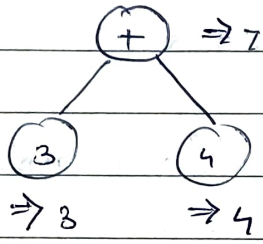
let

lambda (if #t 5 ∞) ⇒ ∞

if

cond

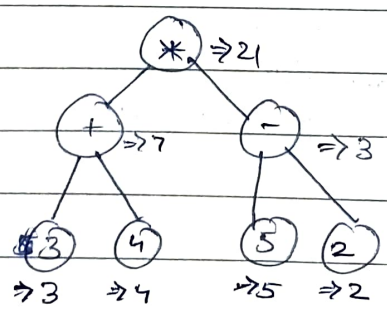
else



eval: ast? ⟶ ans?

ans? = number?

evaluation via annotation



interpreter will be written recursively.

ast? ⟶ answer?

```
(define eval-ast
  (λ (a)
    (cases ast a
      [num (n) n]
      [plus (left right)
```

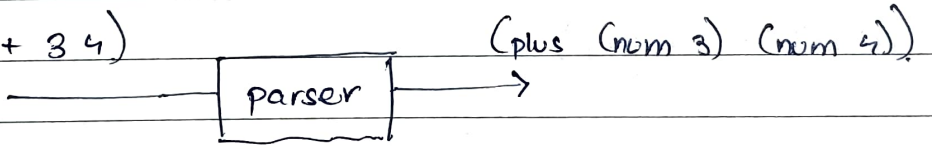
```
(let ([la (eval-ast left)]
      [lr (eval-ast right)])
  (+ la lr)))]
```

CONCRETE SYNTAX

⟨exp⟩ ::= ⟨num⟩ | (+ ⟨exp⟩ ⟨exp⟩)
| (- ⟨exp⟩ ⟨exp⟩)

⟨exp⟩ ::= ⟨num⟩ | (⟨op⟩ ⟨exp⟩ ⟨exp⟩)

⟨op⟩ ::= + | - | *



(any/c)?

;;; parse :: s-exp ⟶ ast?

(define parse

```
'(* (+ 3 4)
    (- 5 2))
```

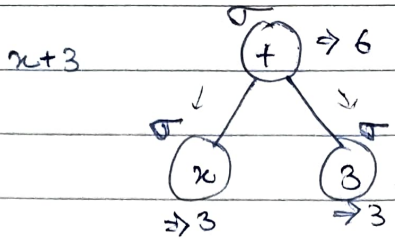
(provide (all-defined-out))

```
(define *keywords* '(+ - *))
(define parse
  λ (x)
    (cond
      [(number? x) (num x)]
      [ (list? x)
        (= (length x) 3)
        (memq (first x) *keywords* )
        (let ([left (parse (second x))]
              [right (parse (third-x))]
              [con (keyword-> constructor ...)]
            (con left right))]
      [else (error 'parse "invalid input ~a" x)]...


(define eval-ast
  (λ (a)
    (cases ast a
      [num (n) n]
      [plus (left right)
            (let ([n1 (eval-ast left)]
                  [n2 ....          )]
              (+ n1 n2)
      .....


(define go
  (λ (x) (eval-ast (parse x))))
```

$e ::= n \mid + e\ e \mid - e\ e \mid * e\ e \mid x$



suppose.

$\sigma = \{x:3, y:4\}$

annotations.

- answers
- environments