WELL FOUNDED INDUCTION

(weak induction)

$$\frac{\forall a \in A,\ (\forall b \in A\ \ a \to b \Rightarrow P(b)) \Rightarrow P(a)}{\forall a \in A\ \ P(a)}$$

$\mathbb{N}$

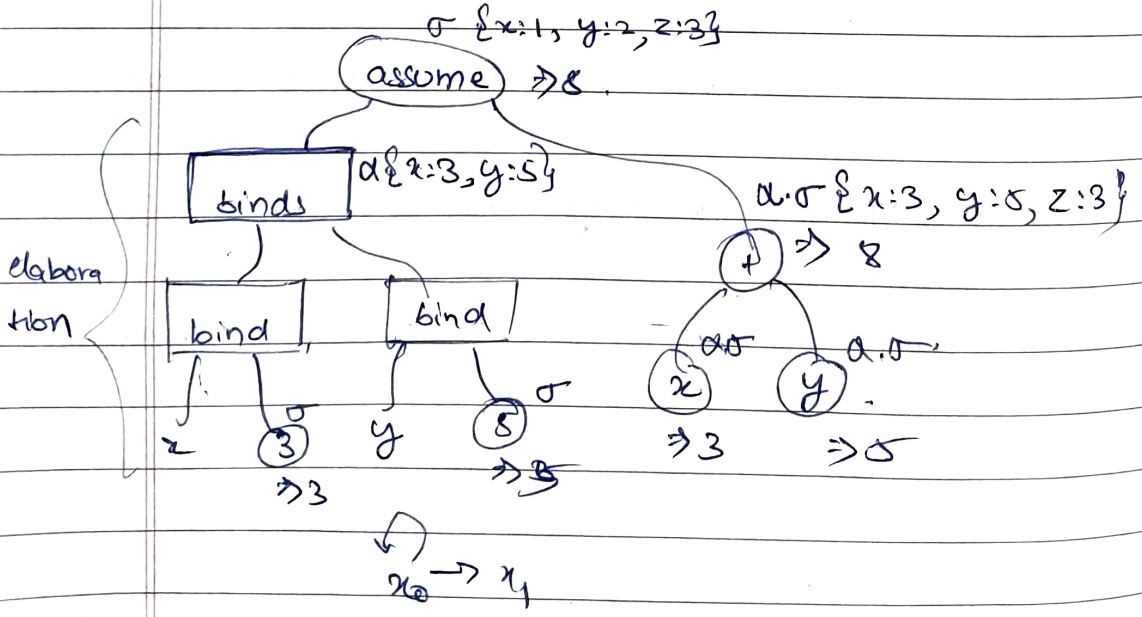
$$\frac{\forall a \in A,\ (\forall b \in A\ \ a \xrightarrow{+} b \Rightarrow P(b)) \Rightarrow P(a)}{\forall a \in A\ \ P(a).}$$

$a \in A$ $\qquad a \Rightarrow b$, $a \Rightarrow c$

$\Rightarrow b \downarrow c$

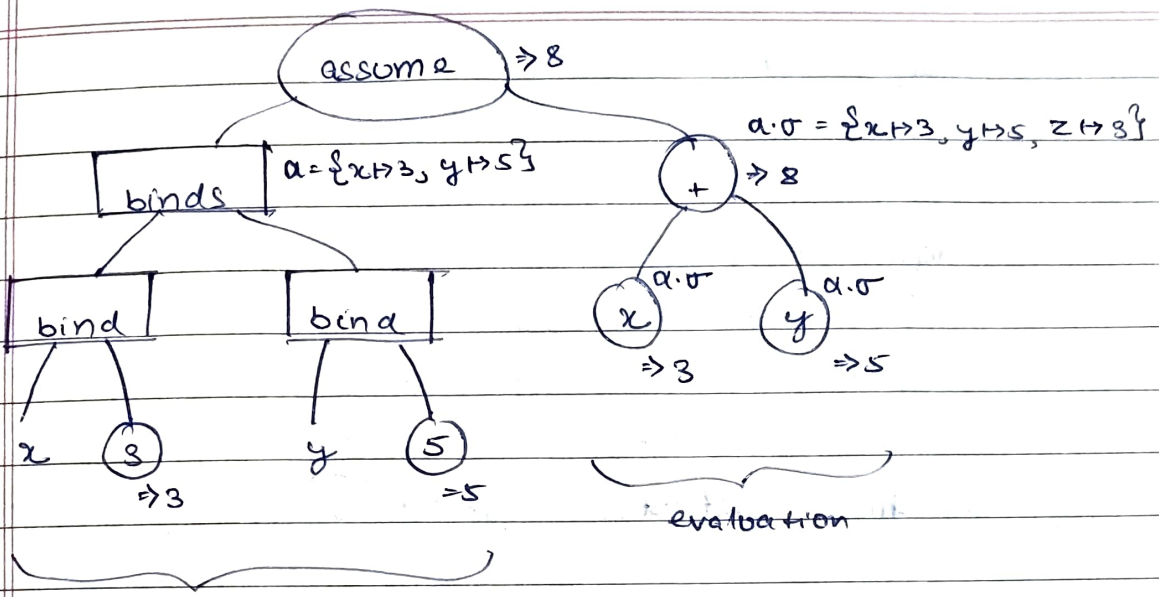
$b = c.$

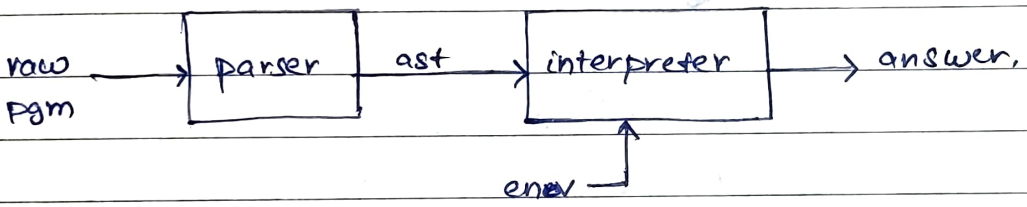
elaboration vs evaluation.



classmate
Date 21.9.2019
Page 6

$\sigma = \{x \mapsto 1, y \mapsto 2, z \mapsto 3\}$





global, lexical } scope.

e ::= n | b | x | (op e e) | (assume [x e] ... e)

op ::= + | - | * | /

answer = expressible value + exception

expressible - result of an expression
denotable - value can be bound to identifier
storable - value may be stored in memory.

```
#lang racket
(require eopl)

(define-datatype ast ast?
  [num (value number?)]
  [add (left ast?) (right ast?)]
  [sub (left ast?) (right ast?)]
  [mul (left ast?) (right ast?)])

(define eval
  (lambda (ast tree)
    (cases ast tree
      (num (value) value)
      (add (left right) (+ (eval left) (eval right)))
      (sub (left right) (- (eval left) (eval right)))
      (mul (left right) (* (eval left) (eval right))))))

(define keywords '(+ - *))
(define constructors (list add sub mul))

(define parse
  (lambda (sexp)
    (cond
      [(number? sexp) (num sexp)]
      [(and
          (list? sexp)
          (= 3 (length sexp))
          (memq (first sexp) keywords))
```

```
(let*
  ([keyword (first sexp)]
   [left (second sexp)]
   [right (third sexp)]
   [constructor (listref constructors
                  (index-of keywords keyword))])
  (constructor (parse left) (parse right))])
 [else (error 'parse "invalid input ~a" sexp)])))

(define go
  (lambda (sexp)
    (eval (parse sexp))))

(define foldl
  (lambda (f init lst)
    (if (null? lst)
        init
        (foldl f (f init (car lst)) (cdr lst)))))

(define foldr
  (lambda (f init lst)
    (if (null? lst)
        init
        (f (car lst) (foldr f init (cdr lst))))))

(define map
  (lambda (f lst)
    (if (null? lst)
        '()
        (cons (f (car lst)) (map f (cdr lst))))))
```