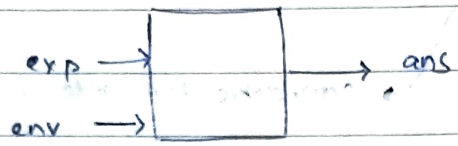
classmate
Date 8.8.2019
Page 1

THEORIES

HW

prose will get marks (book)

1. binding (define, let, λ)
2. currying
3. map, reduce, filter & functional programming.
4. abstract syntax trees.



env = id → value

ans = value + error + ∞ (non-termination)

binding in env.

(+ x 2) where {x ↦ 3}

⇒ (+ 3 2)

⇒ 5

(+ x y) where {x ↦ 3}

⇒ (+ 3 y)

↑ error

using define → global bindings

let → local bindings

lambda → parameterized expressions

> (define f (λ(x) (+ 1 x)))

> (f 3) where f = (λ(x) (+ 1 x))

σ

$\Rightarrow ((\lambda(x)\ (+\ 1\ x))\ 3)$ wrt σ_1

$\overset{\beta}{\Rightarrow} (+\ 1\ x)$ wrt $\sigma_2 \cdot \sigma_1$ $\sigma_2 = \{x \mapsto 3\}$

↳ composition

$\Rightarrow (+\ 1\ 3)$

$\Rightarrow 4$

$\sigma_2 \cdot \sigma_1 = \{x \mapsto 3\}\ \{f \mapsto \ldots\}$

$= \{x \mapsto 3,\ f \mapsto \ldots\}$

$f \mapsto (\lambda(x)\ (+\ 1\ x))$
$x \mapsto 5$ $\Big\}\ \sigma_1$ partial function

(yield → partial function.

$\sigma_2 \cdot \sigma_1 = \{x \mapsto 3\} \cdot \{f \mapsto \ldots, x \mapsto 5\}$

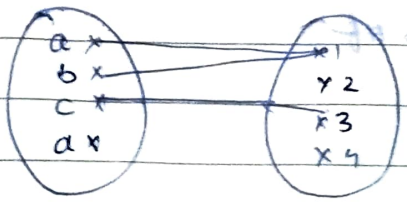
$= \{x \mapsto 3,\ f \mapsto \ldots\}$

σ_1 : env

σ_2 : env

$\sigma_1 \cdot \sigma_2$: env

domain — id → value — co domain



domain of definition (dod)

$$dod(\sigma_2 \cdot \sigma_1) = dod(\sigma_2) \cup dod(\sigma_1)$$

$$(\sigma_2 \cdot \sigma_1)(z) = \begin{cases} z \notin dod(\sigma_2 \cdot \sigma_1) & \text{undefined} \\ z \in dod(\sigma_2) \cap dod(\sigma_1) & \sigma_2(z) \\ z \in dod(\sigma_2) & \sigma_2(z) \\ \text{else} & \sigma_1(z) \end{cases}$$

an environment is a mapping, a functions.

env: i'd → value

ans: value + err + non-termination.

what is a program
how does it run

A x B ⟶ C

```
> (define add                    > (add 3 4)
      (λ(a b)                    7
        (+ a b)))
```

(number? 5) (number? 'a) =

⇒ #t ⇒ #f

↳ space / set.

$A \times B \longrightarrow C \quad \cong \quad A \longrightarrow (B \longrightarrow C)$

$|C|^{|A|\cdot|B|} \quad \stackrel{?}{=} \quad |(B \to C)|^{|A|}$

$(|C|^{|B|})^{|A|} = |C|^{|A|\cdot|B|}$

```
; addc will take (:) number? ⟶ (number? ⟶ number?)
(define addc
  (λ(a)
    (λ(b)
      (+ a b))))

> (addc 3)
#<prodedure>

> (define add3 (addc 3))

> (add3 4)
7
```

(derivative, OOP) integration (Condefinite)

D: $(\mathbb{R} \to \mathbb{R}) \to (\mathbb{R} \to \mathbb{R})$

Haskell Curry → name of logician

FUNCTIONAL PROGRAM

COMBINATORS

(λ(x) — bound
(+ x y) — free

Book

wrt. the expression.

```
(+ x ((λ (x)
        (+ 2 x)))
    3))
```

★ combinators → expression that has no free occurrences of any identifier.

```
> (list 3 1 4)
'(3 1 4)

> (require racket/list)
> (symbol? 'a)
a:

> (symbol? a)
error.

> (number? 5)
#t
```

'5 → 5

inheritance → String with overlearning, memory allocation, number class.

```
>'|5|
>'|5|

> '|hello how are you?|
: ...

> (first '(a b c))
'a

> (first 'c)
error

>'(x 4 6)
'(x 4 6)

> (first '(x 4 6))
'x
```