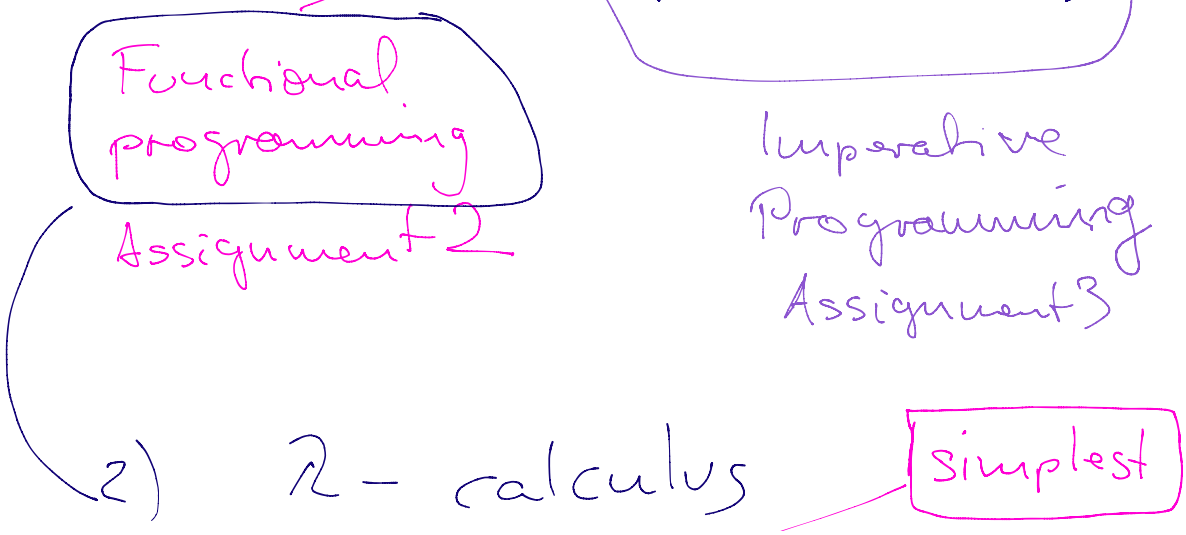
How to build your programming language

1) calculator
2) λ-calculus
3) rewriting theory

1) interpreter for arithmetic

missing: - variables
- functions
(- statements)



is the smallest possible programming language

just: variables
functions

3) Semantics (meaning)

of λ-calculus can be given in terms of rewriting

λ-calculus is an ARS $(A, \to)$

A: λ-expressions

$\to$: α-rule (renaming variable)
β-rule (substitution)

Aim:

- To understand an

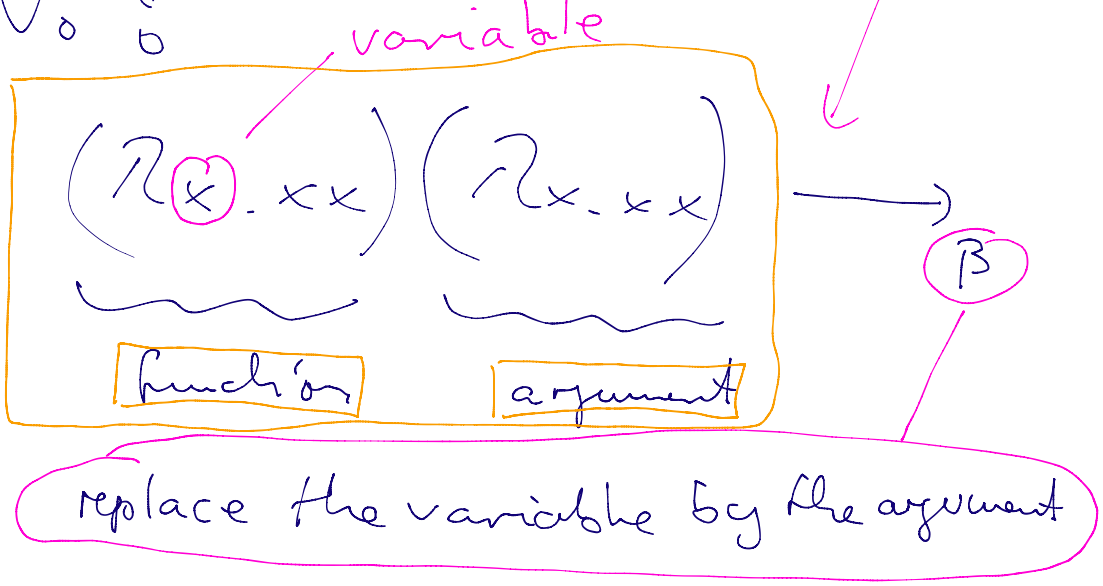
Interpreter for λ-calculus

- To extend λ-calculus with new features to obtain a small functional programming language
  - if-then-else
  - recursion
  - natural numbers
  - lists

Is λ-calculus terminating?

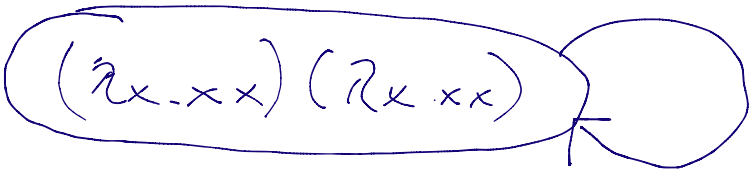
λ-calc is +RS

No.



$$(\lambda x.xx)(\lambda x.xx) \longrightarrow (\lambda x.xx)(\lambda x.xx)$$

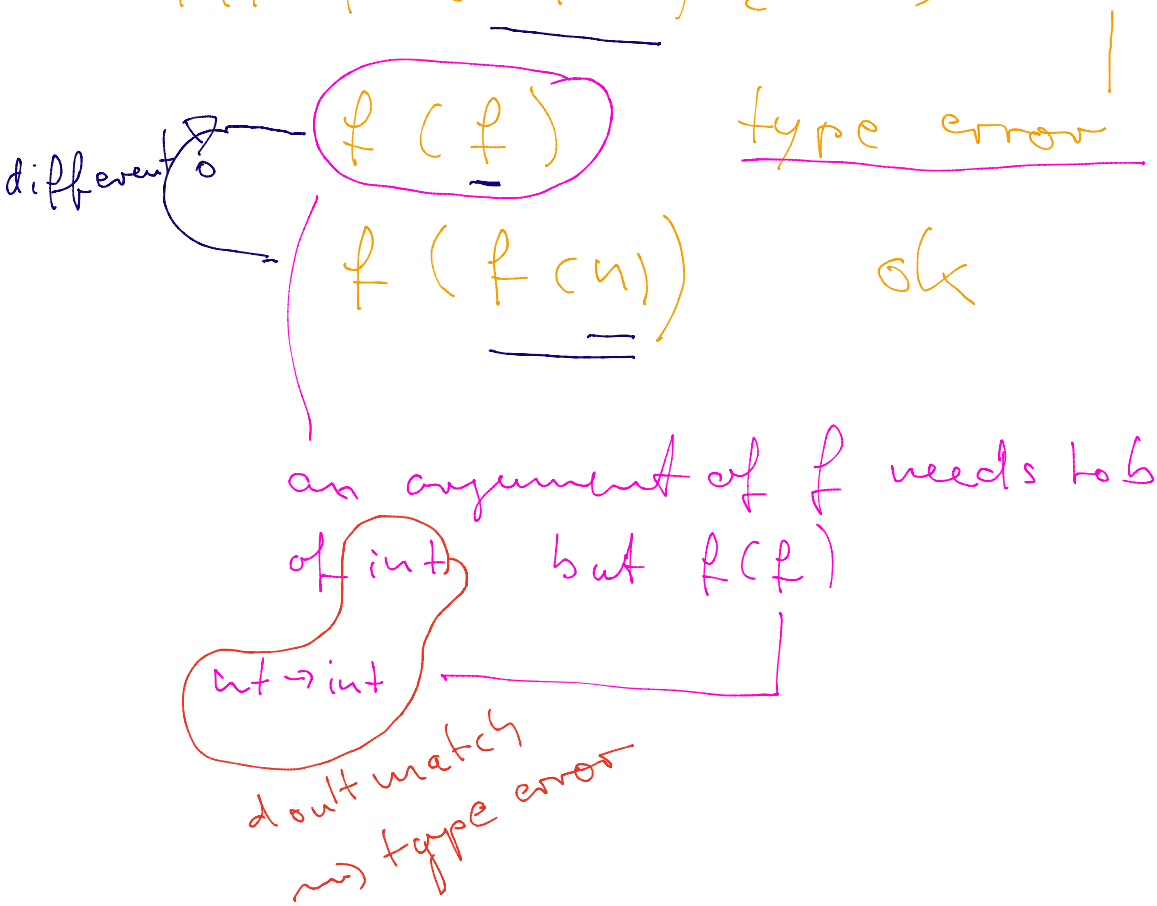
hence λ-calculus is not terminating



normal

f : int → int

int f (int x) {...}



- λ-calc is not terminating

- the example cannot be typed it only in the untyped λ-calculus
- typed λ-calculus is terminating
- typed λ-calculus is not Turing complete
- ~~typed~~ λ-calculus + recursion is not terminating (and Turing complete)

Going back to calculator:

```
eval :: Exp  ⟶  Int
```

Programs ↑ Exp, Values ↑ Int

```
execCBN :: Exp ⟶ Exp
{
```

call by name (lazy)

call by value

$(\lambda x.\, x * 2)\ (2 + 3)$

CBN $\rightarrow (2+3) * 2$

CBV $\rightarrow (\lambda x.\, x * 2)\ 5$

$\rightarrow 5 * 2$

β rule

eval e_1 e_2 = subst i e_2 e_3

$(\lambda i.\, e3)\ e_2$