

# 编程基础 III

## Lambda演算

刘 钦

# Reference

- <https://github.com/txyyss/Lambda-Calculus/releases>
- <http://cgnail.github.io/academic/lambda-1/>

# 知识点

- 定义（原子、应用、抽象）
- 公理（alpha变换、beta规约）
- 算术计算与逻辑谓词
- 递归与Y Combinator
- 有序对

# Outline

- Lambda演算定义
- 算术计算与逻辑谓词
- 递归与Y Combinator
- 有序对

# Outline

- Lambda演算定义
- 算术计算与逻辑谓词
- 递归与Y Combinator
- 有序对

# Lamdar 演算

- 形式化地，我们从一个标识符（identifier）的可数无穷集合开始，比如{a, b, c, ..., x, y, z, x1, x2, ...}，则所有的lambda表达式可以通过下述以BNF范式表达的上下文无关文法描述：
  - $\langle \text{表达式} \rangle ::= \langle \text{标识符} \rangle$
  - $\langle \text{表达式} \rangle ::= (\lambda \langle \text{标识符} \rangle . \langle \text{表达式} \rangle)$
  - $\langle \text{表达式} \rangle ::= (\langle \text{表达式} \rangle \langle \text{表达式} \rangle)$
- 头两条规则用来生成函数，而第三条描述了函数是如何作用在参数上的。
- 例：
  - $(\lambda x. 2x)$
  - $(\lambda x y. x+y) a b$  其实科里化（Currying）后变为 $((\lambda x. (\lambda y. (y+x))) a) b$  单参数的Larmdar演算

**定义 1. ( $\lambda$  项)** 假设我们有一个无穷的字符串集合, 里面的元素被称为变量(和程序语言中变量同, 这里就是指字符串本身)。那么  $\lambda$  项定义如下:

1. 所有的变量都是  $\lambda$  项(名为原子);
2. 若  $M$  和  $N$  是  $\lambda$  项, 那么  $(MN)$  也是  $\lambda$  项(名为应用)
3. 若  $M$  是  $\lambda$  项而  $\phi$  是一个变量, 那么  $(\lambda\phi.M)$  也是  $\lambda$  项(名为抽象)。

## 形式化定义

# lambda 项

示例 1. (一些  $\lambda$  项) 下面这些都是  $\lambda$  项:

$$\begin{aligned} &(\lambda x.(x\ y)) \\ &(((\lambda x.(\lambda y.(y\ x)))\ a)\ b) \end{aligned}$$
$$\begin{aligned} &(x(\lambda x.(\lambda x.x))) \\ &((\lambda y.y)\ (\lambda x.(x\ y))) \end{aligned}$$
$$\begin{aligned} &((((a\ b)\ c)\ d)\ e) \\ &(\lambda x.(y\ z)) \end{aligned}$$



# 符号约定 1 - 省略

**符号约定 1.** 本文中我们用大写英文字母表示任意  $\lambda$  项, 用除  $\lambda$  以外的小写希腊字母如  $\phi, \psi$  等表示任意  $\lambda$  项中的变量。

对于括号, 则有如下的省略规定:

1.  $\lambda$  项中最外层的括号可以省略, 如  $(\lambda x.x)$  可以省略表示为  $\lambda x.x$ ;
2. 左结合的应用型的  $\lambda$  项, 如  $((MN)P)Q$ , 括号可以省略, 表示为  $MNPQ$ ;
3. 抽象型的  $\lambda$  项  $(\lambda \phi.M)$  中,  $M$  最外层的括号可以省略, 如  $\lambda x.(yz)$  可以省略为  $\lambda x.yz$ 。

也就是说, 我们把省略形式视同定义 1 中的  $\lambda$  项。

**示例 2. (省略表示)** 下面给出了一些省略表示的  $\lambda$  项。

省略表示

$\lambda x.\lambda y.yx a b$

$(\lambda x.\lambda y.yx) a b$

$\lambda g.(\lambda x.g(x x)) \lambda x.g(x x)$

$\lambda x.\lambda y.a b \lambda z.z$

完整的  $\lambda$  项

$(\lambda x.(\lambda y.(((y x) a) b)))$

$(((\lambda x.(\lambda y.(y x))) a) b)$

$(\lambda g.((\lambda x.(g(x x))) (\lambda x.(g(x x)))))$

$(\lambda x.(\lambda y.((a b) (\lambda z.z))))$

定义 2. (语法全等) 我们用恒等号 “ $\equiv$ ” 表示两个  $\lambda$  项完全相同。换句话说

$$M \equiv N$$

表示  $M$  和  $N$  有完全相同的结构, 且对应位置上的变量也完全相同。这意味着若  $MN \equiv PQ$  则  $M \equiv P$  且  $N \equiv Q$ , 若  $\lambda\phi.M \equiv \lambda\psi.P$  则  $\phi \equiv \psi$  且  $M \equiv P$ 。

定义 3. (自由变量) 对一个  $\lambda$  项  $P$ , 我们可以定义  $P$  中自由变量的集合  $FV(P)$  如下:

1.  $FV(\phi) = \{\phi\}$
2.  $FV(\lambda\phi.M) = FV(M) \setminus \{\phi\}$
3.  $FV(MN) = FV(M) \cup FV(N)$

从第 2 可以看出抽象  $\lambda\phi.M$  中的变量  $\phi$  是要从  $M$  中被排除出自由变量这个集合的。若  $M$  中有  $\phi$ , 我们可以说它是被约束的。据此可以进一步定义约束变量集合。值得注意的是, 对同一个  $\lambda$  项来说, 这两个集合的交集未必为空。

示例 4. (自由变量)

$\lambda$ 项 $P$	自由变量集合 $FV(P)$
$\lambda x.\lambda y.xyab$	$\{a, b\}$
$abcd$	$\{a, b, c, d\}$
$xy\lambda y.\lambda x.x$	$\{x, y\}$

上面最后一个例子里, 最左边的  $x, y$  是自由变量, 而最右侧的  $x$  则是约束变量。若对  $\lambda$  项  $P$  有  $FV(P) = \emptyset$ , 则称  $P$  是封闭的, 这样的  $P$  又称为组合子。

## 自由变量和组合子

# 演算公理系统

- 置换公理( alpha 变换)
  - $\lambda xy.x+y \Rightarrow \lambda ab.a+b$
  - $\text{lambda } x \ y. \ x + y \Rightarrow \text{lambda } a \ b. \ a + b$
- 代入公理 (beta 规约)
  - $(\lambda xy. x+y) \ a \ b \Rightarrow a+b$
  - $\lambda y. (\lambda x. x+y \ a) \ b$
- 例如:
  - $(\lambda x. \lambda y. x - y) \ 7 \ 2$  与  $(\lambda y. 7 - y) \ 2$  与  $7 - 2$  是等价的



**定义 6.** ( $\alpha$  变换和  $\alpha$  等价) 设  $\lambda\phi.M$  出现在一个  $\lambda$  项  $P$  中, 且设  $\psi \notin \text{FV}(M)$ , 那么把  $\lambda\phi.M$  替换成

$$\lambda\psi.[\psi/\phi]M$$

的操作被称为  $P$  的  $\alpha$  变换。当且仅当若  $P$  经过有限步(包括零步) $\alpha$  变换后, 得到新的  $\lambda$  项  $Q$ , 则我们可以称  $P$  与  $Q$  是  $\alpha$  等价的, 又写作

$$P \equiv_{\alpha} Q$$

# alpha 变换

定义 7. ( $\beta$  规约) 形如

$$(\lambda\phi.M)N$$

的  $\lambda$  项被称为  $\beta$  可约式, 对应的项

$$[N/\phi]M$$

则称为  $\beta$  缩减项。当  $P$  中含有  $(\lambda\phi.M)N$  时, 我们可以把  $P$  中的  $(\lambda\phi.M)N$  整体替换成  $[N/\phi]M$ , 用  $R$  指称替换后得到的项, 那么我们说  $P$  被  $\beta$  缩减为  $R$ , 写做:

$$P \triangleright_{1\beta} R$$

当  $P$  经过有限步(包括零步)的  $\beta$  缩减后得到  $Q$ , 则称  $P$  被  $\beta$  规约到  $Q$ , 写做:

$$P \triangleright_{\beta} Q$$

# beta 规约

# 练习

- $(\lambda x. x (x y)) m$
- $(\lambda x. y) n$
- $(\lambda x. (\lambda y. y x) z) v$
- $(\lambda x. x x)(\lambda x. x x)$
- $(\lambda x. x x y)(\lambda x. x x y)$

# 定理

定理 1. 若  $M \equiv_{\alpha} M'$  且  $N \equiv_{\alpha} N'$ , 则  $[N/x]M \equiv_{\alpha} [N'/x]M'$

定理 2. (*Church-Rosser* 定理) 若  $P \triangleright_{\beta} M$  且  $P \triangleright_{\beta} N$ , 则存在一个  $\lambda$  项  $T$  使得

$$M \triangleright_{\beta} T \quad \text{且} \quad N \triangleright_{\beta} T.$$

定理 3. 若  $P$  有  $\beta$  范式, 则该范式在模  $\equiv_{\alpha}$  的意义下唯一; 也就是说若  $P$  有  $\beta$  范式  $M$  和  $N$ , 则  $M \equiv_{\alpha} N$ 。

定理 4. 对  $P$  的总是先  $\beta$  缩减最左侧最外侧的  $\beta$  可约式, 若这个过程能无限进行下去, 那么对  $P$  的所有任意顺序的规约都能无穷进行下去。

定理 5.  $\lambda$  项是否有  $\beta$  范式是不可判定的。



# 符号约定 2

**符号约定 2.** 本文中,我们用粗体的大写字母及由它们组成的字符串代表具体的组合子,不同的粗体字母字符串如不做特殊说明,一般表示不同的组合子。当它们出现在  $\lambda$  项中时,视同对应的组合子整体出现在  $\lambda$  项中。

用粗体大写字母及其字符串代表组合子,可用等号“=”表示。比如想用 **M** 代表  $\lambda x.x$ ,可写作:  
**M** =  $\lambda x.x$ 。



# 简单例子

- 1. 定义  $\mathbf{I} = \lambda x.x$ , 则  $\mathbf{I}a \equiv (\lambda x.x)a \triangleright_{\beta} a$ 。
- 2. 定义  $\mathbf{SWAP} = \lambda x.\lambda y.yx$ , 则  $\mathbf{SWAP}ab \equiv (\lambda x.\lambda y.yx)ab \triangleright_{\beta} ba$
- 3. 定义  $\mathbf{S} = \lambda x.\lambda y.\lambda z.xz(yz)$ , 则  $\mathbf{S} a b c \equiv (\lambda x.\lambda y.\lambda z.xz(yz)) a b c \triangleright_{\beta} a c (b c)$

# 示例

- Lambda> I = \x.x
- Lambda> I a
- a
- Lambda> SWAP = \x.\y.y x
- Lambda> SWAP a b
- b a
- Lambda> S = \x.\y.\z.x z(y z)
- Lambda> S a b c  
a c (b c)
- Lambda> I = S I
- Lambda> I m n
- n (m n)

# Outline

- Lambda演算定义
- 算术计算与逻辑谓词
- 递归与Y Combinator
- 有序对

# “模拟”自然数

- **ZERO** =  $\lambda f.\lambda x.x$
- **SUCC** =  $\lambda n.\lambda f.\lambda x.f (n f x)$
- **PLUS** =  $\lambda m.\lambda n.m \text{ SUCC } n$
- **MULT** =  $\lambda m.\lambda n.\lambda f.m (n f)$
- **POW** =  $\lambda b.\lambda e.e b$
- **PRED** =  $\lambda n.\lambda f.\lambda x.n (\lambda g.\lambda h.h (g f)) (\lambda u.x) (\lambda u.u)$
- **SUB** =  $\lambda m.\lambda n.n \text{ PRED } m$

# 结果

- 定义
  - $\text{Lambda} \triangleright \text{ZERO} = \lambda f. \lambda x. x$
  - $\text{Lambda} \triangleright \text{SUCC} = \lambda n. \lambda f. \lambda x. f (n f x)$
- 示例
  - $\text{Lambda} \triangleright \text{SUCC ZERO}$
  - $\lambda n. \lambda f. \lambda x. f (n f x) \lambda f. \lambda x. x$
  - $\lambda f. \lambda x. f (\lambda f. \lambda x. x) f x$
  - $\lambda f. \lambda x. f x$
  - $\text{Lambda} \triangleright \text{SUCC (SUCC ZERO)}$
  - $\lambda f. \lambda x. f (f x)$
  - $\text{Lambda} \triangleright \text{SUCC (SUCC (SUCC ZERO))}$

- $\lambda f.\lambda x.f(f(f x))$
- Lambda> SUCC(SUCC( SUCC (SUCC ZERO)))
- $\lambda f.\lambda x.f(f(f(f x)))$

- 定义
  - $\text{Lambda} \triangleright \text{ONE} = \text{SUCC ZERO}$
  - $\text{Lambda} \triangleright \text{TWO} = \text{SUCC ONE}$
  - $\text{Lambda} \triangleright \text{THREE} = \text{SUCC TWO}$
  - $\text{Lambda} \triangleright \text{FOUR} = \text{SUCC THREE}$

- [illegible]

# MULT THREE TWO

- $= \lambda m. \lambda n. \lambda f. m(n f) \text{ THREE TWO} \quad \# \text{ THREE 替换 } m \quad \text{TWO 替换 } n$
- $= \lambda f. \text{ THREE } (\text{ TWO } f)$
- $= \lambda f. \text{ THREE } (\lambda f1. \lambda x1. f1(f1(x1))) f) \quad \# f \text{ 替换 } f1$
- $= \lambda f. \text{ THREE } (\lambda x1. f(f(x1)))$
- $= \lambda f. \lambda f2. \lambda x2. f2(f2(f2(x2))) \quad (\lambda x1. f(f(x1))) \quad \# (\lambda x1. f(f(x1))) \text{ 替换 } f2$
- $= \lambda f. \lambda x2. (\lambda x1. f(f(x1))) ((\lambda x1. f(f(x1))) \quad (\lambda x1. f(f(x1))) x2)) \quad \# x2 \text{ 替换 } x1$
- $= \lambda f. \lambda x2. (\lambda x1. f(f(x1))) ((\lambda x1. f(f(x1))) f(f(x2))) \quad \# f(f(x2)) \text{ 替换 } x1$
- $= \lambda f. \lambda x2. (\lambda x1. f(f(x1))) f(f(f(f(x2)))) \quad \# f(f(f(f(x2)))) \text{ 替换 } x1$
- $= \lambda f. \lambda x2. f(f(f(f(f(f(x2)))))) \quad \# x2 \text{ 变换为 } x$
- $= \lambda f. \lambda x. f(f(f(f(f(f(x))))))$

# POW TWO THREE

- $= \lambda b. \lambda e. e \ b \ \text{TWO} \ \text{THREE}$  # TWO替换b THREE替换e
- $= \text{THREE} \ \text{TWO}$
- $= \lambda f. \lambda x. f(f(f(x))) \ \text{TWO}$  # TWO替换f
- $= \lambda x. \text{TWO}(\text{TWO}(\text{TWO}(x)))$
- $= \lambda x. \text{TWO}(\text{TWO}(\lambda f1. \lambda x1. f1(f1(x1) \ x))))$  # x替换f1
- $= \lambda x. \text{TWO}(\text{TWO}(\lambda x1. x(x(x1))))$
- $= \lambda x. \text{TWO}(\lambda f2. \lambda x2. f2(f2(x2)) \ (\lambda x1. x(x(x1))))$  #  $(\lambda x1. x(x(x1)))$ 替换f2
- $= \lambda x. \text{TWO}(\lambda x2. (\lambda x1. x(x(x1))) \ ((\lambda x1. x(x(x1)))(x2)))$  # x2替换x1
- $= \lambda x. \text{TWO}(\lambda x2. (\lambda x1. x(x(x1)))(x(x(x2))))$  #  $(x(x(x2)))$ 替换x1
- $= \lambda x. \text{TWO}(\lambda x2. x(x(x(x2))))$

- $=\lambda x. \text{TWO}(\lambda x2. x(x(x(x2))))$
- $=\lambda x. \lambda f3. \lambda x3. f3(f3(x3)(\lambda x2. x(x(x(x2)))))) \# (\lambda x2. x(x(x(x2))))$  替换f3
- $=\lambda x. \lambda x3. (\lambda x2. x(x(x(x2)))) (\lambda x2. x(x(x(x2)))) (x3)) \# x3$  替换x2
- $=\lambda x. \lambda x3. (\lambda x2. x(x(x(x2)))) x(x(x(x3))) \# (x(x(x(x3))))$  替换x2
- $=\lambda x. \lambda x3. x(x(x(x(x(x(x3))))))) \# x$  变换为f x3变换为x
- $=\lambda f. \lambda x. f(f(f(f(f(f(f(x))))))))$



# “模拟”逻辑

- 定义

- Lambda> TRUE =  $\lambda x.\lambda y.x$
- Lambda> FALSE =  $\lambda x.\lambda y.y$

- 逻辑

- Lambda> AND =  $\lambda p.\lambda q.p\ q\ p$
- Lambda> OR =  $\lambda p.\lambda q.p\ p\ q$
- Lambda> NOT =  $\lambda p.\lambda a.\lambda b.p\ b\ a$
- Lambda> IF =  $\lambda p.\lambda a.\lambda b.p\ a\ b$

- 示例

- Lambda> AND TRUE FALSE
- $\lambda x.\lambda y.y$
- Lambda> AND TRUE TRUE
- $\lambda x.\lambda y.x$

- Lambda> OR TRUE FALSE

- $\lambda x.\lambda y.x$

- Lambda> NOT TRUE

- $\lambda a.\lambda b.b$

- Lambda> NOT (NOT TRUE)

- $\lambda a.\lambda b.a$

- Lambda> IF TRUE a b

- a

- Lambda> IF FALSE a b

- b

- Lambda> IF (OR FALSE FALSE) a b

- b

# “模拟”谓词

- 定义
  - $\text{Lambda} > \text{ISZERO} = \lambda n.n (\lambda x.\text{FALSE}) \text{TRUE}$
  - $\text{Lambda} > \text{LEQ} = \lambda m.\lambda n.\text{ISZERO} (\text{SUB } m \ n)$
  - $\text{Lambda} > \text{EQ} = \lambda m.\lambda n. \text{AND} (\text{LEQ } m \ n) (\text{LEQ } n \ m)$
- 示例
  - $\text{Lambda} > \text{ISZERO TWO}$
  - $\lambda x.\lambda y.y$
  - $\text{Lambda} > \text{ISZERO ZERO}$
  - $\lambda x.\lambda y.x$
  - $\text{Lambda} > \text{LEQ ONE ONE}$
  - $\lambda x.\lambda y.x$
  - $\text{Lambda} > \text{LEQ TWO ONE}$
  - $\lambda x.\lambda y.y$
  - $\text{Lambda} > \text{IF } (\text{EQ ONE TWO}) \ a \ b$
  - $b$

# “模拟”函数

- `Lambda> MAX = \m.\n.IF (LEQ m n) n m`
- `Lambda> MAX ONE TWO`
- `\f.\x.f (f x)`
- `Lambda> MAX FOUR TWO`
- `\f.\x.f (f (f (f x)))`
- `Lambda> MIN = \m.\n.IF (LEQ m n) m n`
- `Lambda> MIN TWO THREE`
- `\f.\x.f (f x)`

# Outline

- Lambda演算定义
- 算术计算与逻辑谓词
- 递归与Y Combinator
- 有序对

# “模拟”递归

- 阶乘
  - **FACT** =  $\lambda n. \text{IF } (\text{ISZERO } n) \text{ ONE } (\text{MULT } n (\text{FACT } (\text{PRED } n)))$
- 有一个问题
  - FACT在Lamdar运算中不能递归定义

# 多一个参数

- [illegible]

# 定义一个组合子

- `Lambda> W = \x.x x`
- `Lambda> FACTD = W (\f.\n.IF (ISZERO n) ONE (MULT n (f f (PRED n))))`
- `Lambda> FACTD THREE`
- `\f.\x.f (f (f (f (f x))))`

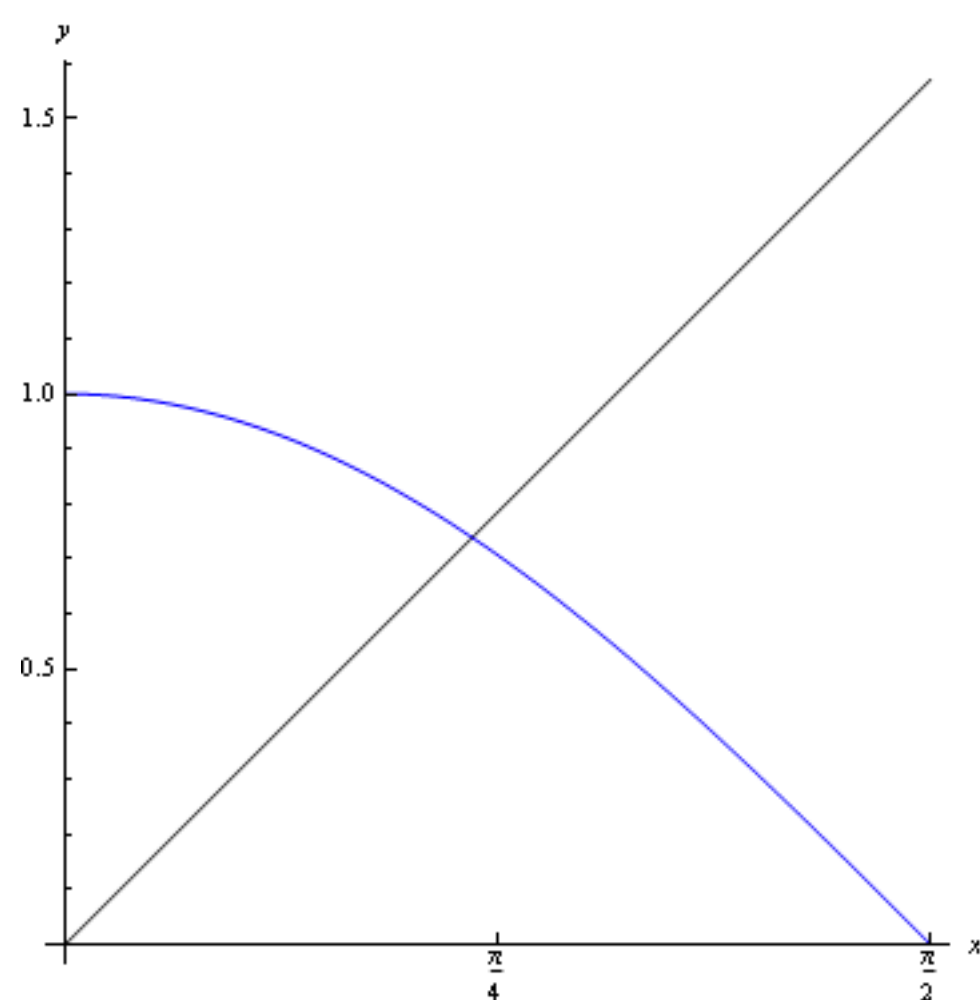
# 双重应用

- Lambda> W = \x.x x
- Lambda> ADD = W (\f.\n.\m.IF (ISZERO m) n (f f (SUCC n) (PRED m)))
- Lambda> ADD TWO FOUR
- \f.\x.f (f (f (f (f (f x)))))
- Lambda> ADD FOUR FOUR
- \f.\x.f (f (f (f (f (f (f (f (f x))))))))



# 我们的目的是什么？

- 只用一重应用来定义递归函数。



### Algorithm - Fixed Point Iteration Scheme

Given an equation  $f(x) = 0$

Convert  $f(x) = 0$  into the form  $x = g(x)$

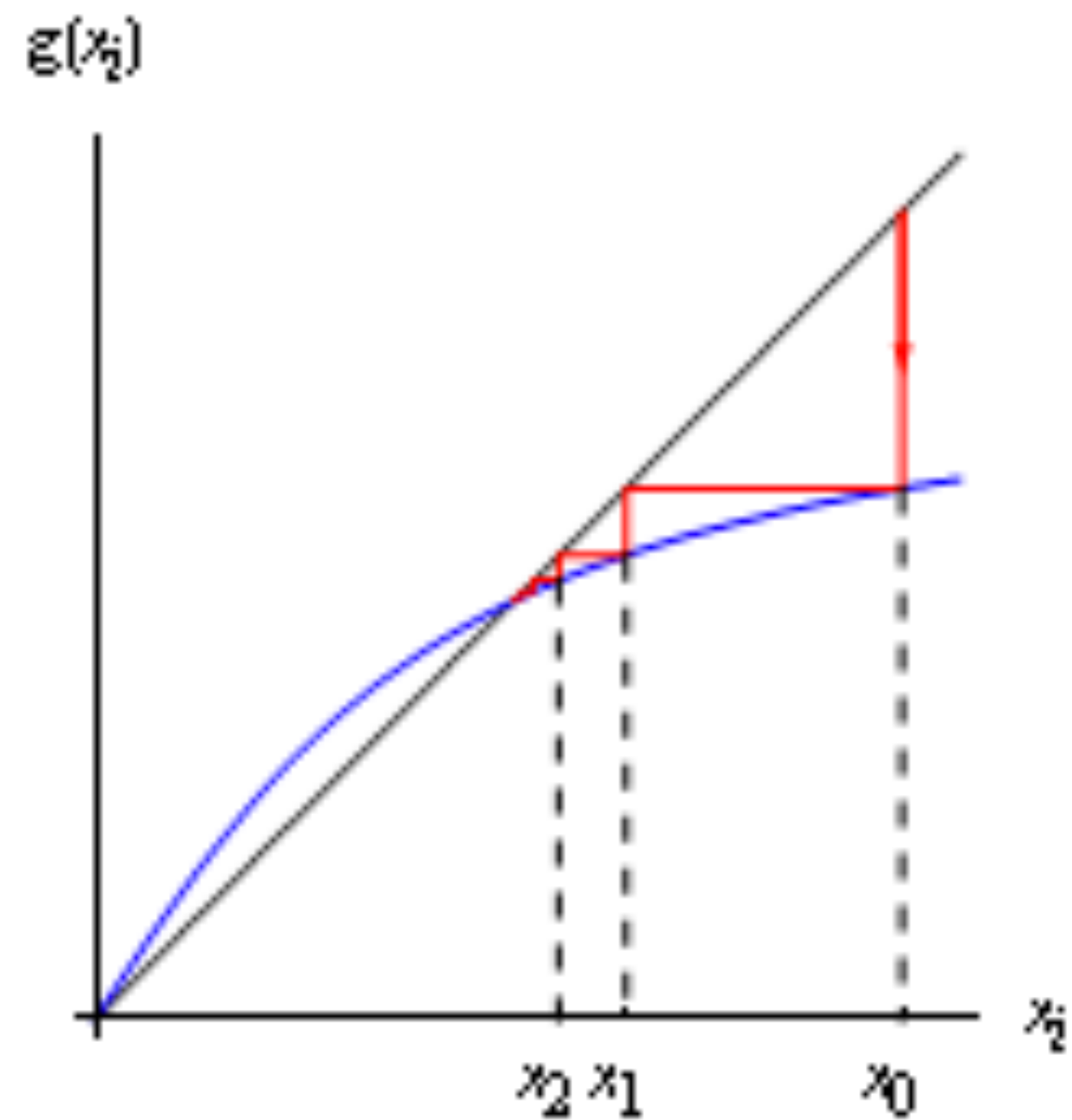
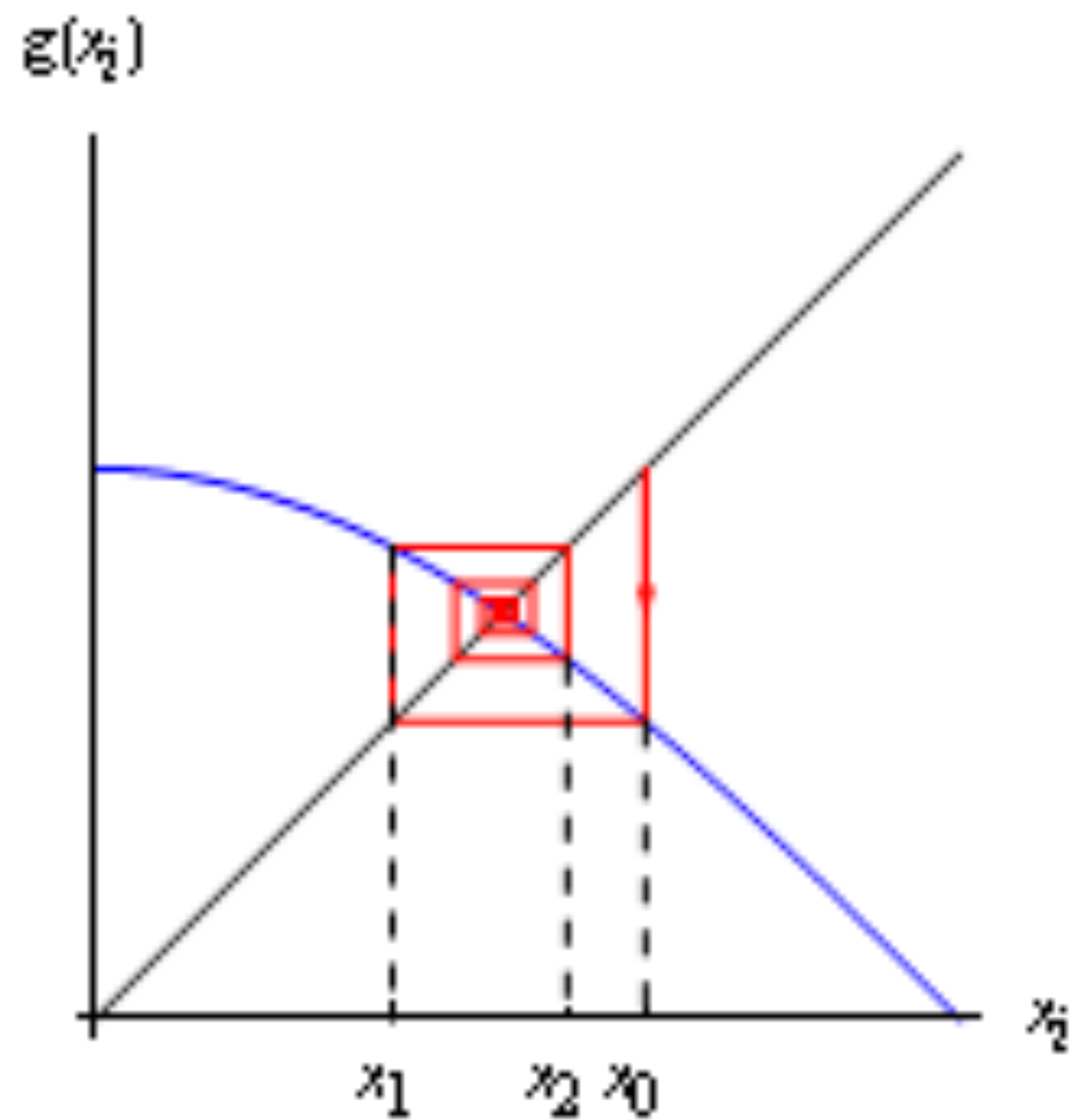
Let the initial guess be  $x_0$

Do

$$x_{i+1} = g(x_i)$$

while (none of the convergence criterion C1 or C2 is met)

# 不动点



不动点迭代收敛的过程

# Y Combinator

- 不动点
  - $f(x) = x$
- Y Combinator
  - Lambda> :set +hold
  - Lambda> Y
  - $\backslash g.(\backslash x.g (x x)) \backslash x.g (x x)$

# $YF$ 就是 $F$ 的不动点

- $YF$
- $\equiv (\lambda g. (\lambda x. g (x x)) \lambda x. g (x x)) F$
- $=_{\beta} (\lambda x. F (x x)) \lambda x. F (x x)$
- $=_{\beta} F ((\lambda x. F (x x)) \lambda x. F (x x))$
- $=_{\beta} F(YF)$ // $Y$ 的定义带入 $F$

# 利用不动点消除两重应用

```
Lambda> FACT2 = \f.\n.IF (ISZERO n) ONE (MULT n (f (PRED n)))
```

```
Lambda> FACTY = Y FACT2
```

```
Lambda> FACTY THREE
```

$$\backslash f. \backslash x. f \ (f \ (f \ (f \ (f \ (f \ x))))$$

```
Lambda> FACTY FOUR
```

[illegible]

# 利用不动点消除两重应用

- **FACT THREE**
- $=_{\beta}$  **Y FACT2 THREE** (由定义)
- $=_{\beta}$  **FACT2 (Y FACT2) THREE** (因为  $Y F =_{\beta} F (Y F)$ )
- $=_{\beta}$  **IF (ISZERO THREE) ONE (MULT THREE (Y FACT2 TWO))**
- $=_{\beta}$  **MULT THREE (Y FACT2 TWO)**

# 练习

- $R \equiv (\lambda f \lambda n . \text{ISZERO } n \text{ ZERO } (n \text{ SUCC}(f(\text{PRED } n))))$ 
  - This definition tells us that the number  $n$  is tested: **if it is zero** the result of the sum is zero. If  $n$  is not zero, then the **successor** function is applied  $n$  times to the recursive call (the argument  $r$ ) of the function applied to the **predecessor** of  $n$ .
- Y R THREE



# Y R THREE

- = R (Y R) THREE
- = THREE SUCC ((Y R) TWO)
- = THREE SUCC(TWO SUCC((Y R) ONE))
- = THREE SUCC(TWO SUCC(ONE SUCC ((Y R) ZERO)))
- = THREE SUCC(TWO SUCC(ONE SUCC (ZERO)))
- = THREE SUCC(TWO SUCC((\f.\x.f x) SUCC (ZERO)))
- = THREE SUCC(TWO SUCC(ONE))
- = SIX

# 图灵不动点组合子

- [illegible]

# Outline

- Lambda演算定义
- 算术计算与逻辑谓词
- 递归与Y Combinator
- 有序对

还记得对数据抽象的有序对

# 构建有序对

- 合并一个表
  - Lambda> CONS = \x.\y.\f. f x y
- 取出表的第一个元素
  - Lambda> CAR = \p.p TRUE
- 去除表的第一个元素
  - Lambda> CDR = \p.p FALSE
- 空的有序对
  - Lambda> NIL = \x. TRUE
- 谓词用于判断一个有序对是否为空
  - Lambda> NULL = \p.p (\x.\y.FALSE)

# 验证这些基本定义

- `Lambda> CONS a (CONS b (CONS c NIL))`
- `\f.f a \f.f b \f.f c \x.\x.y.x`
- `Lambda> CAR (CONS a (CONS b (CONS c NIL)))`
- `a`
- `Lambda> CDR (CONS a (CONS b (CONS c NIL)))`
- `\f.f b \f.f c \x.\x.y.x`
- `Lambda> CAR (CDR (CONS a (CONS b (CONS c NIL))))`
- `b`
- `Lambda> NULL (CDR (CONS a (CONS b (CONS c NIL))))`
- `\x.\y.y`
- `Lambda> NULL NIL`
- `\x.\y.x`

# 定义长度函数

- `Lambda> LENGTH = Y (\g.\c.\x. NULL x c (g (SUCC c) (CDR x))) ZERO`
- `Lambda> LENGTH NIL`
- `\f.\x.x`
- `Lambda> LENGTH (CONS a (CONS b (CONS c NIL)))`
- `\f.\x.f (f (f x))`
- `Lambda> LENGTH (CONS a (CONS b (CONS c (CONS d NIL))))`
- `\f.\x.f (f (f (f x)))`

# LENGTH NIL

- $\text{LENGTH} = Y (\backslash g.\backslash c.\backslash x. \text{NULL } x \text{ } c \text{ } (g \text{ } (\text{SUCC } c) \text{ } (\text{CDR } x))) \text{ ZERO}$
- $\text{LENGTH NIL}$