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 Comibnator
- 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：
  - <表达式> ::= <标识符>
  - <表达式> ::=(λ<标识符> .<表达式>)
  - <表达式> ::=(<表达式> <表达式>)
- 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：
  - (λ x. 2x)
  - (λ x y. x+y) a b   其实科里化（Currying）后变为$(((\backslash x.(\backslash y.(y+x)))\ a)\ b)$单参数的Larmdar演算

定义 1. (λ 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7
同,这里就是指字符串本身)。那么 λ 项定义如下:

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

形式化定义

lambda 项

示例 1.（一些 λ 项） 下面这些都是 λ 项：

$$(\lambda x.(x\,y)) \qquad (x(\lambda x.(\lambda x.x))) \qquad ((((a\,b)\,c)\,d)\,e)$$

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

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 λ 项，用除 λ 以外的小写希腊字母如 ϕ, ψ 等表示任意 λ 项中的变量。

对于括号，则有如下的省略规定：

1. λ 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 λ 项，如 $(((M\,N)\,P)\,Q)$，括号可以省略，表示为 $M\,N\,P\,Q$；
3. 抽象型的 λ 项 $(\lambda\phi.M)$ 中，M 最外层的括号可以省略，如 $\lambda x.(y\,z)$ 可以省略为 $\lambda x.y\,z$。

也就是说，我们把省略形式视同定义 1 中的 λ 项。

示例 2.（省略表示） 下面给出了一些省略表示的 λ 项。

省略表示	完整的 λ 项
$\lambda x.\lambda y.y\,x\,a\,b$	$(\lambda x.(\lambda y.(((y\,x)\,a)\,b)))$
$(\lambda x.\lambda y.y\,x)\,a\,b$	$(((\lambda x.(\lambda y.(y\,x)))\,a)\,b)$
$\lambda g.(\lambda x.g\,(x\,x))\,\lambda x.g\,(x\,x)$	$(\lambda g.((\lambda x.(g\,(x\,x)))\,(\lambda x.(g\,(x\,x)))))$
$\lambda x.\lambda y.a\,b\,\lambda z.z$	$(\lambda x.(\lambda y.((a\,b)\,(\lambda z.z))))$

自由变量和组合子

定义 2.（语法全等） 我们用恒等号 "≡" 表示两个 λ 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 λ 项 P，我们可以定义 P 中自由变量的集合 $\mathrm{FV}(P)$ 如下：

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

从第 2 可以看出抽象 $\lambda\phi.M$ 中的变量 ϕ 是要从 M 中被排除出自由变量这个集合的。若 M 中有 ϕ，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 λ 项来说，这两个集合的交集未必为空。

示例 4.（自由变量）

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

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 λ 项 P 有 $\mathrm{FV}(P) = \varnothing$，则称 P 是封闭的，这样的 P 又称为组合子。

演算公理系统

- 置换公理(alpha 变换)
  - λ xy.x+y => λab.a+b
  - lambda x y. x + y => lambda a b. a + b
- 代入公理（beta 规约）
  - (λ xy. x+y) a b=>a+b
  - λ y.(λ x. x+y a) b
- 例如：
  - (λx.λy.x - y) 7 2与(λy.7 - y) 2与7 - 2是等价的

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

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

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

$$P \quad \equiv_\alpha \quad Q$$

alpha 变换

定义 7. (**β 规约**)形如

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

的 λ 项被称为 β 可约式,对应的项

$$[N/\phi]\,M$$

则称为 β 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 指称替换后的得到的项,那么我们说 P 被 β 缩减为 R,写做:

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

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

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

beta 规约

练习

- (λ x. x (x y)) m
- (λ x. y) n
- (λ x. (λ y. y x) z) v
- (λ x. x x)(λ x. x x)
- (λ x. x x y)(λ 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 $P \rhd_\beta M$ 且 $P \rhd_\beta N$, 则存在一个 λ 项 T 使得

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

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

定理 4. 对 P 的总是先 β 缩减最左侧最外侧的 β 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. λ 项是否有 β 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 λ 项中时，视同对应的组合子整体出现在 λ 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：$\mathbf{M} = \lambda x.x$。

简单例子

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

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** = $\backslash f.\backslash x.x$
- **SUCC** = $\backslash n.\backslash f.\backslash x.f\ (n\ f\ x)$
- **PLUS** = $\backslash m.\backslash n.m$ **SUCC** n
- **MULT** = $\backslash m.\backslash n.\backslash f.m\ (n\ f)$
- **POW** = $\backslash b.\backslash e.e\ b$
- **PRED** = $\backslash n.\backslash f.\backslash x.n$ (\g.\h.h $(g\ f)$) $(\backslash u.x)$ $(\backslash u.u)$
- **SUB** = $\backslash m.\backslash n.n$ **PRED** m

结果

- 定义
  - Lambda> ZERO = \f.\x.x
  - Lambda> SUCC = \n.\f.\x.f (n f x)
- 示例
  - Lambda> SUCC ZERO
  - \n.\f.\x.f (n f x) \f.\x.x
  - \f.\x.f ((\f.\x.x) f x)
  - \f.\x.f x
  - Lambda> SUCC (SUCC ZERO)
  - \f.\x.f (f x)
  - Lambda> SUCC(SUCC (SUCC ZERO))
  - \f.\x.f (f (f x))
  - Lambda> SUCC(SUCC(SUCC (SUCC ZERO)))
  - \f.\x.f (f (f (f x)))
- 定义
  - Lambda> ONE = SUCC ZERO
  - Lambda> TWO = SUCC ONE
  - Lambda> THREE = SUCC TWO
  - Lambda> FOUR = SUCC THREE
- 示例
  - Lambda> PLUS TWO THREE
  - \f.\x.f (f (f (f (f x))))
  - Lambda> POW TWO FOUR
  - \x.\a.x (x (x (x (x (x (x (x (x (x (x (x (x (x (x (x a)))))))))))))))
  - Lambda> MULT THREE TWO
  - \f.\x.f (f (f (f (f (f x)))))
  - Lambda> SUB FOUR TWO
  - \f.\x.f (f x)
  - Lambda> PRED ONE
  - \f.\x.x

MULT THREE TWO

- =\m.\n.\f. m(n f) THREE TWO # THREE替换m TWO替换n
- =\f. THREE (TWO f)
- =\f. THREE (\f1.\x1. f1(f1(x1)) f)   # f替换f1
- =\f. THREE (\x1. f(f(x1)))
- =\f. \f2.\x2.f2(f2(f2(x2))) (\x1. f(f(x1))) # (\x1. f(f(x1)))替换f2
- =\f. \x2. (\x1.f(f(x1)) ((\x1.f(f(x1))) ((\x1.f(f(x1)) x2)))#x2替换x1
- =\f. \x2. (\x1.f(f(x1)) ((\x1.f(f(x1))) f(f(x2))) #f(f(x2))替换x1
- =\f. \x2. (\x1.f(f(x1)) f(f(f(f(x2))))) #f(f(f(f(x2))))替换x1
- =\f. \x2. f(f(f(f(f(f(x2)))))) #x2变换为x
- =\f. \x. f(f(f(f(f(f(x))))))

POW TWO THREE

- = \b.\e.e b TWO THREE # TWO替换b THREE替换e
- = THREE TWO
- = \f.\x. f(f(f(x))) TWO   # TWO替换f
- =\x. TWO(TWO(TWO(x)))
- =\x. TWO(TWO(\f1.\x1.f1(f1(x1) x))) #x替换f1
- =\x. TWO(TWO(\x1.x(x(x1))))
- =\x. TWO(\f2.\x2.f2(f2(x2)) (\x1.x(x(x1))))   # (\x1.x(x(x1)))替换f2
- =\x. TWO(\x2.(\x1.x(x(x1))) ((\x1.x(x(x1)))(x2)) # x2替换x1
- =\x. TWO(\x2.(\x1.x(x(x1)))(x(x(x2)))) # (x(x(x2)))替换x1
- =\x. TWO(\x2. x(x(x(x(x2)))))

- =\x. TWO(\x2. x(x(x(x(x2)))))
- =\x. \f3.\x3.f3(f3(x3)(\x2. x(x(x(x(x2))))) # (\x2. x(x(x(x(x2)))))替换f3
- =\x. \x3. (\x2.x(x(x(x(x2))))) (\x2. x(x(x(x(x2)))) (x3)) #x3替换x2
- =\x. \x3. (\x2.x(x(x(x(x2))))) x(x(x(x(x3)))) #(x(x(x(x(x3))))替换x2
- =\x. \x3. x(x(x(x(x(x(x(x(x3)))))))) #x变换为f x3变换为x
- =\f. \x. f(f(f(f(f(f(f(f(x))))))))

“模拟”逻辑

- 定义
    - Lambda> TRUE = \x.\y.x
    - Lambda> FALSE = \x.\y.y
- 逻辑
    - Lambda> AND = \p.\q.p q p
    - Lambda> OR = \p.\q.p p q
    - Lambda> NOT = \p.\a.\b.p b a
    - Lambda> IF = \p.\a.\b.p a b
- 示例
- Lambda> AND TRUE FALSE
- \x.\y.y
- Lambda> AND TRUE TRUE
- \x.\y.x
- Lambda> OR TRUE FALSE
- \x.\y.x
- Lambda> NOT TRUE
- \a.\b.b
- Lambda> NOT (NOT TRUE)
- \a.\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
  - Lambda> ISZERO = \n.n (\x.FALSE) TRUE
  - Lambda> LEQ = \m.\n.ISZERO (SUB m n)
  - Lambda> EQ = \m.\n. AND (LEQ m n) (LEQ n m)
- 示例
  - Lambda> ISZERO TWO
  - \x.\y.y
  - Lambda> ISZERO ZERO
  - \x.\y.x
  - Lambda> LEQ ONE ONE
  - \x.\y.x
  - Lambda> LEQ TWO ONE
  - \x.\y.y
  - Lambda> IF (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
  - $\mathbf{FACT} = \backslash n.\mathbf{IF}\ (\mathbf{ISZERO}\ n)\ \mathbf{ONE}\ (\mathbf{MULT}\ n\ (\mathbf{FACT}\ (\mathbf{PRED}\ n)))$
- 有一个问题
  - FACT在Lamdar运算中不能递归定义

多一个参数

- Lambda> FACT1 = \f.\n.IF (ISZERO n) ONE (MULT n (f f (PRED n)))
- Lambda> FACT = FACT1 FACT1
- Lambda> FACT THREE
- \f.\x.f (f (f (f (f (f x)))))
- Lambda> FACT FOUR
  \f.\x.f (f (f (f (f (f (f (f (f (f (f (f (f (f (f (f (f (f (f (f (f (f (f (f x)))))))))))))))))))))))

定义一个组合子

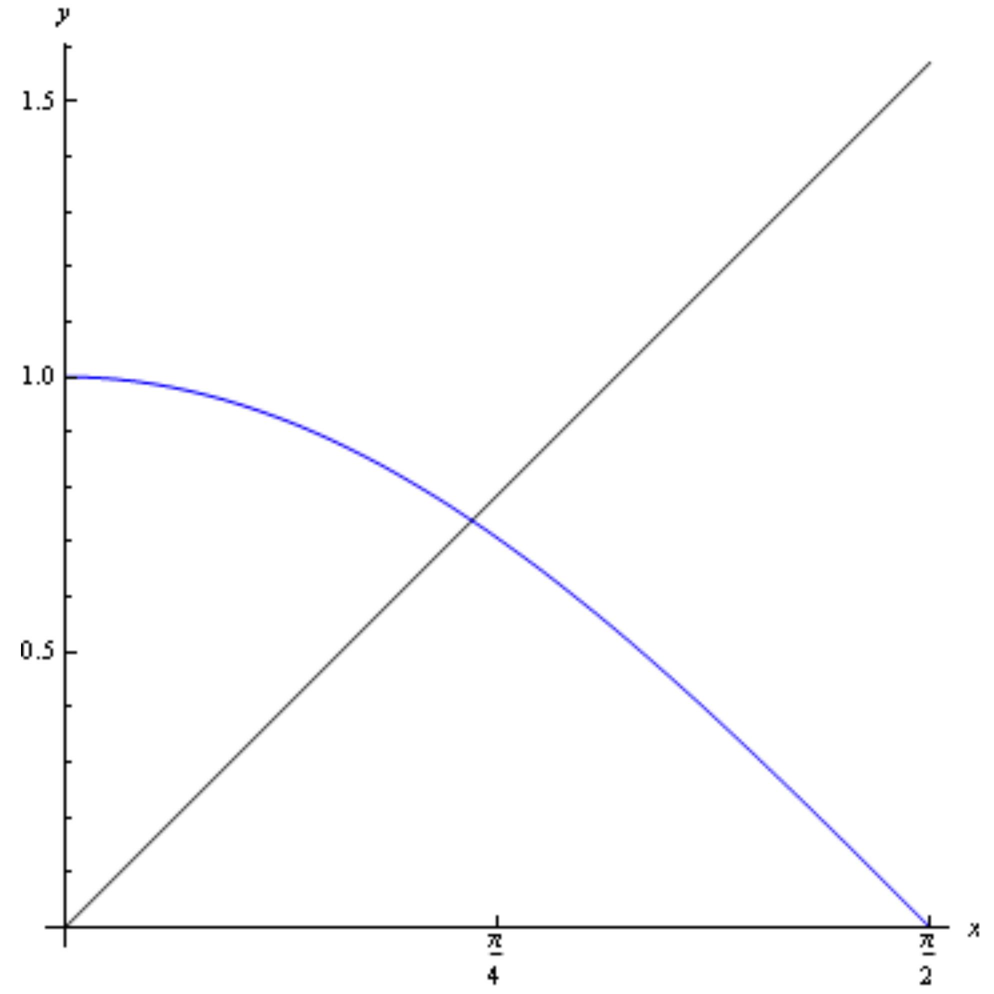
- Lambda> W = \x.x x
- Lambda> FACTD = W (\f.\n.IF (ISZERO n) ONE (MULT n (f f (PRED n))))
- Lambda> FACTD THREE
- \f.\x.f (f (f (f (f (f x)))))

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 x)))))))

我们的目的是什么？

- 只用一重应用来定义递归函数。



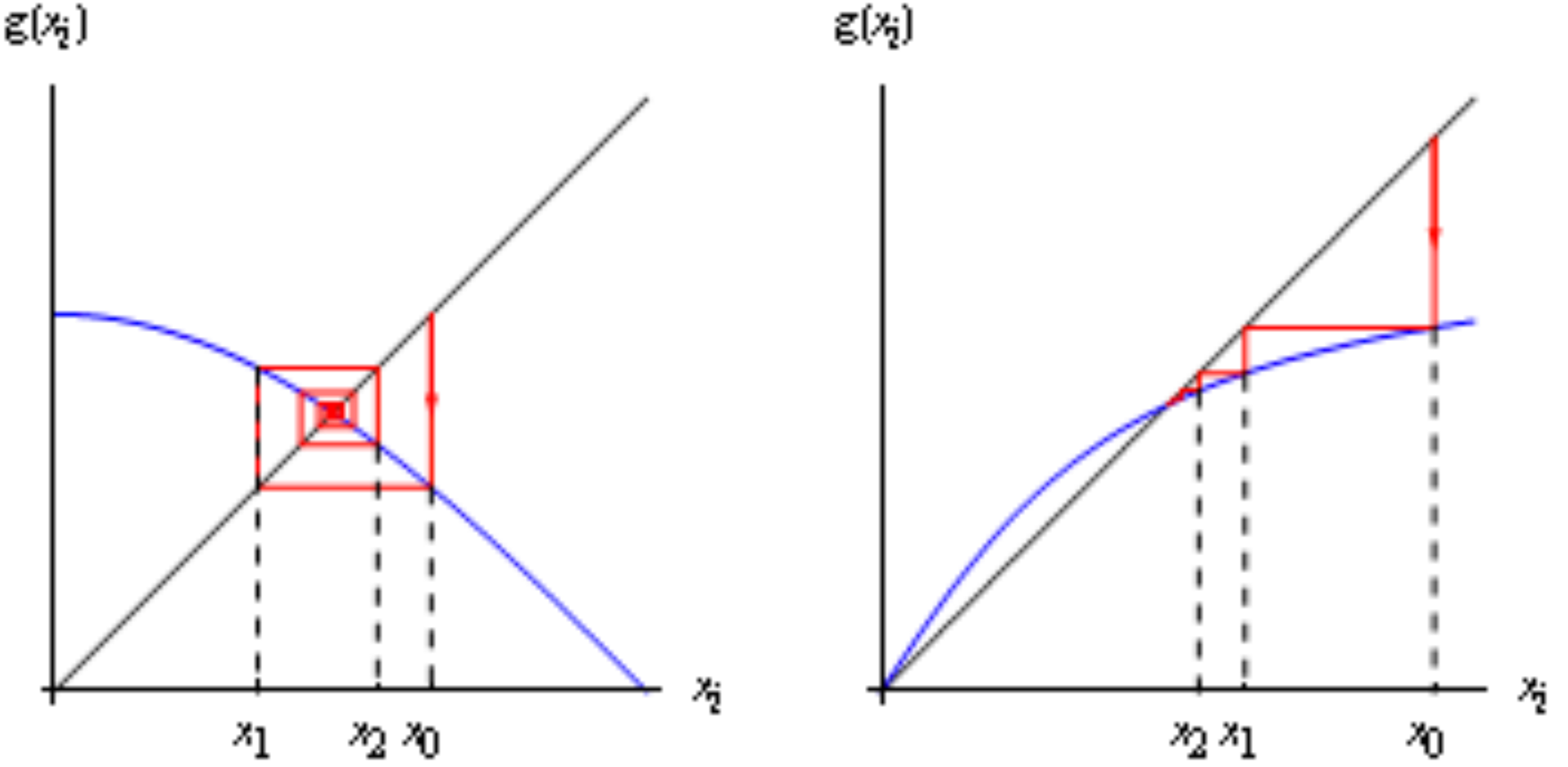
Algorithm - Fixed Point Iteration Scheme

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
  - \g.(\x.g (x x)) \x.g (x x)

YF就是F的不动点

- $\mathbf{Y}F$
- $\equiv$ (\g.(\$x$.g ($x$ x)) \$x$.g ($x$ x)) F
- $=_\beta$ (\$x$.$F$ (x x)) \$x$.$F$ (x x)
- $=_\beta$ F ((\$x$.$F$ (x x)) \$x$.$F$ (x x))
- $=_\beta$ $F(\mathbf{Y}F)$//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
\f.\x.f (f (f (f (f (f x)))))
Lambda> FACTY FOUR
\f.\x.f (f (f (f (f (f (f (f (f (f (f (f (f (f (f (f (f (f (f (f (f (f
(f (f x)))))))))))))))))))))))
```

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≡(\f.\n. ISZERO n ZERO (n SUCC(f(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

- Lambda> T = (\x.\y.y (x x y)) \x.\y.y (x x y)
- Lambda> FACTT = T FACT2
- Lambda> FACTT FOUR
  \f.\x.f (f (f (f (f (f (f (f (f (f (f (f (f (f (f (f (f (f (f (f (f (f (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

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

- LENGTH = Y (\g.\c.\x. NULL x c (g (SUCC c) (CDR x))) ZERO
- LENGTH NIL