



λ -calculus

Chapter 11.7

What can be done by a computer?

- Algorithm formalization – 1930s
 - Church, Turing, Kleene, Post, etc.
 - *Church's thesis*:
All intuitive computing models are equally powerful.
- Turing machine
 - automaton with an unbounded tape
 - *imperative programming*
- Church's λ -calculus
 - computes by substituting parameters into expressions
 - *functional programming*
- Logic: Horn clauses
 - collection of axioms to solve a goal
 - *logic programming*

λ -calculus

- λ -calculus
 - Church (1941) – to study computations with functions
 - *Everything is a function!*
- λ -expressions – defined recursively:
 - name: x, y, z, u, v, \dots
 - abstraction: $\lambda x.M$ Lambda is a non name function
 - function with parameter x and body M
 - applications: $M N$ – function M applied to N
- Examples
 - $(\lambda x.x^*x)$ - a function that maps x to x^*x
a function take in x and return x^*x
 - $(\lambda x.x^*x) 4$ - the same function applied to 4
M N

result is 16 $\rightarrow 4^*4$

λ -calculus

- Syntactic rules
 - application is left-associative
 $x\ y\ z$ means $(x\ y)\ z$
 - application has higher precedence than abstraction
 $\lambda x. A\ B$ means $\lambda x. (A\ B)$ (not $(\lambda x. A)\ B$)
 - consecutive abstractions:
 $\lambda x_1 x_2 \dots x_n. e$ means $\lambda x_1. (\lambda x_2. (\dots (\lambda x_n. e) \dots))$
怎么分裂多个parameters
- Example:
 $\lambda xyz. \boxed{x\ z\ (y\ z)} = (\lambda x. (\lambda y. (\lambda z. ((x\ z)\ (y\ z)))))$

这后面是个E 是一个大程序 然后里面 x z因为left associative合在一起

λ -calculus

- Context-free grammars (CFG)

- CFG for λ -expressions

$$\textit{expr} \rightarrow \textit{name} \mid (\lambda \textit{name} . \textit{expr}) \mid (\textit{expr} \textit{expr})$$

- CFG for λ -expressions with minimum parentheses

$$\textit{expr} \rightarrow \textit{name} \mid \lambda \textit{name} . \textit{expr} \mid \textit{func arg}$$
$$\textit{func} \rightarrow \textit{name} \mid (\lambda \textit{name} . \textit{expr}) \mid \textit{func arg}$$
$$\textit{arg} \rightarrow \textit{name} \mid (\lambda \textit{name} . \textit{expr}) \mid (\textit{func arg})$$

λ -calculus

For lambda calculus, you have one argument only

- Examples

square = $\lambda x.\text{times } x \ x$

identity = $\lambda x.x$

const7 = $\lambda x.7$

hypot = $\lambda x.\lambda y.\text{sqrt}(\text{plus}(\text{square } x)(\text{square } y))$

$\lambda xy.____$ is the short written form

Free and bound variables

- $\lambda x.M$ - is a *binding* of the variable (or name) x
 - lexical scope
 - x is said to be *bound* in $\lambda x.M$
 - all x in $\lambda x.M$ are bound within the scope of this binding
- x is *free* in M if it is not bound
- $free(M)$ - the set of free variables in M
 - $free(x) = \{x\}$
 - $free(M N) = free(M) \cup free(N)$
 - $free(\lambda x.M) = free(M) - \{x\}$
- $bound(M)$ - the set of variables which are not free
 - any occurrence of a variable is free or bound; not both

Free and bound variables

- Example
 - x is free
 - y, z are bound

$$\lambda \textcolor{red}{y}.\lambda \textcolor{green}{z}.x \textcolor{green}{z} (\textcolor{red}{y} \textcolor{green}{z})$$

只要变成parameters就都不是free 全是bound

Computing with pure λ -terms

- Computing idea:
 - reduce the terms into as simple a form as possible
 - $(\lambda x.M) N =_{\beta} \{N/x\}M$ – substitute N for x in M
 - the right-hand side is expected to be simpler
- Example:

$$(\lambda xy.x) u v =_{\beta} (\lambda y.u) v =_{\beta} u$$

把u当成x带入方程式

把v当成y带入方程式
但因为不存在y所以直接就给 β u

$$\begin{aligned} & (\lambda xy.x) u v \\ \Rightarrow & (\lambda x. (\lambda y.x)) u v \\ \Rightarrow & ((\lambda x. (\lambda y.x)) u) v \quad \rightarrow \text{left associativity} \end{aligned}$$

这是Function
 $\lambda x.M$ applied to
N

Substitution

- $\{N/x\}M$ – substitution of term N for variable x in M
- Substitution rules (informal):
 - (i) if $free(N) \cap bound(M) = \emptyset$
then just replace all free occurrences of x in M
 - (ii) otherwise, rename with fresh variables until (i) applies

M的parameter不能和你要替代的variable一样

Substitution rules

- In variables: the same or different variable

- $\{N/x\}x = N$
 - $\{N/x\}y = y, y \neq x$

- In applications – the substitution distributes

- $\{N/x\}(P Q) = \{N/x\}P \{N/x\}Q$

- In abstractions – several cases

- no free x :

$$\{N/x\}(\lambda x.P) = \lambda x.P$$

- no interaction – y is not free in N :

$$\{N/x\}(\lambda y.P) = \lambda y. \{N/x\}P, \quad y \neq x, y \notin \text{free}(N)$$

- *renaming* – y is free in N ; y is renamed to z in P :

$$\{N/x\}(\lambda y.P) = \lambda z. \{N/x\} \{z/y\}P,$$

$$y \neq x, y \in \text{free}(N), z \notin \text{free}(N) \cup \text{free}(P)$$

Computing with pure λ -terms

- Rewriting rules
- α -conversion – renaming the formal parameters
$$\lambda x.M \xrightarrow{\alpha} \lambda y. \{y/x\}M, y \notin \text{free}(M)$$
- β -reduction – applying an abstraction to an argument
$$(\lambda x.M) N \xrightarrow{\beta} \{N/x\} M$$

($\lambda x.M$) N 可以 reduce
但 $\lambda x.M$ N 不可以 reduce
因为 application 有更高的运算等级
 $\lambda x.M$ N =» $\lambda x. (M N)$

Equality of pure λ -terms

- Example

$$(\lambda xyz.x z (y z)) (\underline{\lambda x.x}) (\lambda x.x)$$

$$\Rightarrow_{\alpha} (\lambda xyz.x z (y z)) (\lambda u.u) (\underline{\lambda x.x})$$

$$\Rightarrow_{\alpha} (\underline{\lambda xyz.x z (y z)}) (\underline{\lambda u.u}) (\lambda v.v)$$

$$\Rightarrow_{\beta} (\lambda yz.(\underline{\lambda u.u}) \underline{z} (y z)) (\lambda v.v)$$

两种都可以 可以把 λu reduce 也可以把 λv reduce了

$$\Rightarrow_{\beta} (\underline{\lambda yz.z} (y z)) (\underline{\lambda v.v})$$

$$\Rightarrow_{\beta} \lambda z.z ((\underline{\lambda v.v}) \underline{z})$$

$$\Rightarrow_{\beta} \lambda z.z z$$

here is nothing to apply, 不能继续reduce下去了

Equality of pure λ -terms

- Example

$$\begin{aligned} & ((\underline{\lambda fgh.fg}(h h)) (\underline{\lambda xy.x})) \underline{h} (\lambda x.x x) \\ & \Rightarrow_{\beta} (\lambda gh.(\lambda xy.x) g (\underline{h} h)) h (\lambda x.x x) \\ & \Rightarrow_{\alpha} (\underline{\lambda gk.(\lambda xy.x)} g (\underline{k} k)) \underline{h} (\lambda x.x x) \\ & \Rightarrow_{\beta} (\underline{\lambda k.(\lambda xy.x)} h (\underline{k} k)) (\underline{\lambda x.x} x) \\ & \Rightarrow_{\beta} (\underline{\lambda xy.x} \underline{h} ((\lambda x.x x) (\lambda x.x x))) \\ & \Rightarrow_{\beta} (\underline{\lambda y.h} ((\underline{\lambda x.x} x) (\lambda x.x x))) \\ & \Rightarrow_{\beta} h \end{aligned}$$

application之间有left associativity

Computing with pure λ -terms

- Rewriting rules
- *Reduction*: any sequence of $\Rightarrow_\alpha, \Rightarrow_\beta$
- Normal form: term that cannot be β -reduced
 - β -normal form
 - Example of normal form
 $\underline{\lambda x.x\ x}$ – cannot be reduced

Computing with pure λ -terms

- There may be several ways to reduce to a normal form
- Example: any path below is such a reduction

$$\begin{array}{c} (\underline{\lambda xyz.x} z (y z)) (\underline{\lambda x.x}) (\lambda x.x) \\ \Downarrow \quad \text{因为 } \lambda x.x \text{ 没有 free variable 所以不 conversion 不要紧} \\ (\lambda yz.(\underline{\lambda x.x}) z (y z)) (\underline{\lambda x.x}) \\ \Downarrow_{\beta} \quad \Downarrow_{\beta} \\ (\lambda yz.z (y z)) (\underline{\lambda x.x}) \\ \Downarrow_{\beta} \quad \Downarrow_{\beta} \\ \lambda z.z ((\underline{\lambda x.x}) z) \quad \lambda z.(\underline{\lambda x.x}) z z \\ \Downarrow_{\beta} \quad \Downarrow_{\beta} \\ \lambda z.z z \end{array}$$

如果有的function有free variable 带入apply进一个function 变成bound variable了就会有问题

Computing with pure λ -terms

- Nonterminating reductions

- Never reach a normal form
- Example

$$\underline{(\lambda x.x\ x)\ (\lambda x.x\ x) \Rightarrow_{\beta} (\lambda x.x\ x)\ (\lambda x.x\ x)}$$

There is no normal form for this
(可能会考?)

Computing with pure λ -terms

- *Theorem (Church-Rosser, 1936)*

For all pure λ -terms M, P, Q , if

$$M \Rightarrow_{\beta}^* P \text{ and } M \Rightarrow_{\beta}^* Q,$$

then there exists a term R such that

$$P \Rightarrow_{\beta}^* R \text{ and } Q \Rightarrow_{\beta}^* R.$$

- In particular, the normal form, when exists, is unique.

所有的normal form都是
独一无二的

Computing with pure λ -terms

- Reduction strategies
- *Call-by-value reduction (applicative order)*
 - parameters are evaluated first, then passed
 - might not reach a normal form even if there is one
 - leftmost innermost lambda that can be applied
- Example

$$\begin{aligned} & (\lambda y. h) ((\underline{\lambda x. x \ x}) (\underline{\lambda x. x \ x})) \\ & \Rightarrow_{\beta} (\lambda y. h) ((\underline{\lambda x. x \ x}) (\lambda x. x \ x)) \\ & \Rightarrow_{\beta} (\lambda y. h) ((\lambda x. x \ x) (\lambda x. x \ x)) \\ & \Rightarrow_{\beta} \dots \end{aligned}$$

Computing with pure λ -terms

- Reduction strategies
- *Call-by-name reduction (normal order)*
 - parameters are passed unevaluated
 - leftmost outermost lambda that can be applied

- Example

$$(\lambda y. h) ((\lambda x. x\ x) (\lambda x. x\ x)) \Rightarrow_{\beta} h$$

- *Theorem (Church-Rosser, 1936)*

Normal order reduction reaches a normal form if there is one.

- Functional languages use also call-by-value because it can be implemented efficiently and it might reach the normal form faster than call-by-name.

λ -calculus can model everything

- Boolean values
- True: $T \equiv \lambda x. \lambda y. x$
 - interpretation: of a pair of values, choose the first
- False: $F \equiv \lambda x. \lambda y. y$
 - interpretation: of a pair of values, choose the second
- Properties:

TPQ (left associative)

$$\frac{((T \ P) \ Q) \Rightarrow_{\beta} (((\lambda x. \lambda y. x) \ P) \ Q) \Rightarrow_{\beta} ((\lambda y. P) \ Q) \Rightarrow_{\beta} P}{((F \ P) \ Q) \Rightarrow_{\beta} (((\lambda x. \lambda y. y) \ P) \ Q) \Rightarrow_{\beta} ((\lambda y. y) \ Q) \Rightarrow_{\beta} Q}$$

λ -calculus can model everything

- Boolean functions
- $\text{not} \equiv \lambda x.((x \ F) \ T)$
- $\text{and} \equiv \lambda x.\lambda y.((x \ y) \ F)$
- $\text{or} \equiv \lambda x.\lambda y.((x \ T) \ y)$
- Interpretation is consistent with predicate logic:

$$\begin{aligned}\text{not } T &\Rightarrow_{\beta} (\lambda x.((x \ F) \ T)) \ T \Rightarrow_{\beta} ((T \ F) \ T) \Rightarrow_{\beta} F && \text{因为T是选第一个 第一个是F} \\ \text{not } F &\Rightarrow_{\beta} (\lambda x.((x \ F) \ T)) \ F \Rightarrow_{\beta} ((F \ F) \ T) \Rightarrow_{\beta} T && \text{因为F是选第二个 第二个是T}\end{aligned}$$

λ -calculus can model everything

- Integers

$$0 \equiv \lambda f. \lambda c. c$$

$$1 \equiv \lambda f. \lambda c. (f c) \quad \text{successor}(0)$$

$$2 \equiv \lambda f. \lambda c. (f(f c)) \quad \text{successor(successor(0))}$$

$$3 \equiv \lambda f. \lambda c. (f(f(f c)))$$

...

$$N \equiv \lambda f. \lambda c. (\underbrace{f(f \dots f c)}_N \dots)$$

- Interpretation:

- c is the zero element
- f is the successor function

λ -calculus can model everything

- Integers (cont'd)
- Example calculations:

$$(N\ a) = (\lambda f.\lambda c.(\underbrace{f\dots(f\ c)}_N)\dots))\ a \xrightarrow{\beta} \lambda c.(\underbrace{a\dots(a\ c)}_N\dots)$$

$$((N\ a)\ b) = (\underbrace{a\ (a\dots(a\ b))\dots}_N)$$

λ -calculus can model everything

- Integer operations

- Addition: $+$ $\equiv \lambda M. \lambda N. \lambda a. \lambda b. ((M a) ((N a) b))$

$$[M + N] = \lambda a. \lambda b. ((M a) ((N a) b)) \xrightarrow{\beta^*} \lambda a. \lambda b. (a \underbrace{(a \dots (a}_{M+N} b)) \dots)$$

- Multiplication: \times $\equiv \lambda M. \lambda N. \lambda a. (M (N a))$

$$[M \times N] = \lambda a. (M (N a)) \xrightarrow{\beta^*} \lambda a. \lambda b. (a \underbrace{(a \dots (a}_{M \times N} b)) \dots)$$

- Exponentiation: \wedge $\equiv \lambda M. \lambda N. (N M)$

$$[M^N] = (N \ M) \xrightarrow{\beta^*} \lambda a. \lambda b. (a \underbrace{(a \dots (a}_{M^N} b)) \dots)$$

- This way we can develop all computable math. functions.

λ -calculus can model everything

- Control flow
- $\text{if } \equiv \lambda c. \lambda t. \lambda e. c\ t\ e$
 - Interpretation: $c = \text{condition}$, $t = \text{then}$, $e = \text{else}$
- $\text{if T 3 4} = (\lambda c. \lambda t. \lambda e. c\ t\ e)(\lambda x. \lambda y. x) \ 3\ 4$
$$\Rightarrow_{\beta}^{*} (\lambda t. \lambda e. t) \ 3\ 4$$
$$\Rightarrow_{\beta}^{*} 3$$
- $\text{if F 3 4} = (\lambda c. \lambda t. \lambda e. c\ t\ e)(\lambda x. \lambda y. y) \ 3\ 4$
$$\Rightarrow_{\beta}^{*} (\lambda t. \lambda e. e) \ 3\ 4$$
$$\Rightarrow_{\beta}^{*} 4$$

λ -calculus can model everything

If True x y

y-> if True x y

- Recursion
 - greatest common divisor
- $\text{gcd} = \lambda a. \lambda b. (\text{if } (\text{equal } a \ b) \ a \ (\text{if } (\text{greater } a \ b) \ (\text{gcd } (\text{minus } a \ b) \ b) \ (\text{gcd } (\text{minus } b \ a) \ a)))$
- This is not a definition because gcd appears in both sides
 - If we substitute this, the definition only gets bigger
- To obtain a real definition, we rewrite using β -abstraction:
$$\text{gcd} = (\lambda g. \lambda a. \lambda b. (\text{if } (\text{equal } a \ b) \ a \ (\text{if } (\text{greater } a \ b) \ (g \ (\text{minus } a \ b) \ b) \ (g \ (\text{minus } b \ a) \ a)))) \underline{\text{gcd}}$$
- we obtain the equation
$$\underline{\text{gcd}} = f \underline{\text{gcd}}, \text{ where}$$

$$\underline{f} = \lambda g. \lambda a. \lambda b. (\text{if } (\text{equal } a \ b) \ a \ (\text{if } (\text{greater } a \ b) \ (g \ (\text{minus } a \ b) \ b) \ (g \ (\text{minus } b \ a) \ a)))$$
- $\underline{\text{gcd}}$ is a fixed point of f

λ -calculus can model everything

- Define the *fixed point combinator*:

$$\text{Y} \equiv \underline{\lambda h.(\lambda x.h(x\ x))\ (\lambda x.h(x\ x))}$$

- $\text{Y}\ f$ is a fixed point of f
 - if the normal order evaluation of $\text{Y}\ f$ terminates then $f(\text{Y}\ f)$ and $\text{Y}\ f$ will reduce to the same normal form
- We get then a good definition for gcd:

$$\text{gcd} \equiv \text{Y}\ f = (\lambda h.(\lambda x.h(x\ x))\ (\lambda x.h(x\ x)))\ (\lambda g.\lambda a.\lambda b.(\text{if } (\text{equal } a\ b)\ a\ (\text{if } (\text{greater } a\ b)\ (g\ (\text{minus } a\ b)\ b)\ (g\ (\text{minus } b\ a)\ a)))))$$

λ -calculus can model everything

- Example

$$\text{gcd } 2 \ 4$$

$$\equiv Y f 2 \ 4$$

$$\equiv ((\lambda h.(\lambda x.h(x\ x)))\ (\lambda x.h(x\ x)))\ f\ 2 \ 4$$

$$\Rightarrow_{\beta} ((\lambda x.f(x\ x))\ (\lambda x.f(x\ x)))\ 2 \ 4$$

$$\equiv (f((\lambda x.f(x\ x))\ (\lambda x.f(x\ x))))\ 2 \ 4$$

denote $k \equiv \lambda x.f(x\ x)$

$$\Rightarrow_{\beta} (f(k\ k))\ 2 \ 4$$

$$\equiv ((\lambda g.\lambda a.\lambda b.(\text{if } (= a\ b)\ a\ (\text{if } (> a\ b)\ (g\ (- a\ b)\ b)\ (g\ (- b\ a)\ a))))(k\ k))\ 2 \ 4$$

$$\Rightarrow_{\beta} (\lambda a.\lambda b.(\text{if } (= a\ b)\ a\ (\text{if } (> a\ b)\ ((k\ k)\ (- a\ b)\ b)\ ((k\ k)(- b\ a)\ a))))\ 2 \ 4$$

$$\Rightarrow_{\beta}^{*} \text{if } (= 2\ 4)\ 2\ (\text{if } (> 2\ 4)\ ((k\ k)\ (- 2\ 4)\ 4)\ ((k\ k)\ (- 4\ 2)\ 2))$$

$$\equiv (\lambda c.\lambda t.\lambda e.c\ t\ e)\ (= 2\ 4)\ 2\ (\text{if } (> 2\ 4)\ ((k\ k)\ (- 2\ 4)\ 4)\ ((k\ k)\ (- 4\ 2)\ 2))$$

λ -calculus can model everything

$$\equiv (\lambda c.\lambda t.\lambda e.c\ t\ e) (= 2\ 4)\ 2\ (\text{if } (> 2\ 4) ((k\ k)(- 2\ 4)\ 4) ((k\ k)(- 4\ 2)\ 2))$$

$$\Rightarrow_{\beta}^* (= 2\ 4)\ 2\ (\text{if } (> 2\ 4) ((k\ k)(- 2\ 4)\ 4) ((k\ k)(- 4\ 2)\ 2))$$

$$\Rightarrow_{\delta} F\ 2\ (\text{if } (> 2\ 4) ((k\ k)(- 2\ 4)\ 4) ((k\ k)(- 4\ 2)\ 2))$$

$$\equiv (\lambda x.\lambda y.y)\ 2\ (\text{if } (> 2\ 4) ((k\ k)(- 2\ 4)\ 4) ((k\ k)(- 4\ 2)\ 2))$$

$$\Rightarrow_{\beta}^* \text{if } (> 2\ 4) ((k\ k)(- 2\ 4)\ 4) ((k\ k)(- 4\ 2)\ 2)$$

$$\Rightarrow_{\beta} \dots$$

$$\Rightarrow_{\beta} (k\ k)(- 4\ 2)\ 2$$

$$\equiv ((\lambda x.f(x\ x))\ k)(- 4\ 2)\ 2$$

$$\Rightarrow_{\beta} (f(k\ k))(- 4\ 2)\ 2$$

$$\equiv ((\lambda g.\lambda a.\lambda b.(\text{if } (= a\ b)\ a\ (\text{if } (> a\ b)\ (g\ (- a\ b)\ b)\ (g\ (- b\ a)\ a))))\ (k\ k))(- 4\ 2)\ 2$$

$$\Rightarrow_{\beta} (\lambda a.\lambda b.(\text{if } (= a\ b)\ a\ (\text{if } (> a\ b)\ ((k\ k)(- a\ b)\ b)\ ((k\ k)(- b\ a)\ a))))(- 4\ 2)\ 2$$

λ -calculus can model everything

$$\begin{aligned}&\Rightarrow_{\beta} (\lambda a. \lambda b. (\text{if } (= a b) a (\text{if } (> a b) ((k k) (- a b) b) ((k k) (- b a) a))))(- 4 2) 2 \\&\Rightarrow_{\beta}^{*} \text{if } (= (- 4 2) 2) (- 4 2) (\text{if } (> (- 4 2) 2) ((k k) (- (- 4 2) 2) 2) ((k k) (- 2 (- 4 2)) (- 4 2))) \\&\equiv (\lambda c. \lambda t. \lambda e. c t e) (= (- 4 2) 2) (- 4 2) (\text{if } (> (- 4 2) 2) ((k k) (- (- 4 2) 2) 2) ((k k) (- 2 (- 4 2)) (- 4 2))) \\&\Rightarrow_{\beta}^{*} (= (- 4 2) 2) (- 4 2) (\text{if } (> (- 4 2) 2) ((k k) (- (- 4 2) 2) 2) ((k k) (- 2 (- 4 2)) (- 4 2))) \\&\Rightarrow_{\delta} (= 2 2) (- 4 2) (\text{if } (> (- 4 2) 2) ((k k) (- (- 4 2) 2) 2) ((k k) (- 2 (- 4 2)) (- 4 2))) \\&\Rightarrow_{\delta} \text{T} (- 4 2) (\text{if } (> (- 4 2) 2) ((k k) (- (- 4 2) 2) 2) ((k k) (- 2 (- 4 2)) (- 4 2))) \\&\equiv (\lambda x. \lambda y. x) (- 4 2) (\text{if } (> (- 4 2) 2) ((k k) (- (- 4 2) 2) 2) ((k k) (- 2 (- 4 2)) (- 4 2))) \\&\Rightarrow_{\beta}^{*} (- 4 2) \\&\Rightarrow_{\delta} 2\end{aligned}$$

λ -calculus can model everything

- Structures
- $\text{select_first} \equiv \lambda x. \lambda y. x$
- $\text{select_second} \equiv \lambda x. \lambda y. y$
- $\text{cons} \equiv \lambda a. \lambda d. \lambda x. x\ a\ d$
- $\text{car} \equiv \lambda l. l\ \text{select_first}$
- $\text{cdr} \equiv \lambda l. l\ \text{select_second}$
- $\text{null?} \equiv \lambda l. l(\lambda x. \lambda y. F)$

λ -calculus can model everything

$\text{car}(\text{cons } A B)$

$\equiv (\lambda l.l \text{ select_first})(\text{cons } A B)$

$\Rightarrow_{\beta} (\text{cons } A B) \text{ select_first}$

$\equiv ((\lambda a.\lambda d.\lambda x.x a d) A B) \text{ select_first}$

$\Rightarrow_{\beta}^{*} (\lambda x.x A B) \text{ select_first}$

$\Rightarrow_{\beta} \text{select_first } A B$

$\equiv (\lambda x.\lambda y.x) A B$

$\Rightarrow_{\beta}^{*} A$

λ -calculus can model everything

$\text{cdr}(\text{cons } A B)$

$\equiv (\lambda l.l \text{ select_second})(\text{cons } A B)$

$\Rightarrow_{\beta} (\text{cons } A B) \text{ select_second}$

$\equiv ((\lambda a.\lambda d.\lambda x.x a d) A B) \text{ select_second}$

$\Rightarrow_{\beta}^{*} (\lambda x.x A B) \text{ select_second}$

$\Rightarrow_{\beta} \text{select_second } A B$

$\equiv (\lambda x.\lambda y.x) A B$

$\Rightarrow_{\beta}^{*} B$

λ -calculus can model everything

$\text{null?}(\text{cons } A B)$

$\equiv (\lambda l.l(\lambda x.\lambda y.\text{select_second}))(\text{cons } A B)$

$\Rightarrow_{\beta} (\text{cons } A B)(\lambda x.\lambda y.\text{select_second})$

$\equiv ((\lambda a.\lambda d.\lambda x.x\ a\ d)\ A\ B)(\lambda x.\lambda y.\text{select_second})$

$\Rightarrow_{\beta}^{*} (\lambda x.x\ A\ B)(\lambda x.\lambda y.\text{select_second})$

$\Rightarrow_{\beta} (\lambda x.\lambda y.\text{select_second})\ A\ B$

$\Rightarrow_{\beta}^{*} \text{select_second}$

$\equiv F$