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*: $x, y, z, u, v, \ldots$
  - *abstraction*: $\lambda x.M$
    - 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 $x{*}x$
  - $(\lambda\ x.x{*}x)\ 4$ - the same function applied to 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:

    $\lambda x_1 x_2 \ldots x_n.e$ means $\lambda x_1.(\lambda x_2.(\ldots(\lambda x_n.e)\ldots))$

- Example:

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

λ-calculus

- Context-free grammars (CFG)

- CFG for λ-expressions

  $$expr \;\rightarrow\; \mathtt{name} \mid (\,\lambda\, \mathtt{name}\,.\, expr\,) \mid (\, expr \;\; expr\,)$$

- CFG for λ-expressions with minimum parentheses

  $$\begin{aligned} expr &\rightarrow \mathtt{name} \mid \lambda\, \mathtt{name}\,.\, expr \mid func \;\; arg \\ func &\rightarrow \mathtt{name} \mid (\,\lambda\, \mathtt{name}\,.\, expr\,) \mid func \;\; arg \\ arg &\rightarrow \mathtt{name} \mid (\,\lambda\, \mathtt{name}\,.\, expr\,) \mid (\, func \;\; arg\,) \end{aligned}$$

λ-calculus

- Examples

$$\texttt{square} = \lambda x.\texttt{times}\; x\; x$$
$$\texttt{identity} = \lambda x.x$$
$$\texttt{const7} = \lambda x.7$$
$$\texttt{hypot} = \lambda x.\lambda y.\texttt{sqrt}\;(\texttt{plus}\;(\texttt{square}\; x)\;(\texttt{square}\; y))$$

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

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

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:

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

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

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:

    $\{N/x\}(\lambda y.P) = \lambda y.\{N/x\}P, \quad y \neq x, y \notin free(N)$
  - *renaming* – y is free in N; y is renamed to z in P:

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

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

Computing with pure λ-terms

- Rewriting rules
- *α-conversion* – renaming the formal parameters

  $\lambda x.M \Rightarrow_{\alpha} \lambda y.\{y/x\}M,\ y \notin \mathit{free}(M)$
- *β-reduction* – applying an abstraction to an argument

  $(\lambda x.M)\ N \Rightarrow_{\beta} \{N/x\}\ M$

Equality of pure λ-terms

- Example

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

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

$\Longrightarrow_\alpha (\underline{\lambda x}yz.x\ z\ (y\ z))\ (\underline{\lambda u.u})\ (\lambda v.v)$

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

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

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

$\Longrightarrow_\beta \lambda z.z\ z$

Equality of pure λ-terms

- Example

$(\underline{\lambda f}gh.f\ g\ (h\ h))\ (\underline{\lambda xy.x})\ h\ (\lambda x.x\ x)$

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

$\Rightarrow_\alpha (\underline{\lambda g}k.(\lambda xy.x)\ g\ (k\ k))\ \underline{h}\ (\lambda x.x\ x)$

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

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

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

$\Rightarrow_\beta h$

Computing with pure λ-terms

- Rewriting rules
- *Reduction*: any sequence of $\Longrightarrow_{\alpha}$, $\Longrightarrow_{\beta}$
- *Normal form*: term that cannot be β-reduced
  - β-normal form
  - Example of normal form

    $\lambda x.x\ x$ – 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

$$
\begin{tikzcd}[column sep=tiny]
 & & (\underline{\lambda x}yz.x\, z\, (y\, z))\, (\underline{\lambda x.x})\, (\lambda x.x) \arrow[d, Rightarrow, "\beta"] & & \\
 & & (\underline{\lambda y}z.(\underline{\lambda x.x)\, z}\, (y\, z))\, (\underline{\lambda x.x}) \arrow[dl, Rightarrow, "\beta"'] \arrow[dr, Rightarrow, "\beta"] & & \\
 & (\underline{\lambda y}z.z\, (y\, z))\, (\underline{\lambda x.x}) \arrow[dr, Rightarrow, "\beta"'] & & \lambda z.\underline{(\lambda x.x)\, z}\, (\underline{(\lambda x.x)\, z}) \arrow[dl, Rightarrow, "\beta"'] \arrow[dr, Rightarrow, "\beta"] & \\
 & & \lambda z.z\, (\underline{(\lambda x.x)\, z}) \arrow[dr, Rightarrow, "\beta"'] & & \lambda z.\underline{(\lambda x.x)\, z}\, z \arrow[dl, Rightarrow, "\beta"] \\
 & & & \lambda z.z\, z &
\end{tikzcd}
$$

Computing with pure λ-terms

- *Nonterminating reductions*
  - Never reach a normal form
  - Example

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

Computing with pure λ-terms

- *Theorem* (*Church-Roser,* 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.

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

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

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

  $(\lambda y.h)\ ((\lambda x.x\ x)\ (\lambda x.x\ x)) \Longrightarrow_\beta h$

- *Theorem* (*Church-Roser,* 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: $\mathtt{T} \equiv \lambda x.\lambda y.x$
  - interpretation: of a pair of values, choose the first
- False: $\mathtt{F} \equiv \lambda x.\lambda y.y$
  - interpretation: of a pair of values, choose the second

- Properties:

$$((\mathtt{T}\ \ P)\ Q) \Longrightarrow_\beta (((\lambda x.\lambda y.x)\ P)\ Q) \Longrightarrow_\beta ((\lambda y.P)\ Q) \Longrightarrow_\beta P$$
$$((\mathtt{F}\ \ P)\ Q) \Longrightarrow_\beta (((\lambda x.\lambda y.y)\ P)\ Q) \Longrightarrow_\beta ((\lambda y.y)\ Q) \Longrightarrow_\beta Q$$

λ-calculus can model everything

- Boolean functions
- $\texttt{not} \equiv \lambda x.((x\ \texttt{F})\ \texttt{T})$
- $\texttt{and} \equiv \lambda x.\lambda y.((x\ y)\ \texttt{F})$
- $\texttt{or} \equiv \lambda x.\lambda y.((x\ \texttt{T})\ y)$

- Interpretation is consistent with predicate logic:

$$\texttt{not T} \Rightarrow_\beta (\lambda x.((x\ \texttt{F})\ \texttt{T}))\ \texttt{T} \Rightarrow_\beta ((\texttt{T F})\ \texttt{T}) \Rightarrow_\beta \texttt{F}$$

$$\texttt{not F} \Rightarrow_\beta (\lambda x.((x\ \texttt{F})\ \texttt{T}))\ \texttt{F} \Rightarrow_\beta ((\texttt{F F})\ \texttt{T}) \Rightarrow_\beta \texttt{T}$$

λ-calculus can model everything

- Integers

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

$$1 \equiv \lambda f.\lambda c.(f\,c)$$

$$2 \equiv \lambda f.\lambda c.(f\,(f\,c))$$

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

…

$$\mathrm{N} \equiv \lambda f.\lambda c.\underbrace{(f\,(f\,\ldots(f}_{\mathrm{N}}\,c))\ldots)$$

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

λ-calculus can model everything

- Integers (cont'd)
- Example calculations:

$$(\mathrm{N}\ a) = (\lambda f.\lambda c.\underbrace{(f \ldots (f}_{\mathrm{N}}\ c))\ldots))\ a \Longrightarrow_\beta \lambda c.\underbrace{(a \ldots (a}_{\mathrm{N}}\ c)\ldots)$$

$$((\mathrm{N}\ a)\ b) = (\underbrace{a\ (a \ldots (a}_{\mathrm{N}}\ b))\ldots)$$

λ-calculus can model everything

- Integer operations
- *Addition*: $+ \;\equiv\; \lambda M.\lambda N.\lambda a.\lambda b.((M\ a)((N\ a)\ b))$

  $[\mathtt{M} + \mathtt{N}] = \lambda a.\lambda b.((M\ a)\ ((N\ a)\ b)) \Rightarrow_\beta^* \lambda a.\lambda b.(\underbrace{a\ (a\ldots(a}_{\mathrm{M+N}}\ b))\ldots)$

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

  $[\mathtt{M} \times \mathtt{N}] = \lambda a.(M\ (N\ a)) \Rightarrow_\beta^* \lambda a.\lambda b.(\underbrace{a\ (a\ldots(a}_{\mathrm{M\times N}}\ b))\ldots)$

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

  $[\mathtt{M}^{\mathtt{N}}] = (\mathtt{N}\ \ \mathtt{M}) \Rightarrow_\beta^* \lambda a.\lambda b.(\underbrace{a\ (a\ldots(a}_{\mathrm{M^N}}\ b))\ldots)$

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

λ-calculus can model everything

- Control flow
- `if` $\equiv \lambda c.\lambda t.\lambda e.c\ t\ e$
  - Interpretation: c = condition, t = then, e = else

- `if T 3 4` $= (\lambda c.\lambda t.\lambda e.c\ t\ e)(\lambda x.\lambda y.x)$ `3 4`

  $\Longrightarrow_\beta^* (\lambda t.\lambda e.t)$ `3 4`

  $\Longrightarrow_\beta^*$ `3`
- `if F 3 4` $= (\lambda c.\lambda t.\lambda e.c\ t\ e)(\lambda x.\lambda y.y)$ `3 4`

  $\Longrightarrow_\beta^* (\lambda t.\lambda e.e)$ `3 4`

  $\Longrightarrow_\beta^*$ `4`

λ-calculus can model everything

- Recursion
- $\texttt{gcd} = \lambda a.\lambda b.(\texttt{if}\ (\texttt{equal}\ a\ b)\ a\ (\text{if}\ (\texttt{greater}\ a\ b)\ (\texttt{gcd}\ (\texttt{minus}\ a\ b)\ b)\ (\texttt{gcd}\ (\texttt{minus}\ b\ a)\ a)))$
- 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:
  $\texttt{gcd} = (\lambda g.\lambda a.\lambda b.(\texttt{if}\ (\texttt{equal}\ a\ b)\ a\ (\text{if}\ (\texttt{greater}\ a\ b)\ (g\ (\texttt{minus}\ a\ b)\ b)\ (g\ (\texttt{minus}\ b\ a)\ a))))\ \texttt{gcd}$
- we obtain the equation
  $\texttt{gcd} = f\ \texttt{gcd}$, where
  $f = \lambda g.\lambda a.\lambda b.(\texttt{if}\ (\texttt{equal}\ a\ b)\ a\ (\text{if}\ (\texttt{greater}\ a\ b)\ (g\ (\texttt{minus}\ a\ b)\ b)\ (g\ (\texttt{minus}\ b\ a)\ a)))$
- `gcd` is a fixed point of f

λ-calculus can model everything

- Define the *fixed point combinator:*

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

- $\mathtt{Y}\ f$ is a fixed point of f
  - if the normal order evaluation of $\mathtt{Y}\ f$ terminates then $f\,(\mathtt{Y}\ f)$ and $\mathtt{Y}\ f$ will reduce to the same normal form

- We get then a good definition for gcd:

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

λ-calculus can model everything

- Example

$\texttt{gcd}\ 2\ 4$

$\equiv\ \texttt{Y}\, f\, 2\ 4$

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

$\Longrightarrow_\beta ((\lambda x.f\,(x\ x))\ (\lambda x.f\,(x\ x)))\ 2\ 4$

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

$\Longrightarrow_\beta (f\,(k\ k))\ 2\ 4$

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

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

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

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

λ-calculus can model everything

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

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

$\Longrightarrow_\delta \texttt{F}\ 2\ (\texttt{if}\ (> 2\ 4)\ ((k\ k)\ (-\ 2\ 4)\ 4)\ ((k\ k)\ (-\ 4\ 2)\ 2))$

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

$\Longrightarrow_\beta{}^* \texttt{if}\ (> 2\ 4)\ ((k\ k)\ (-\ 2\ 4)\ 4)\ ((k\ k)\ (-\ 4\ 2)\ 2)$

$\Longrightarrow_\beta \ldots$

$\Longrightarrow_\beta (k\ k)\ (-\ 4\ 2)\ 2$

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

$\Longrightarrow_\beta (f\ (k\ k))(-\ 4\ 2)\ 2$

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

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

λ-calculus can model everything

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

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

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

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

$\Rightarrow_\delta (= 2\ 2)(-\ 4\ 2)\ (\texttt{if}\ (> (-\ 4\ 2)\ 2)\ ((k\ k)(-\ (-\ 4\ 2)\ 2)\ 2)\ ((k\ k)\ (-\ 2\ (-\ 4\ 2))\ (-\ 4\ 2)))$

$\Rightarrow_\delta \texttt{T}\ (-\ 4\ 2)\ (\texttt{if}\ (> (-\ 4\ 2)\ 2)\ ((k\ k)\ (-\ (-\ 4\ 2)\ 2)\ 2)\ ((k\ k)\ (-\ 2\ (-\ 4\ 2))\ (-\ 4\ 2)))$

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

$\Rightarrow_\beta^* (-\ 4\ 2)$

$\Rightarrow_\delta 2$

λ-calculus can model everything

- Structures
- `select_first` ≡ $\lambda x.\lambda y.x$
- `select_second` ≡ $\lambda x.\lambda y.y$
- `cons` ≡ $\lambda a.\lambda d.\lambda x.x\ a\ d$
- `car` ≡ $\lambda l.l$ `select_first`
- `cdr` ≡ $\lambda l.l$ `select_second`
- `null?` ≡ $\lambda l.l\ (\lambda x.\lambda y.$`F`$)$

λ-calculus can model everything

$\texttt{car}\,(\texttt{cons}\; A\; B)$

$\equiv (\lambda l.l\; \texttt{select\_first})\,(\texttt{cons}\; A\; B)$

$\Longrightarrow_\beta (\texttt{cons}\; A\; B)\; \texttt{select\_first}$

$\equiv ((\lambda a.\lambda d.\lambda x.x\; a\; d)\; A\; B)\; \texttt{select\_first}$

$\Longrightarrow_\beta^* (\lambda x.x\; A\; B)\; \texttt{select\_first}$

$\Longrightarrow_\beta \texttt{select\_first}\; A\; B$

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

$\Longrightarrow_\beta^* A$

λ-calculus can model everything

`cdr` (`cons` A B)

$\equiv (\lambda l.l$ `select_second`$)$ (`cons` A B)

$\Rightarrow_\beta$ (`cons` A B) `select_second`

$\equiv ((\lambda a.\lambda d.\lambda x.x\ a\ d)\ A\ B)$ `select_second`

$\Rightarrow_\beta^*$ $(\lambda x.x\ A\ B)$ `select_second`

$\Rightarrow_\beta$ `select_second` A B

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

$\Rightarrow_\beta^*\ B$

λ-calculus can model everything

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

$\equiv (\lambda l.l\ (\lambda x.\lambda y.\texttt{select\_second}))\ (\texttt{cons}\ A\ B)$

$\Rightarrow_{\beta} (\texttt{cons}\ A\ B)\ (\lambda x.\lambda y.\texttt{select\_second})$

$\equiv ((\lambda a.\lambda d.\lambda x.x\ a\ d)\ A\ B)\ (\lambda x.\lambda y.\texttt{select\_second})$

$\Rightarrow_{\beta}^{*} (\lambda x.x\ A\ B)\ (\lambda x.\lambda y.\texttt{select\_second})$

$\Rightarrow_{\beta} (\lambda x.\lambda y.\texttt{select\_second})\ A\ B$

$\Rightarrow_{\beta}^{*} \texttt{select\_second}$

$\equiv \texttt{F}$