

Some λ -calculus stuff:

η -extension

\rightarrow derives from the semantic equivalence

$$(e_0 e_1) = (\lambda u (e_0 u) e_1) ,$$

(u must be a fresh variable that does not occur free in e_0)

Some λ -calculus stuff:

η -extension

\rightarrow derives from the semantic equivalence

$$(e_0 e_1) = (\lambda u (e_0 u) e_1) ,$$

(u must be a fresh variable that does not occur free in e_0)

\rightarrow in deBruijn notation, we have

$$(e_0 e_1) = (\Lambda (e_0^{(+1)} \#0) e_1) ,$$

($e_0^{(+1)}$ denotes the addition of 1 to all free occurrences of deBruijn indices in e_0)

Some λ -calculus stuff:

η -extension

→ derives from the semantic equivalence

$$(e_0 e_1) = (\lambda u (e_0 u) e_1) ,$$

(u must be a fresh variable that does not occur free in e_0)

→ in deBruijn notation, we have

$$(e_0 e_1) = (\Lambda (e_0^{(+1)} \#0) e_1) ,$$

($e_0^{(+1)}$ denotes the addition of 1 to all free occurrences of deBruijn indices in e_0)

→ It implies that

$$e_0 = \Lambda (e_0^{(+1)} \#0)$$

Some λ -calculus stuff:

η -extension

→ derives from the semantic equivalence

$$(e_0 e_1) = (\lambda u (e_0 u) e_1) ,$$

(u must be a fresh variable that does not occur free in e_0)

→ in deBruijn notation, we have

$$(e_0 e_1) = (\Lambda (e_0^{(+1)} \#0) e_1) ,$$

($e_0^{(+1)}$ denotes the addition of 1 to all free occurrences of deBruijn indices in e_0)

→ It implies that

$$e_0 = \Lambda (e_0^{(+1)} \#0)$$

or, more generally,

$$e = \underbrace{\Lambda \dots \Lambda}_n (\dots (e^{(+n)} \#(n-1)) \dots \#0) .$$

Some λ -calculus stuff:

η -extension

→ derives from the semantic equivalence

$$(e_0 e_1) = (\lambda u (e_0 u) e_1) ,$$

(u must be a fresh variable that does not occur free in e_0)

→ in deBruijn notation, we have

$$(e_0 e_1) = (\Lambda (e_0^{(+1)} \#0) e_1) ,$$

($e_0^{(+1)}$ denotes the addition of 1 to all free occurrences of deBruijn indices in e_0)

→ It implies that

$$e_0 = \Lambda (e_0^{(+1)} \#0)$$

or, more generally,

$$e = \underbrace{\Lambda \dots \Lambda}_n (\dots (e^{(+n)} \#(n-1)) \dots \#0) .$$

→ η -extending partial applications

$$\begin{aligned}
 & \underbrace{(\dots)}_k \left(\underbrace{\Lambda \dots \Lambda}_n e_b e_{k-1} \right) \dots e_0 = \\
 & \underbrace{\Lambda \dots \Lambda}_{n-k} \left(\underbrace{(\dots)}_{n-k} \left(\underbrace{(\dots)}_k \underbrace{\Lambda \dots \Lambda}_n e_b e_{k-1} \right) \dots e_0 \right)^{+(n-k)} \\
 & \qquad \qquad \qquad \qquad \qquad \qquad \qquad \qquad \qquad \#(n-k-1) \dots \#0)
 \end{aligned}$$

→ head forms

$$h \mid t =_s \#i \mid \underbrace{\Lambda \dots \Lambda}_n \underbrace{(\dots (h t_1) \dots t_r)}_r$$

$$h \mid t =_s \#i \mid \underbrace{\Lambda \dots \Lambda}_n \underbrace{@ \dots @}_r h t_1 \dots t_r$$

$$r \geq 0, n \geq 0$$

h denotes a **head expression**

$t_1 \dots t_r$ denotes **tail expressions**

→ head forms

$$h \mid t =_s \#i \mid \underbrace{\Lambda \dots \Lambda}_n \underbrace{(\dots (h t_1) \dots t_r)}_r$$

$$h \mid t =_s \#i \mid \underbrace{\Lambda \dots \Lambda}_n \underbrace{@ \dots @}_r h t_1 \dots t_r$$

$$r \geq 0, n \geq 0$$

h denotes a **head expression**

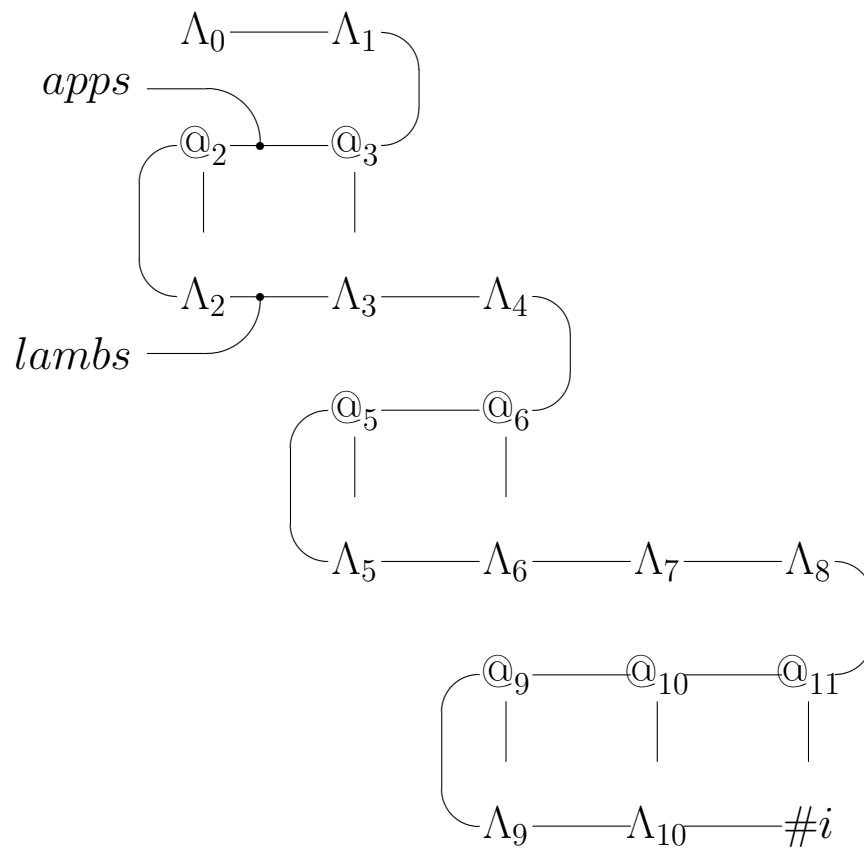
$t_1 \dots t_r$ denotes **tail expressions**

→ head normal forms

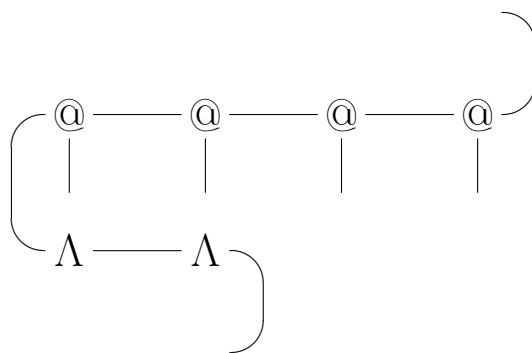
$$hnf = \underbrace{\Lambda \dots \Lambda}_n \underbrace{(\dots (\#i t_1) \dots t_r)}_r$$

$$hnf = \underbrace{\Lambda \dots \Lambda}_n \underbrace{@ \dots @}_r \#i t_1 \dots t_r$$

Typical head form of a λ -expression

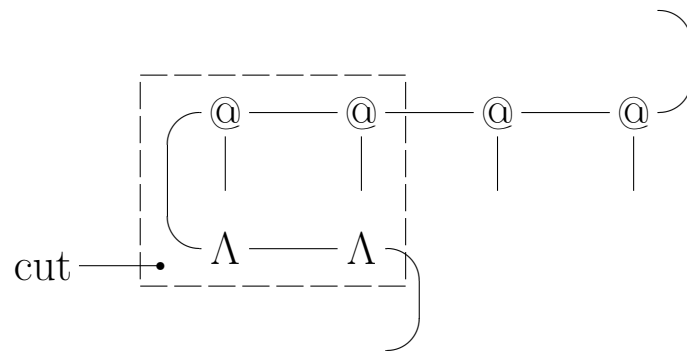


Taking **cuts** out of lefthand corners



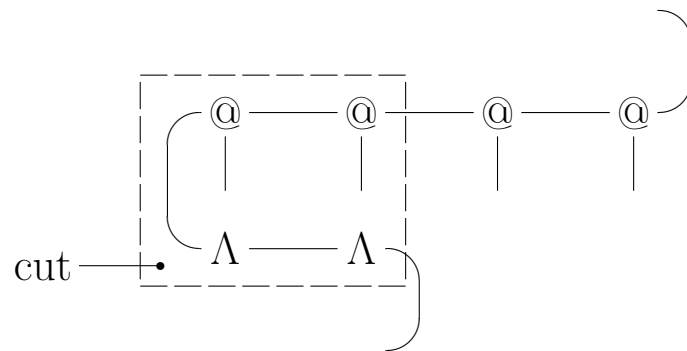
(a) *apps* longer than *lambds*

Taking **cuts** out of lefthand corners

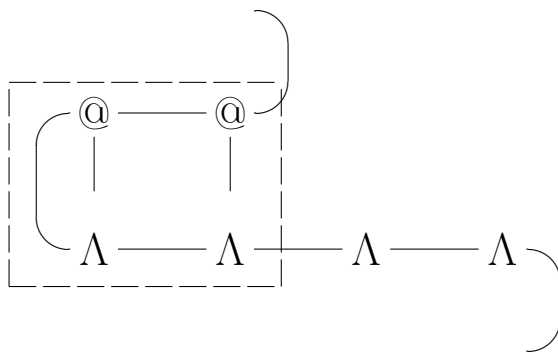


(a) *apps* longer than *lambds*

Taking **cuts** out of lefthand corners

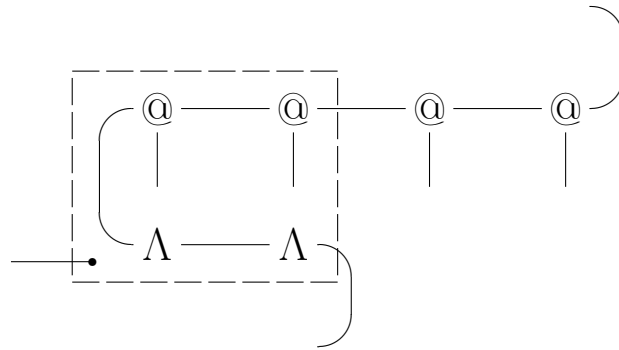


(a) *apps* longer than *lambds*

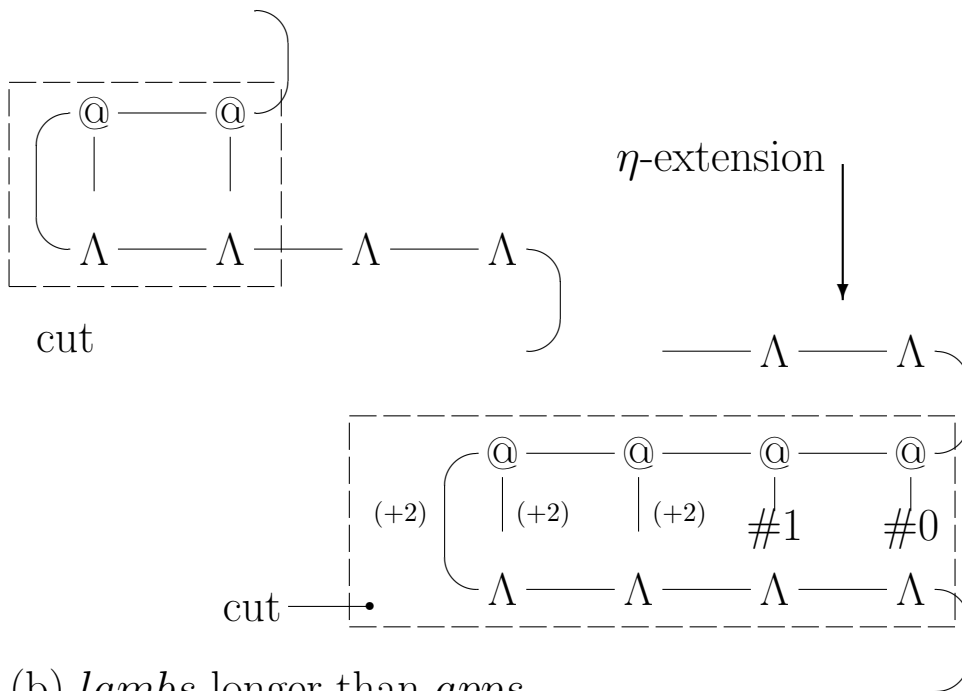


(b) *lambds* longer than *apps*

Taking **cuts** out of lefthand corners

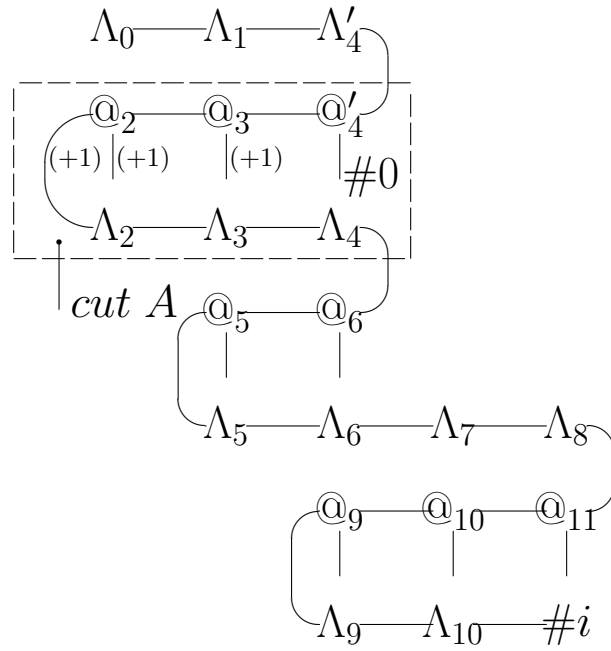


(a) *apps* longer than *lambds*

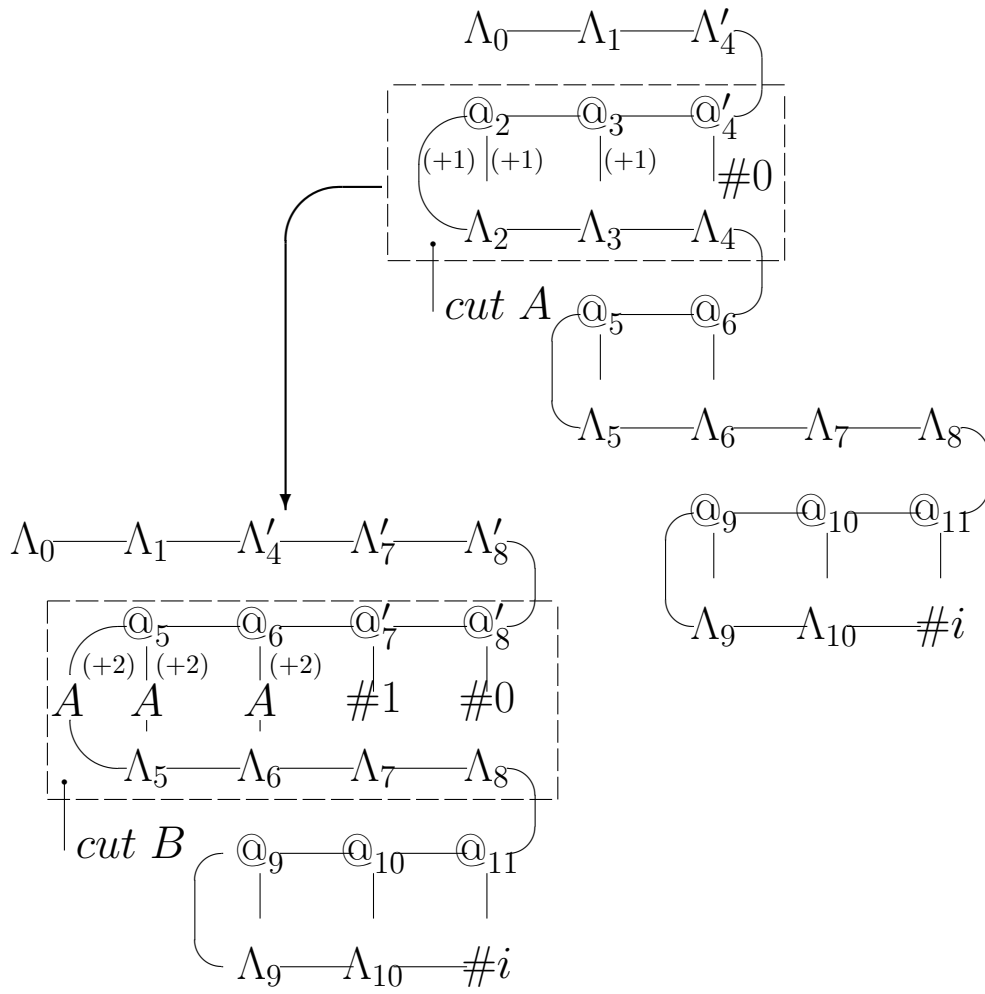


(b) *lambds* longer than *apps*

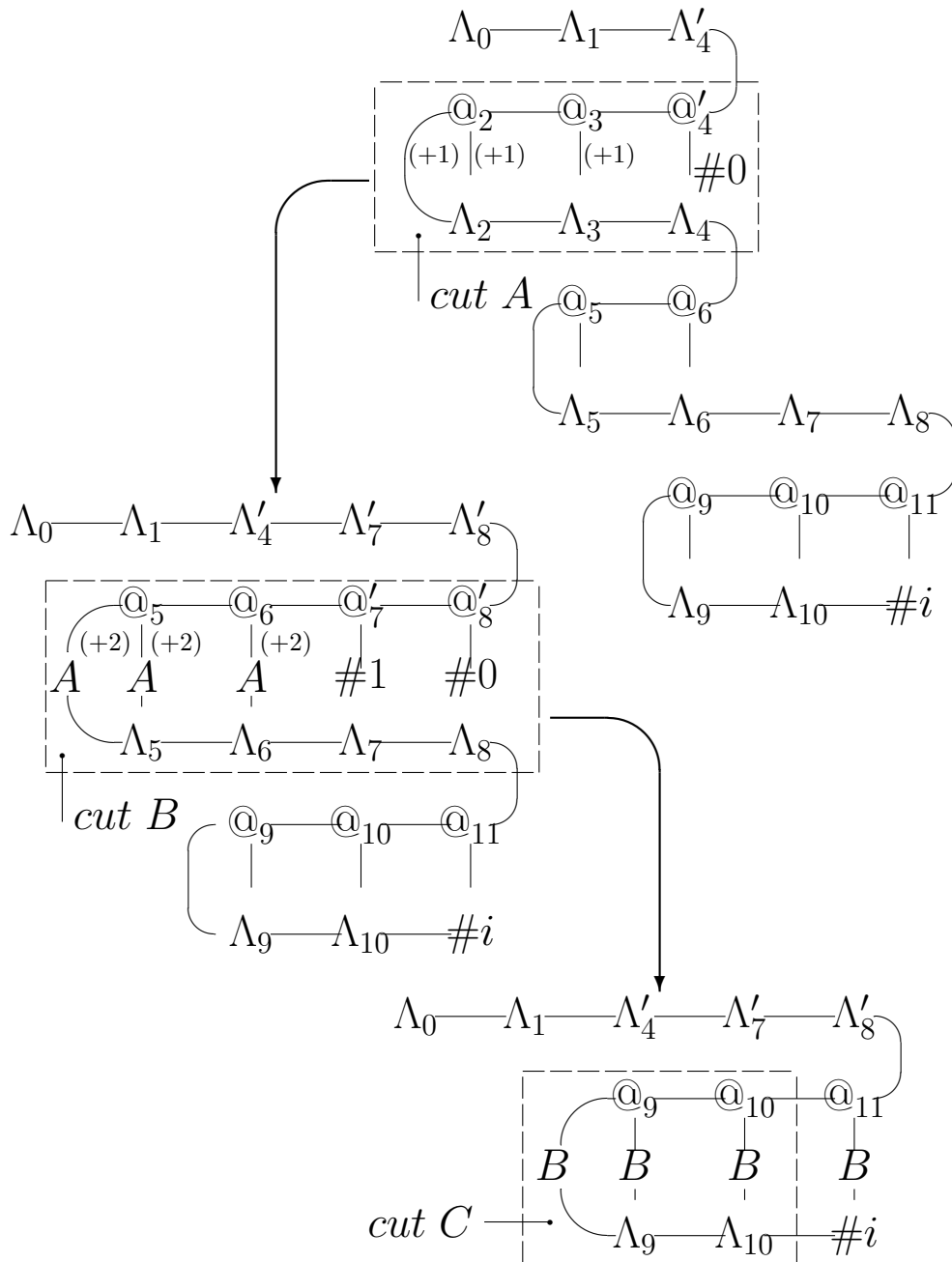
Distributing cuts over the branches of the spine



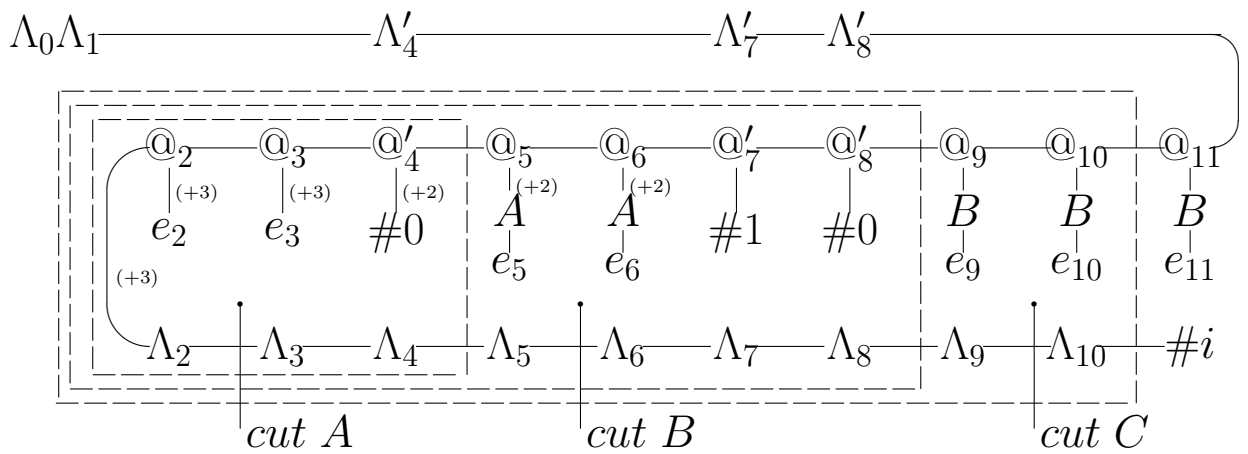
Distributing cuts over the branches of the spine



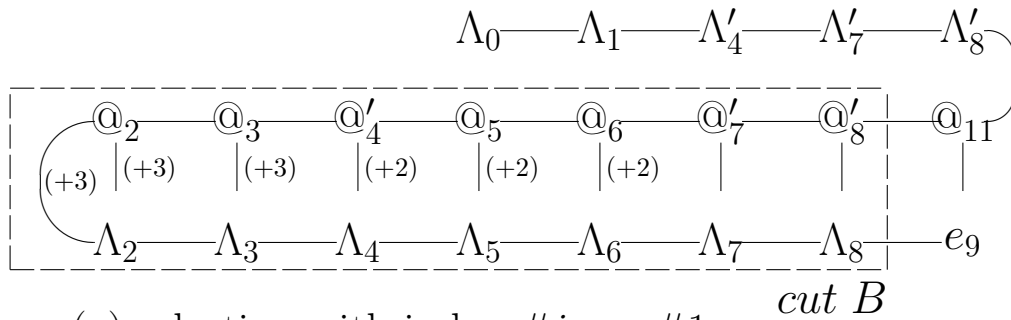
Distributing cuts over the branches of the spine



The head form after having completed all β -distributions and expanded all cuts along the spine

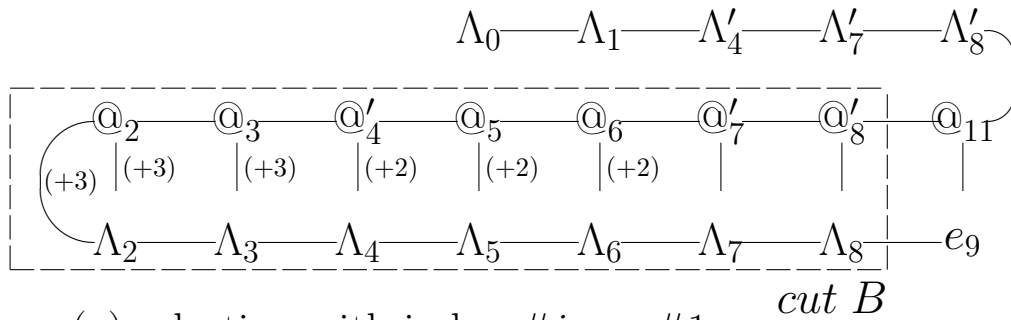


Continuing with substitutions in the head

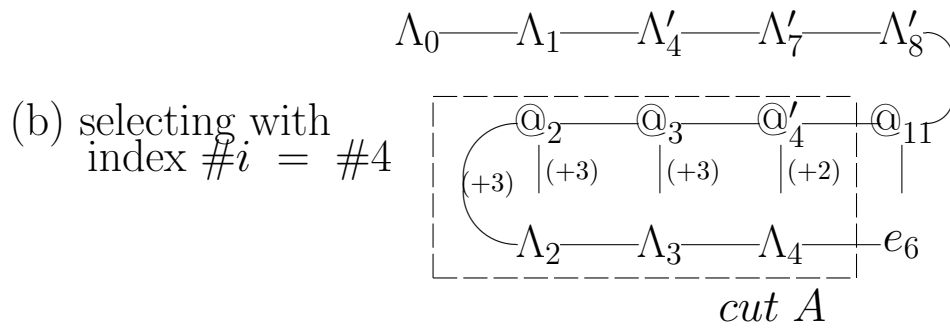


(a) selecting with index $\#i = \#1$

Continuing with substitutions in the head

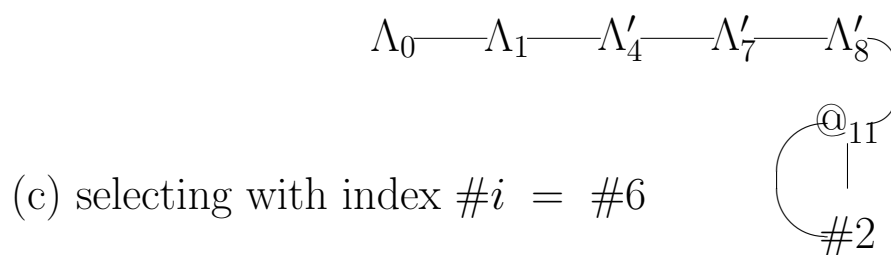
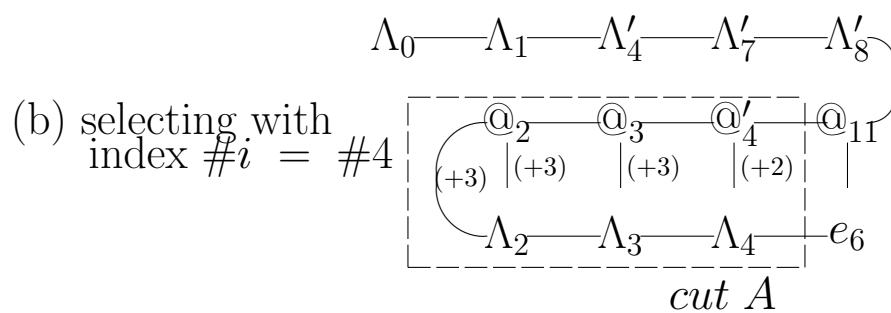
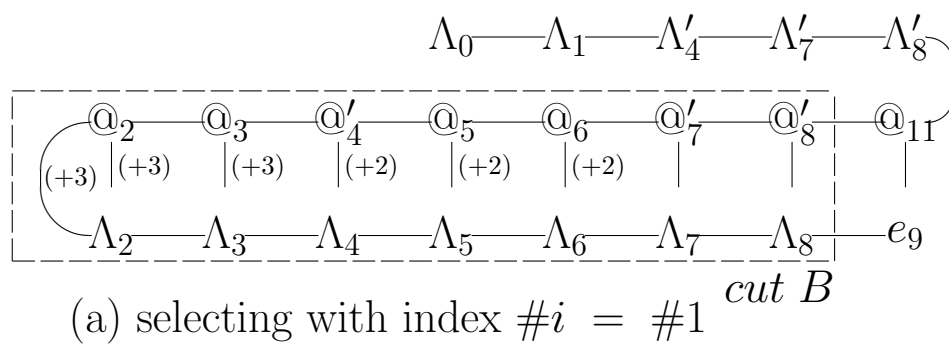


(a) selecting with index $\#i = \#1$



(b) selecting with index $\#i = \#4$

Continuing with substitutions in the head



A fully head-normalizing SECD-machine

Substituting deBruijn indices in head positions

$$(2a) (S, v : E, \#(j + 1) : C, D, u) \rightarrow (S, E, \#j : C, D, u)$$

$$(2b) (S, [E' e] : E, \#0 : C, D, u)$$

$$\rightarrow (S, E', e : nil, (E, C, D, u), u)$$

$$(2c) (S, u' : E, \#0 : C, D, u) \rightarrow (\#(u - u') : S, E, C, D, u)$$

Rearranging applications on C

$$(3) (S, E, @ e_f e_a : C, D, u) \rightarrow ([E e_a] : S, E, e_f : @ : C, D, u)$$

Entering naive β -reductions

$$(5a) (e_a : S, E, \Lambda e_b : @ : C, D, u)$$

$$\rightarrow (S, e_a : E, e_b : nil, (E, C, D, u), u)$$

Dealing with unapplied abstractions in C

$$(4) (S, E, \Lambda e_b : C, D, u)$$

$$\rightarrow (S, (u + 1) : E, e_b : \Lambda : nil, (E, C, D, u), (u + 1))$$

Reducing closures on S

$$(5b) ([E' e'] : S, E, C, D, u) \rightarrow (S, E', e' : nil, (E, C, D, u), u)$$

Dealing with abstractions on S and apply nodes on C

$$(7) (\Lambda e_b : S, E, @ : C, D, u) \rightarrow (S, E, \Lambda e_b : @ : C, D, u)$$

Reconstructing abstractions in S

$$(8) (e_b : S, E, \Lambda : nil, (E', C', D', u'), u)$$

$$\rightarrow (\Lambda e_b : S, E', C', D', u')$$

Reconstructing irreducible applications in S

$$(6) (e_b : e_a : S, E, @ : C, D, u) \rightarrow (@ e_b e_a : S, E, C, D, u)$$

Rules that effect reducing tail closures

$$(9) \quad (e_b : e_a : S, E, @ : C, D, u) \mid e_a = [E' e'_a] \\ \rightarrow ([E' e'_a] : e_b : S, E, @^{\leftrightarrow} : C, D, u)$$

$$(10) \quad (e_a : e_b : S, E, @^{\leftrightarrow} : C, D, u) \mid e_a \neq [E' e'_a] \\ \rightarrow (@ e_b e_a : S, E, C, D, u)$$

(these rules must be squeezed in between rules (7) and (8))

Additional state transition rules for an applied λ -calculus

$e =_s \#i \mid pf \mid const \mid \langle e_0 e_1 \dots e_n \rangle \mid \Lambda e_b \mid @ e_f e_a \mid [E e]$

$\rightarrow pf \in \{+, -, \dots, \wedge, \vee, \dots, head, tail, \dots\}$

\rightarrow

$const = numbers, booleans, chars, free_vars, \dots$

Additional state transition rules for an applied λ -calculus

$e =_s \#i \mid pf \mid const \mid \langle e_0 e_1 \dots e_n \rangle \mid \Lambda e_b \mid @ e_f e_a \mid [E e]$

$\rightarrow pf \in \{+, -, \dots, \wedge, \vee, \dots, head, tail, \dots\}$

\rightarrow

$const = numbers, booleans, chars, free_vars, \dots$

examples:

$(num : S, E, + : @ : C, D, u)$

$\rightarrow (S, E, \{+ num\} : C, D, u)$

$(num_2 : S, E, \{+ num_1\} : @ : C, D, u)$

$\rightarrow ((num_2 + num_1) : S, E, C, D, u)$

Additional state transition rules for an applied λ -calculus

$e =_s \#i \mid pf \mid const \mid \langle e_0 e_1 \dots e_n \rangle \mid \Lambda e_b \mid @ e_f e_a \mid [E e]$

$\rightarrow pf \in \{+, -, \dots, \wedge, \vee, \dots, head, tail, \dots\}$

\rightarrow

$const = numbers, booleans, chars, free_vars, \dots$

examples:

$(num : S, E, + : @ : C, D, u)$

$\rightarrow (S, E, \{+ num\} : C, D, u)$

$(num_2 : S, E, \{+ num_1\} : @ : C, D, u)$

$\rightarrow ((num_2 + num_1) : S, E, C, D, u)$

$([E' e'] : S, E, + : @ : C, D, u)$

$\rightarrow (S, E', e' : nil, (E, + : @ : C, D, u), u)$

$([E' e'] : S, E, \{+ num\} : @ : C, D, u)$

$\rightarrow (S, E', e' : nil, (E, \{+ num\} : @ : C, D, u), u)$

Additional state transition rules for an applied λ -calculus

$e =_s \#i \mid pf \mid const \mid \langle e_0 e_1 \dots e_n \rangle \mid \Lambda e_b \mid @ e_f e_a \mid [E e]$

$\rightarrow pf \in \{+, -, \dots, \wedge, \vee, \dots, head, tail, \dots\}$

\rightarrow

$const = numbers, booleans, chars, free_vars, \dots$

examples:

$(num : S, E, + : @ : C, D, u)$
 $\rightarrow (S, E, \{+ num\} : C, D, u)$

$(num_2 : S, E, \{+ num_1\} : @ : C, D, u)$
 $\rightarrow ((num_2 + num_1) : S, E, C, D, u)$

$([E' e'] : S, E, + : @ : C, D, u)$
 $\rightarrow (S, E', e' : nil, (E, + : @ : C, D, u), u)$

$([E' e'] : S, E, \{+ num\} : @ : C, D, u)$
 $\rightarrow (S, E', e' : nil, (E, \{+ num\} : @ : C, D, u), u)$

$([E' \langle e_0 e_1 \dots e_n \rangle] : S, E, head : @ : C, D, u)$
 $\rightarrow ([E' e_0] : S, E, C, D, u)$

$([E' \langle e_0 e_1 \dots e_n \rangle] : S, E, tail : @ : C, D, u)$
 $\rightarrow ([E' \langle e_1 \dots e_n \rangle] : S, E, C, D, u)$

Another fully normalizing head-order reducer

$$\tau_{hor} : (S, E, T, u, dir) \rightarrow (S', E', T', u, dir)$$

- (1) $(S, E, @ e_0 e_1, u, \downarrow) \rightarrow ([E e_1] : S, E, e_0, u, \downarrow)$
- (2) $([E' e'] : S, E, \Lambda e, u, \downarrow) \rightarrow (S, [E' e'] : E, e, u, \downarrow)$
- (3) $(S, E, \Lambda e, u, \downarrow) \rightarrow (\Lambda : S, (u + 1) : E, e, u + 1, \downarrow)$
- (4) $(S, v : E, \#(i + 1), u, \downarrow) \rightarrow (S, E, \#i, u, \downarrow)$
- (5) $(S, [E' e] : E, \#0, u, \downarrow) \rightarrow (S, E', e, u, \downarrow)$
- (6) $(S, u' : E, \#0, u, \downarrow) \rightarrow (S, -, \#(u - u'), u, \uparrow)$
- (7) $(\Lambda : S, -, e, u, \uparrow) \rightarrow (S, -, \Lambda e, u - 1, \uparrow)$
- (8) $(@ : e_0 : S, -, e_1, u, \uparrow) \rightarrow (S, -, @ e_0 e_1, u, \uparrow)$
- (9) $([E' e_1] : S, -, e_0, u, \uparrow) \rightarrow (@ : e_0 : S, E', e_1, u, \downarrow)$
- (10) $(nil, -, e, u, \uparrow) \rightarrow (-, -, e, -, done)$