Tracking Redexes in the lambda-calculus

jean-jacques.levy@inria.fr
ECNU, Shanghai

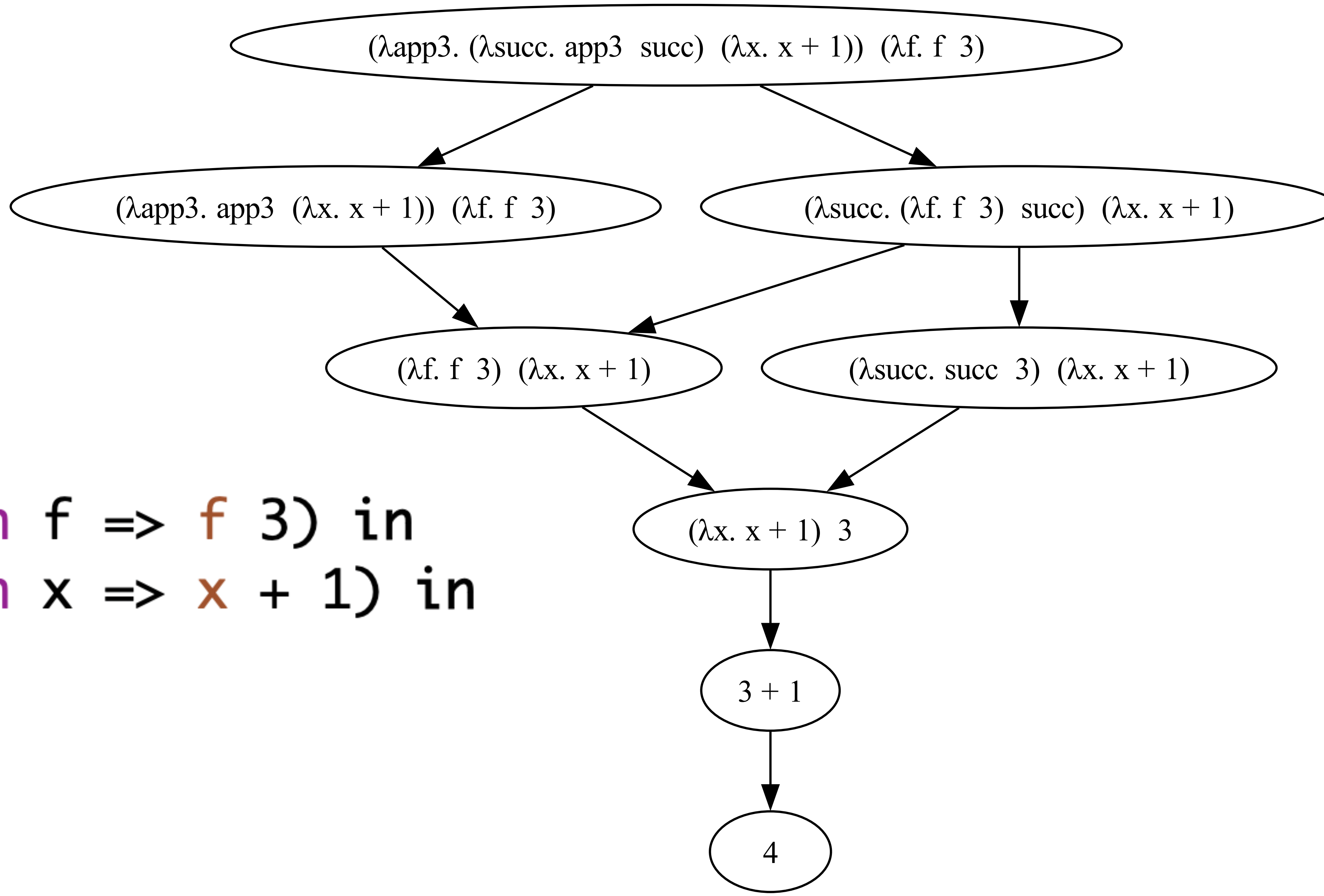
November 15, 2023

http://jeanjacqueslevy.net/talks/23track/track.pdf

The lambda-calculus

- for logicians: important tool for proof theory
- for computer scientists: kernel of functional programming

The lambda-calculus




```
let app3 = (fun f => f 3) in
let succ = (fun x => x + 1) in
app3 succ
```

The lambda-calculus

$$(\lambda x.\, x+1)3 \longrightarrow 3+1 \longrightarrow 4$$

$$(\lambda x.\, 2*x+2)4 \longrightarrow 2*4+2 \longrightarrow 8+2 \longrightarrow 10$$

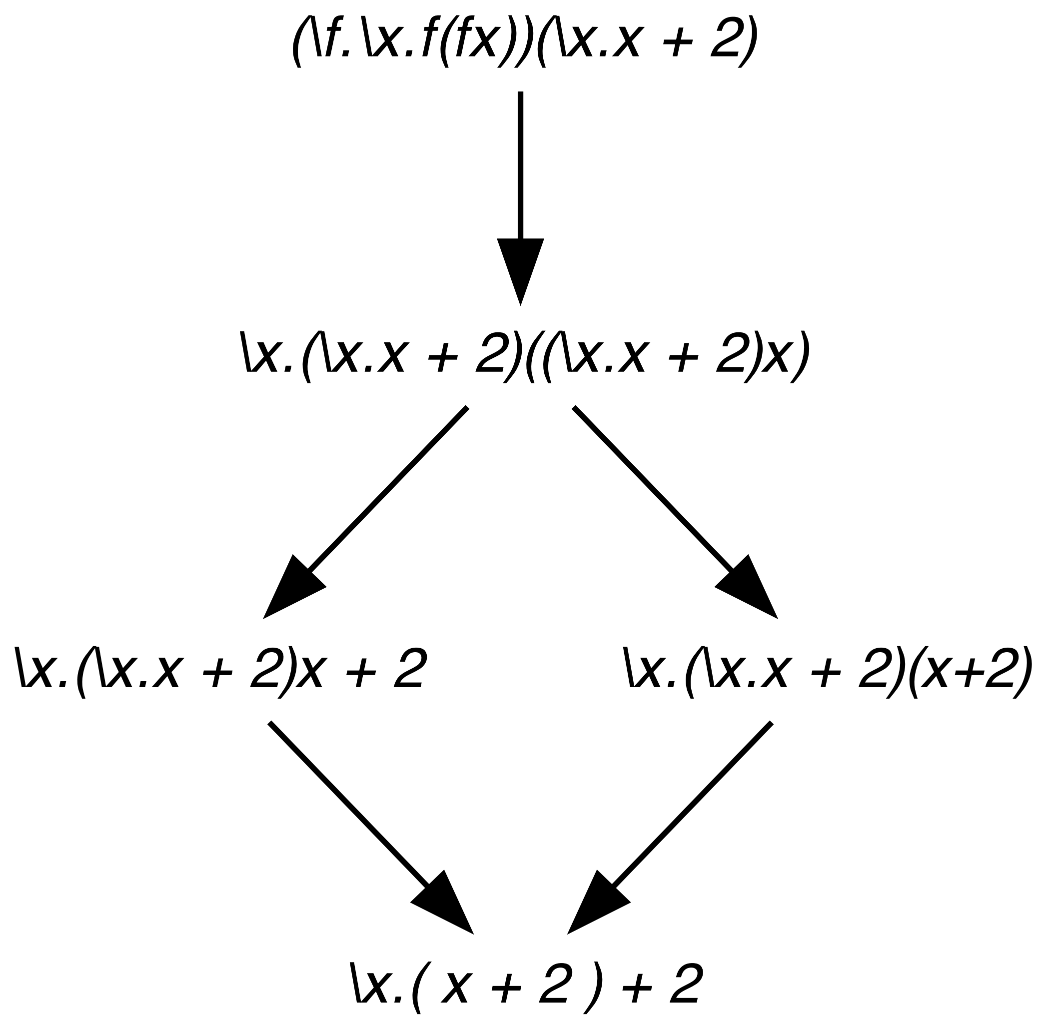
$$(\lambda f.f3)(\lambda x.\, x+2) \longrightarrow (\lambda x.\, x+2)3 \longrightarrow 3+2 \longrightarrow 5$$

$$(\lambda f.\lambda x.f(f\,x))(\lambda x.x+2) \longrightarrow \ldots$$

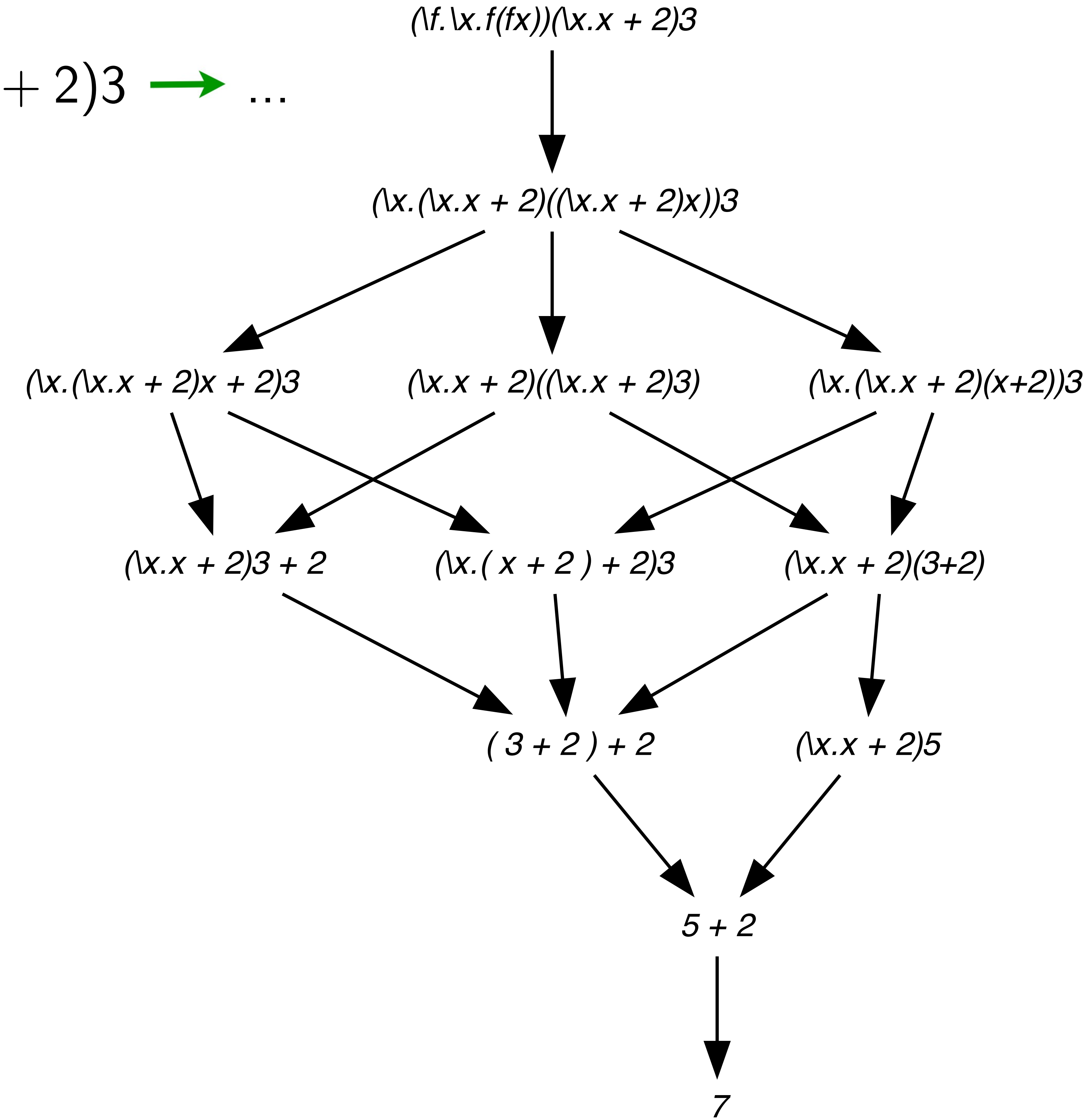
$$\Delta = (\lambda x.x\,x) \quad I = \lambda x.x$$

$$\Delta(\lambda f.f\,I) \rightarrow (\lambda f.f\,I)(\lambda f.f\,I) \rightarrow (\lambda f.f\,I)I \rightarrow I\,I \rightarrow I$$

$$(\lambda f.\lambda x.f(f\,x))(\lambda x.\,x + 2) \longrightarrow \ldots$$



$(\lambda f.\lambda x.f(f\ x))(\lambda x.x + 2)3 \longrightarrow \ldots$



The lambda-calculus

$\texttt{Fact}(3)$

$\texttt{Fact} = Y(\lambda f.\lambda x.\ \texttt{ifz}\ x\ \texttt{then}\ 1\ \texttt{else}\ x \star f(x-1))$

$Y = \lambda f.(\lambda x.f(xx))(\lambda x.f(xx))$

can be written as

$(\lambda\,\texttt{Fact}\,.\,\texttt{Fact}(3))$

$(\ (\lambda Y.Y(\lambda f.\lambda x.\ \texttt{ifz}\ x\ \texttt{then}\ 1\ \texttt{else}\ x \star f(x-1)))$

$(\lambda f.(\lambda x.f(xx))(\lambda x.f(xx)))\ )$

(\Fact.Fact3)((\y.y(\f.\x.ifz x then 1 else x * f(x-1)))(\f.Yf))

↓

(\y.y(\f.\x.ifz x then 1 else x * f(x-1)))(\f.Yf)3

↓

(\f.Yf)(\f.\x.ifz x then 1 else x * f(x-1))3

↓

(\x.(\f.\x.ifz x then 1 else x * f(x-1))(xx))(\x.(\f.\x.ifz x then 1 else x * f(x-1))(xx))3

↓

(\f.\x.ifz x then 1 else x * f(x-1))((\x.(\f.\x.ifz x then 1 else x * f(x-1))(xx))(\x.(\f.\x.ifz x then 1 else x * f(x-1))(xx)))3

↓

(\x.ifz x then 1 else x * (\x.(\f.\x.ifz x then 1 else x * f(x-1))(xx))(\x.(\f.\x.ifz x then 1 else x * f(x-1))(xx))(x-1))3

↓

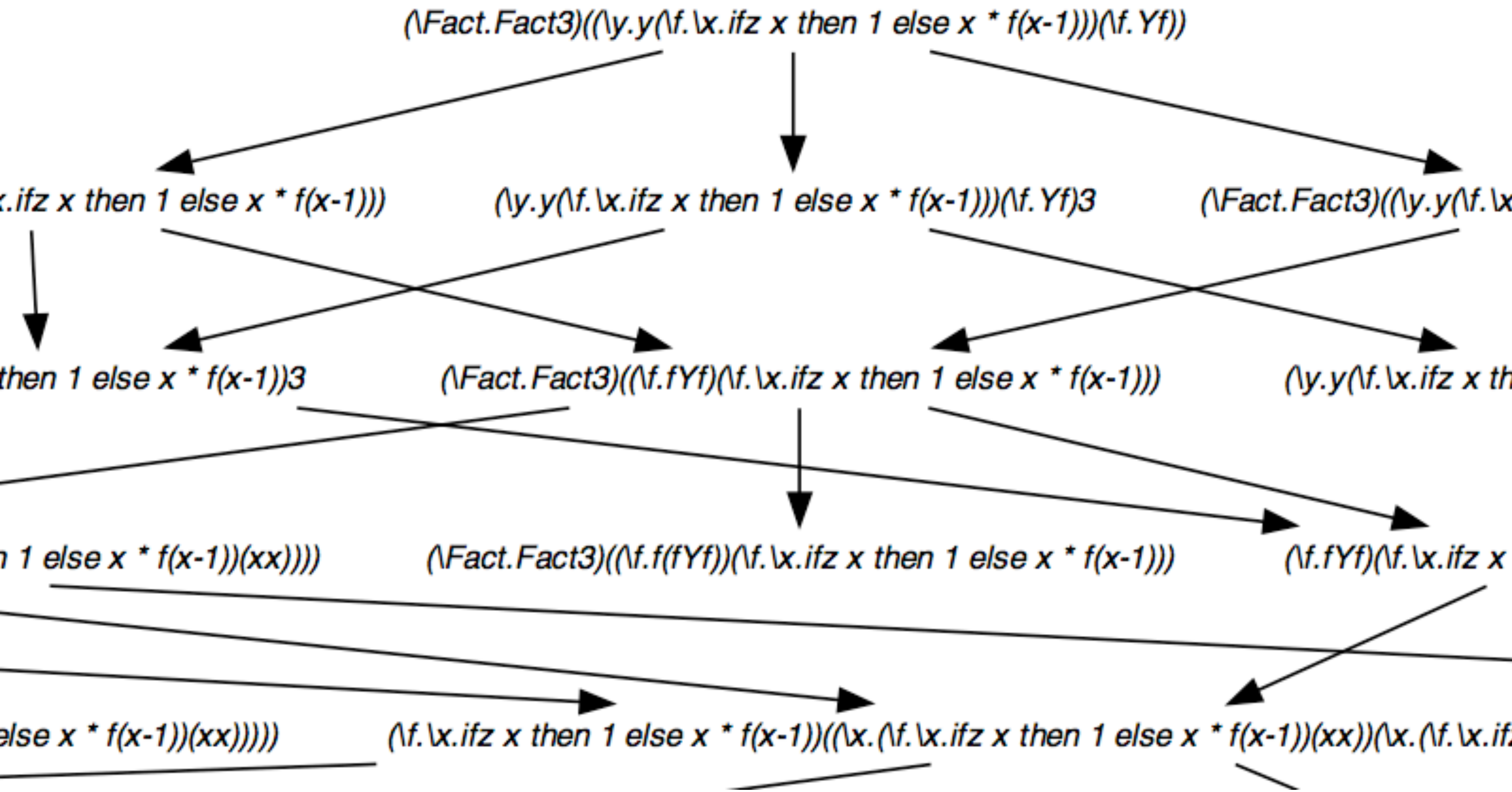
ifz 3 then 1 else 3 * (\x.(\f.\x.ifz x then 1 else x * f(x-1))(xx))(\x.(\f.\x.ifz x then 1 else x * f(x-1))(xx))(3-1)

↓

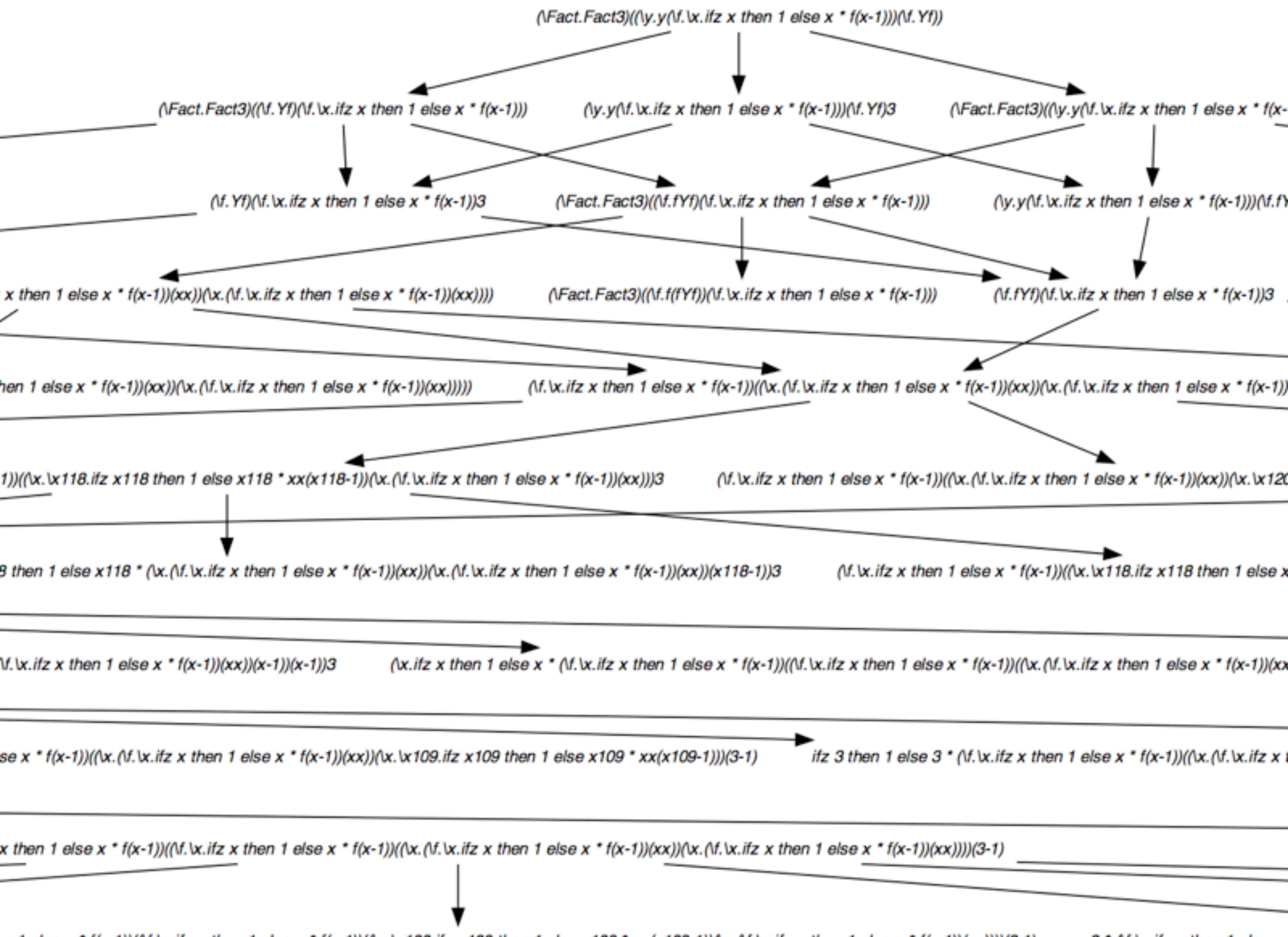
3 * (\x.(\f.\x.ifz x then 1 else x * f(x-1))(xx))(\x.(\f.\x.ifz x then 1 else x * f(x-1))(xx))(3-1)

↓

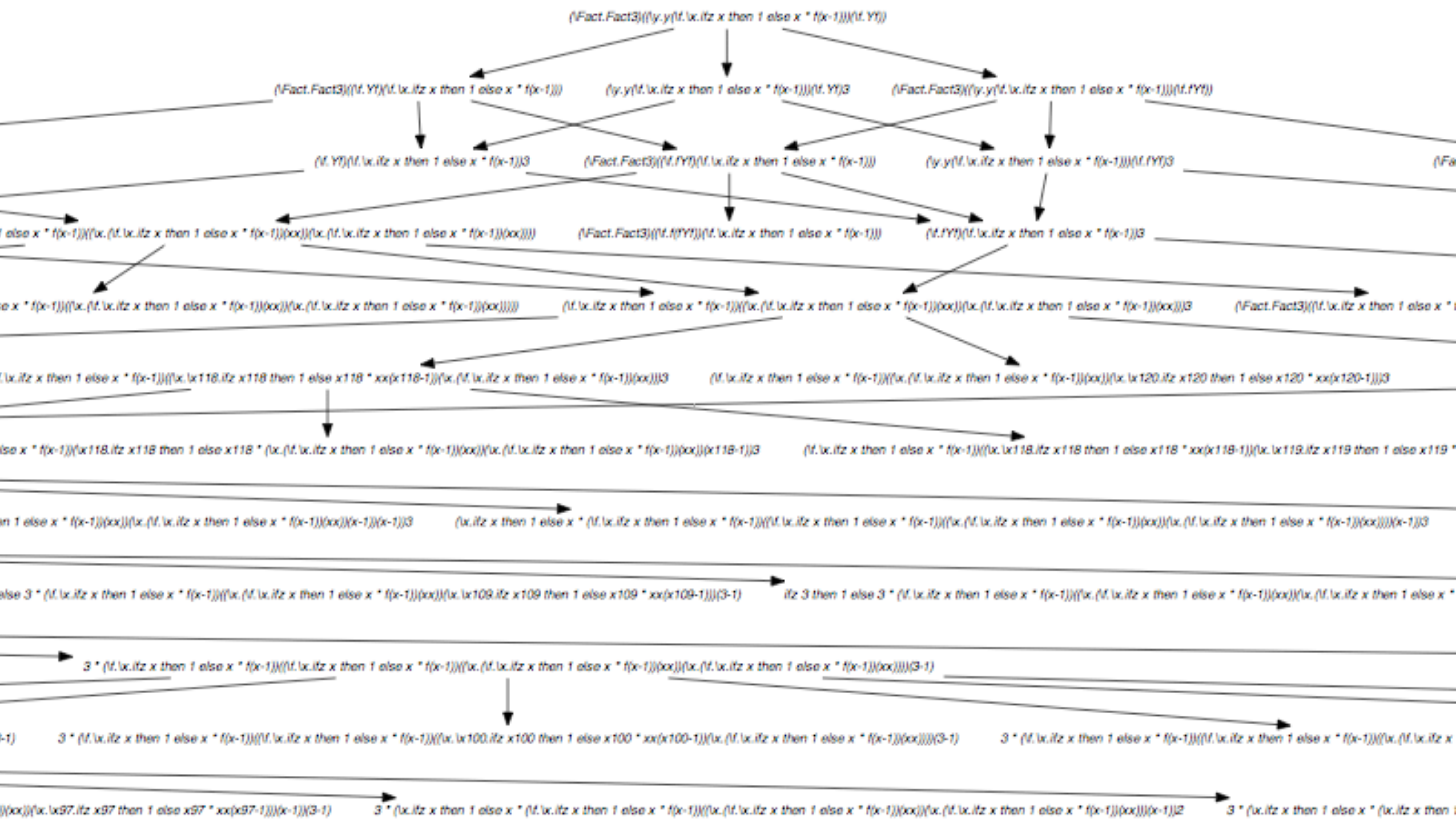
3 * (\f.\x.ifz x then 1 else x * f(x-1))((\x.(\f.\x.ifz x then 1 else x * f(x-1))(xx))(\x.(\f.\x.ifz x then 1 else x * f(x-1))(xx)))(3-1)

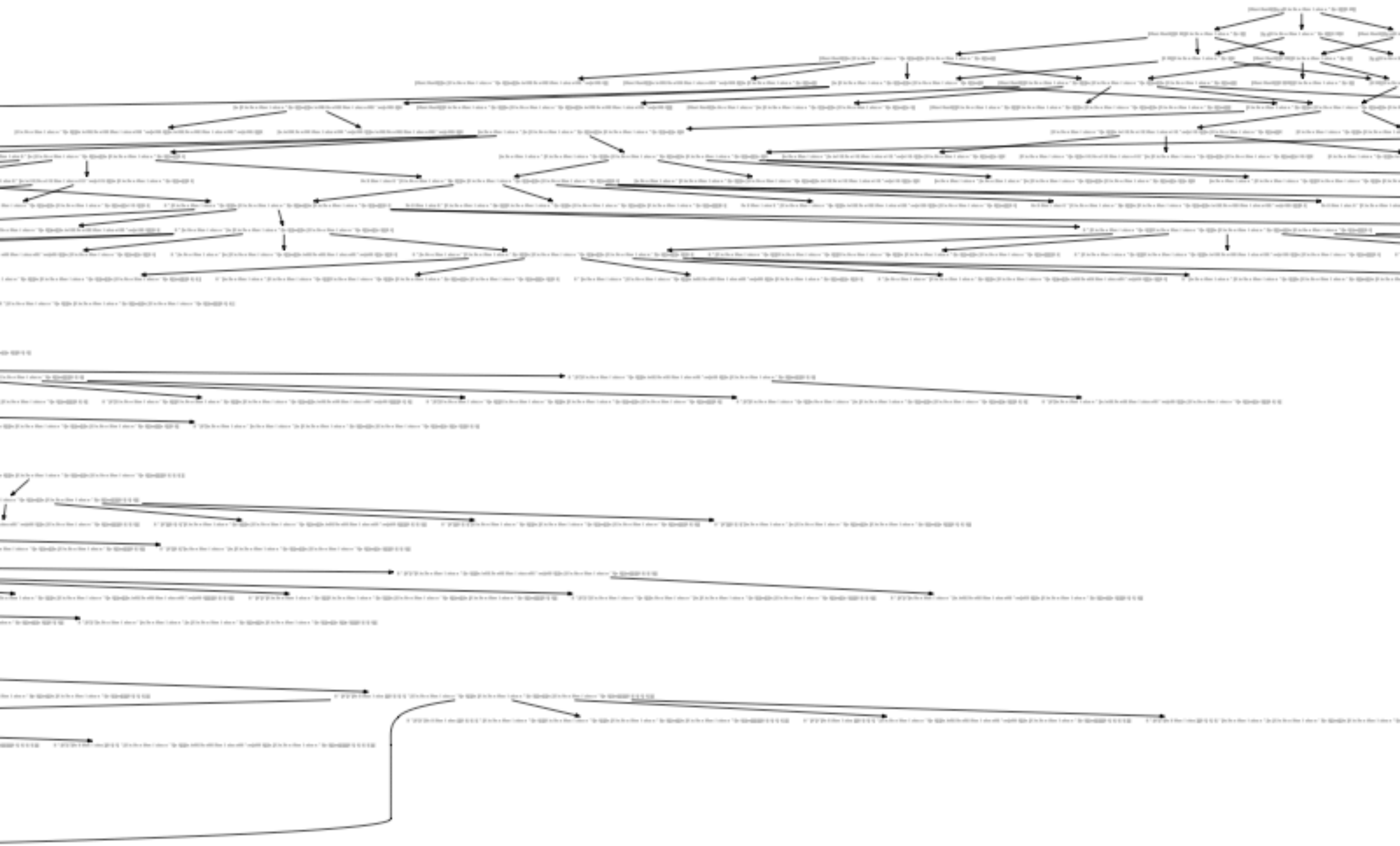


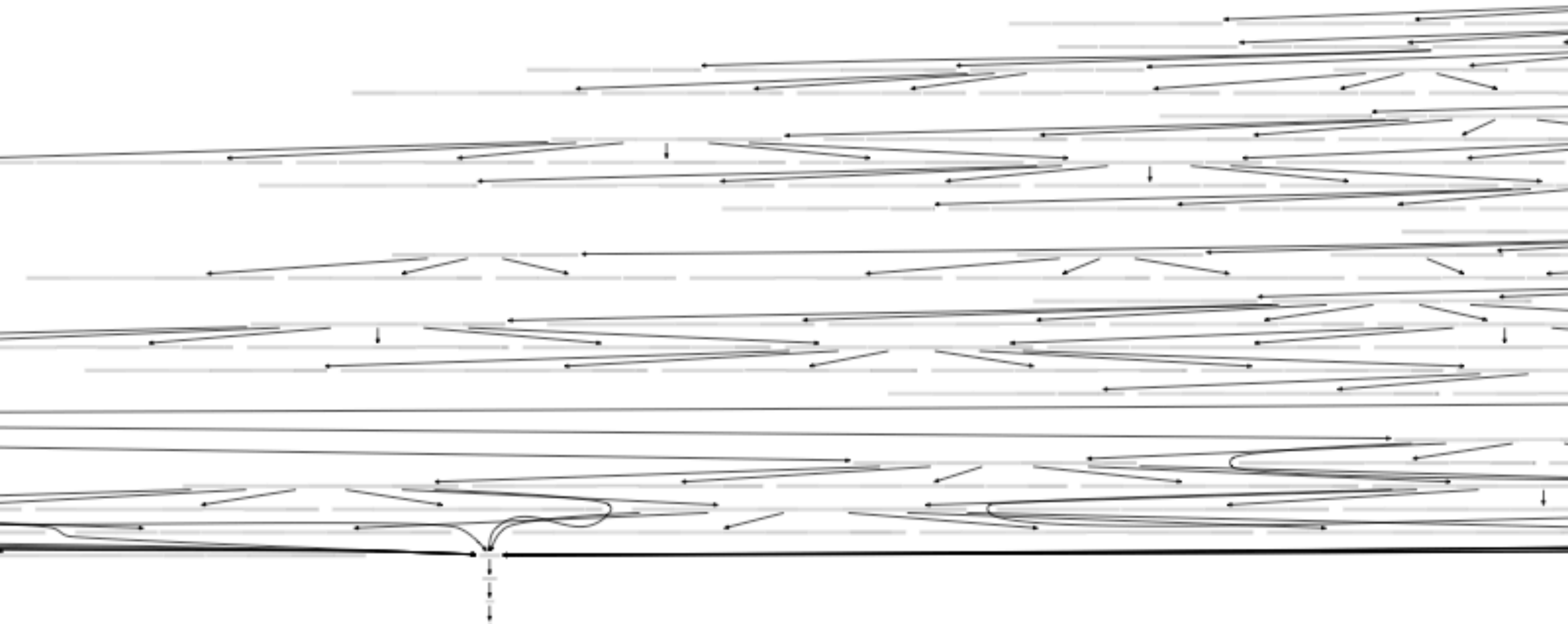
(\Fact.Fact3)((\y.y(\f.\x.ifz x then 1 else x * f(x-1)))(\f.Yf))
.ifz x then 1 else x * f(x-1)))
(\y.y(\f.\x.ifz x then 1 else x * f(x-1)))(\f.Yf)3
(\Fact.Fact3)((\y.y(\f.\x
then 1 else x * f(x-1))3
(\Fact.Fact3)((\f.fYf)(\f.\x.ifz x then 1 else x * f(x-1)))
(\y.y(\f.\x.ifz x th
n 1 else x * f(x-1))(xx))))
(\Fact.Fact3)((\f.f(fYf))(\f.\x.ifz x then 1 else x * f(x-1)))
(\f.fYf)(\f.\x.ifz x
else x * f(x-1))(xx)))))
(\f.\x.ifz x then 1 else x * f(x-1))((\x.(\f.\x.ifz x then 1 else x * f(x-1))(xx))(\x.(\f.\x.if

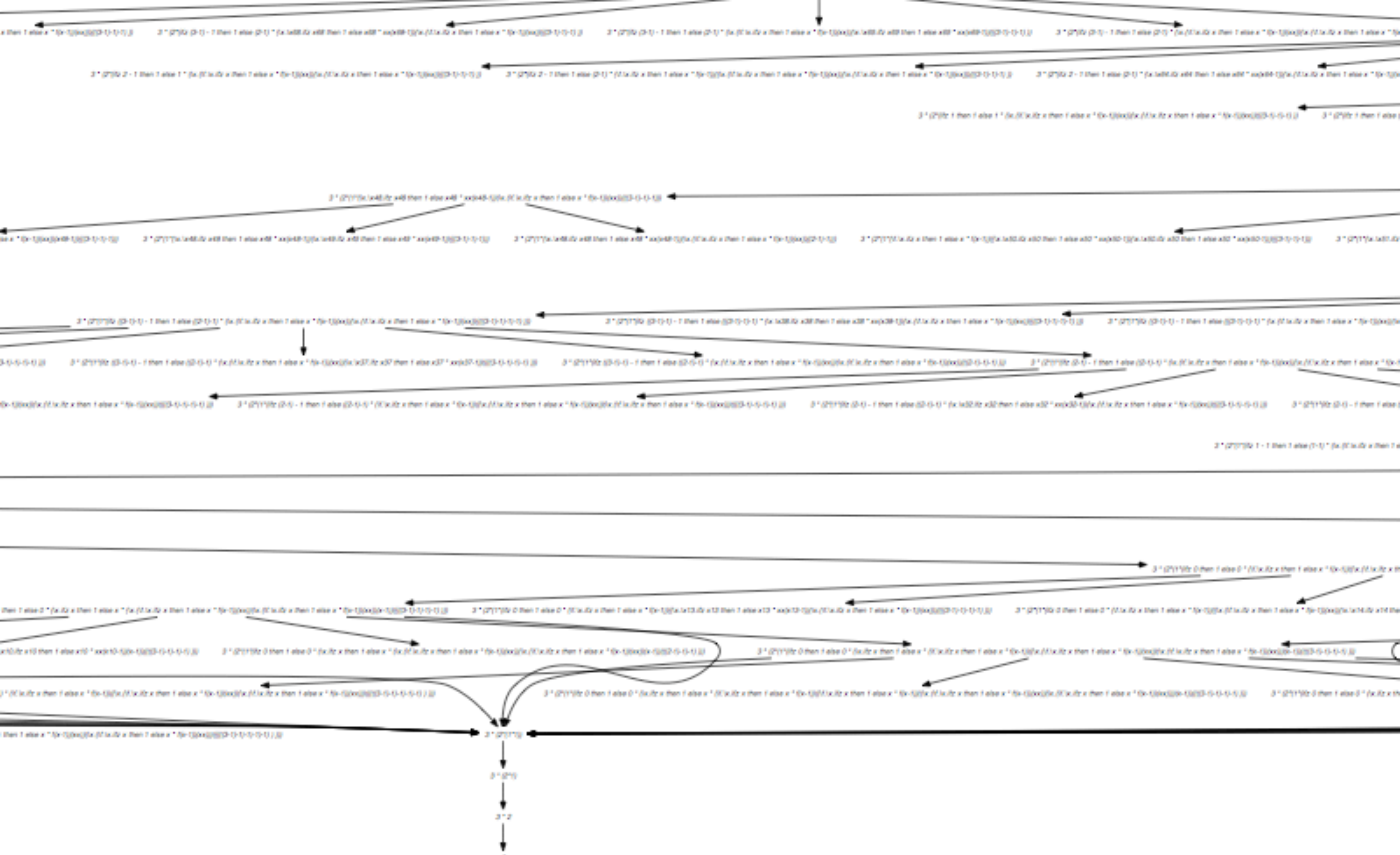


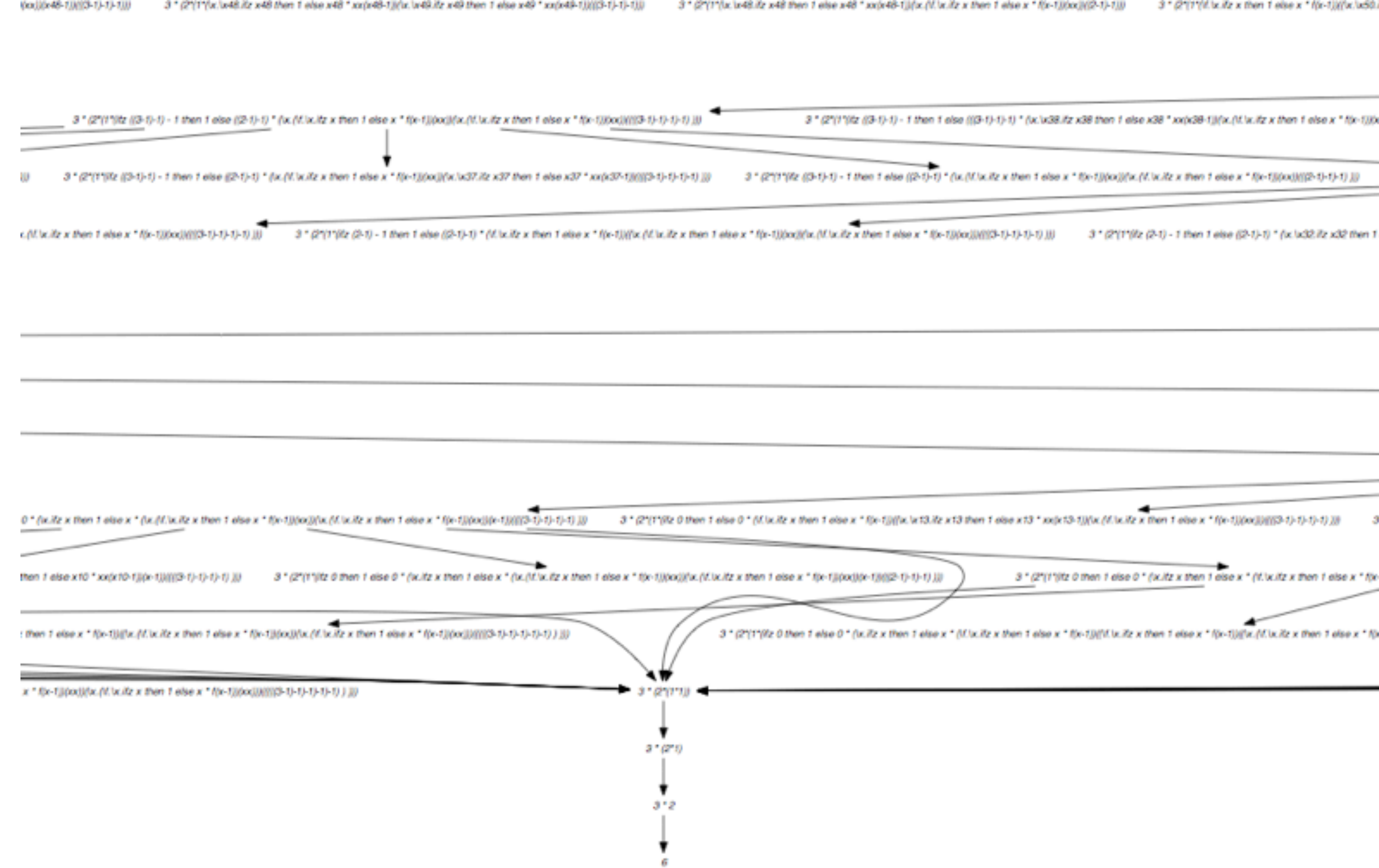
(\Fact.Fact3)((\y.y(\f.\x.ifz x then 1 else x * f(x-1)))(\f.Yf))
(\Fact.Fact3)((\f.Yf)(\f.\x.ifz x then 1 else x * f(x-1)))
(\y.y(\f.\x.ifz x then 1 else x * f(x-1)))(\f.Yf)3
(\Fact.Fact3)((\y.y(\f.\x.ifz x then 1 else x * f(x-
(\f.Yf)(\f.\x.ifz x then 1 else x * f(x-1))3
(\Fact.Fact3)((\f.fYf)(\f.\x.ifz x then 1 else x * f(x-1)))
(\y.y(\f.\x.ifz x then 1 else x * f(x-1)))(\f.fY
x then 1 else x * f(x-1))(xx))(\x.(\f.\x.ifz x then 1 else x * f(x-1))(xx))))
(\Fact.Fact3)((\f.f(fYf))(\f.\x.ifz x then 1 else x * f(x-1)))
(\f.fYf)(\f.\x.ifz x then 1 else x * f(x-1))3
hen 1 else x * f(x-1))(xx))(\x.(\f.\x.ifz x then 1 else x * f(x-1))(xx)))))
(\f.\x.ifz x then 1 else x * f(x-1))((\x.(\f.\x.ifz x then 1 else x * f(x-1))(xx))(\x.(\f.\x.ifz x then 1 else x * f(x-1))
1))((\x.\x118.ifz x118 then 1 else x118 * xx(x118-1))(\x.(\f.\x.ifz x then 1 else x * f(x-1))(xx)))3
(\f.\x.ifz x then 1 else x * f(x-1))((\x.(\f.\x.ifz x then 1 else x * f(x-1))(xx))(\x.\x120
8 then 1 else x118 * (\x.(\f.\x.ifz x then 1 else x * f(x-1))(xx))(\x.(\f.\x.ifz x then 1 else x * f(x-1))(xx))(x118-1))3
(\f.\x.ifz x then 1 else x * f(x-1))((\x.\x118.ifz x118 then 1 else x
\f.\x.ifz x then 1 else x * f(x-1))(xx))(x-1))(x-1))3
(\x.ifz x then 1 else x * (\f.\x.ifz x then 1 else x * f(x-1))((\f.\x.ifz x then 1 else x * f(x-1))((\x.(\f.\x.ifz x then 1 else x * f(x-1))(xx
se x * f(x-1))((\x.(\f.\x.ifz x then 1 else x * f(x-1))(xx))(\x.\x109.ifz x109 then 1 else x109 * xx(x109-1)))(3-1)
ifz 3 then 1 else 3 * (\f.\x.ifz x then 1 else x * f(x-1))((\x.(\f.\x.ifz x t
x then 1 else x * f(x-1))((\f.\x.ifz x then 1 else x * f(x-1))((\x.(\f.\x.ifz x then 1 else x * f(x-1))(xx))(\x.(\f.\x.ifz x then 1 else x * f(x-1))(xx))))(3-1)

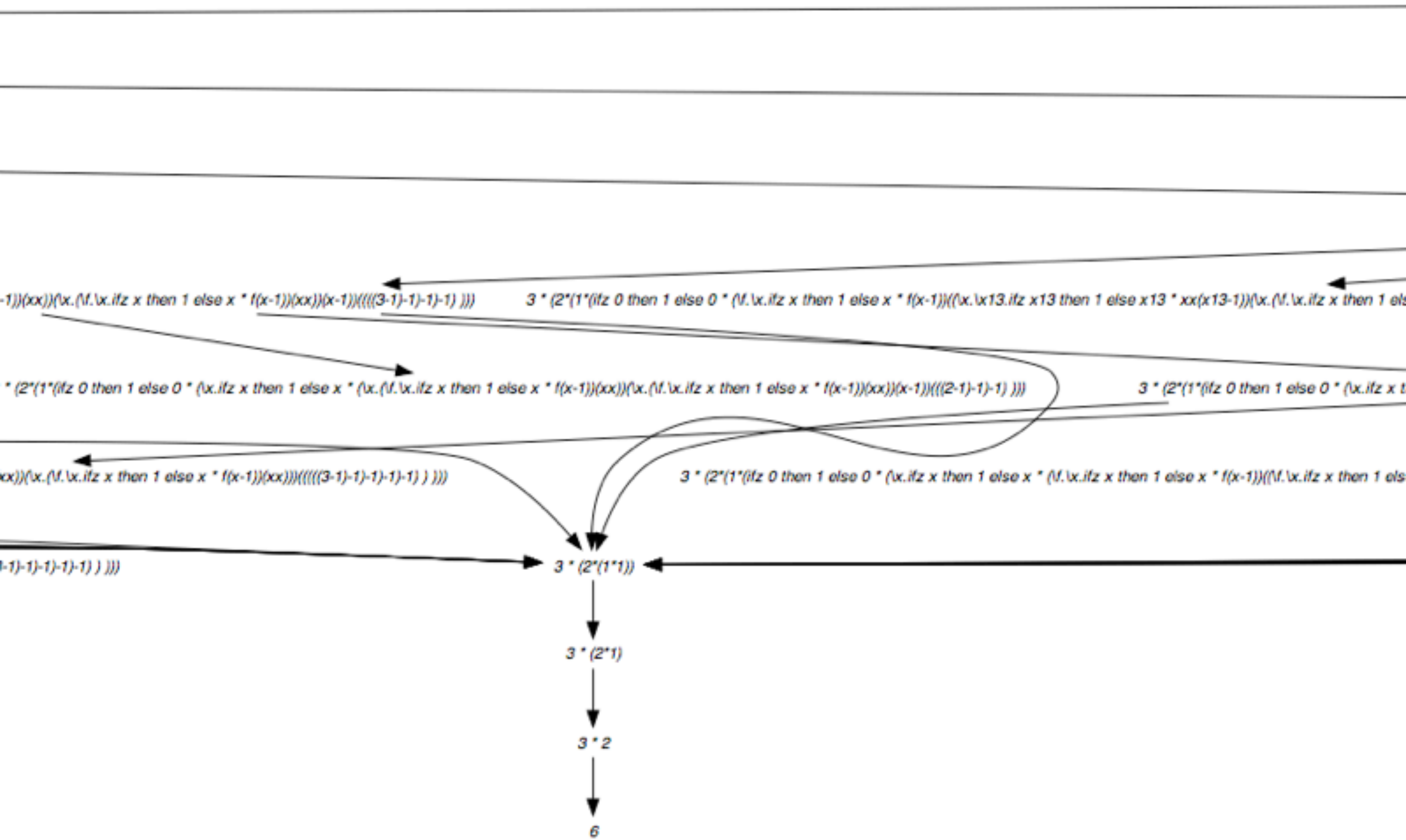

(\Fact.Fact3)((\y.y(\f.\x.ifz x then 1 else x * f(x-1)))(\f.Yf))
(\Fact.Fact3)((\f.Yf)(\f.\x.ifz x then 1 else x * f(x-1)))
(\y.y(\f.\x.ifz x then 1 else x * f(x-1)))(\f.Yf)3
(\Fact.Fact3)((\y.y(\f.\x.ifz x then 1 else x * f(x-1)))(\f.fYf))
(\f.Yf)(\f.\x.ifz x then 1 else x * f(x-1))3
(\Fact.Fact3)((\f.fYf)(\f.\x.ifz x then 1 else x * f(x-1)))
(\y.y(\f.\x.ifz x then 1 else x * f(x-1)))(\f.fYf)3
(\Fact.Fact3)((\f.f(fYf))(\f.\x.ifz x then 1 else x * f(x-1)))
(\f.fYf)(\f.\x.ifz x then 1 else x * f(x-1))3

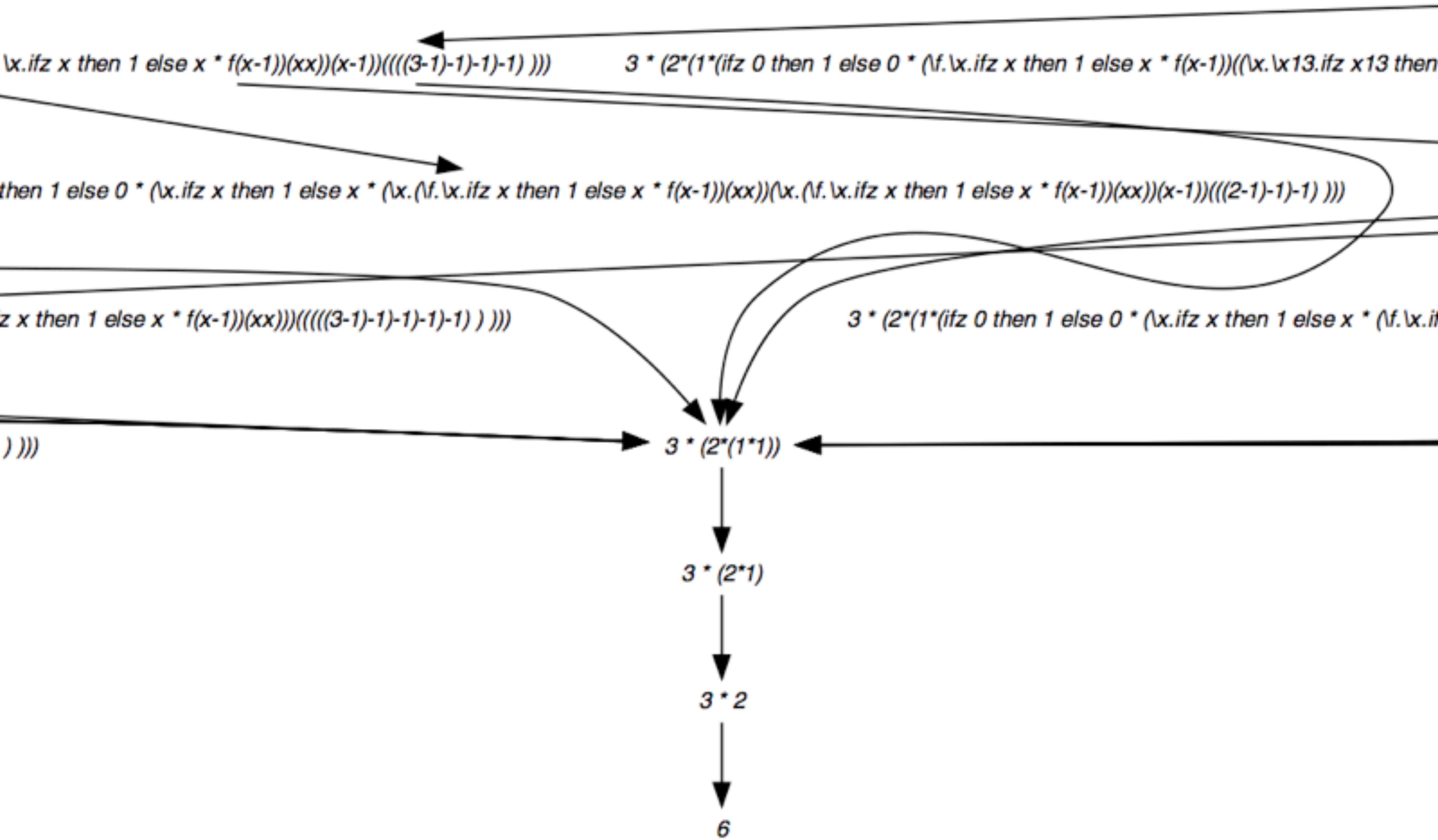





3 * 2
6


3 * (2*(1*1))
3 * (2*1)
3 * 2
6


-1))(xx))(\x.(\f.\x.ifz x then 1 else x * f(x-1))(xx))(x-1))((((3-1)-1)-1)-1))))
3 * (2*(1*(ifz 0 then 1 else 0 * (\f.\x.ifz x then 1 else x * f(x-1))((\x.\x13.ifz x13 then 1 else x13 * xx(x13-1))(\x.(\f.\x.ifz x then 1 els
* (2*(1*(ifz 0 then 1 else 0 * (\x.ifz x then 1 else x * (\x.(\f.\x.ifz x then 1 else x * f(x-1))(xx))(\x.(\f.\x.ifz x then 1 else x * f(x-1))(xx))(x-1))(((2-1)-1)-1))))
3 * (2*(1*(ifz 0 then 1 else 0 * (\x.ifz x t
xx))(\x.(\f.\x.ifz x then 1 else x * f(x-1))(xx)))(((((3-1)-1)-1)-1)-1)))))
3 * (2*(1*(ifz 0 then 1 else 0 * (\x.ifz x then 1 else x * (\f.\x.ifz x then 1 else x * f(x-1))((\f.\x.ifz x then 1 els
-1)-1)-1)-1)-1)))))
3 * (2*(1*1))
3 * (2*1)
3 * 2
6


\x.ifz x then 1 else x * f(x-1))(xx))(x-1))((((3-1)-1)-1)-1))))
3 * (2*(1*(ifz 0 then 1 else 0 * (\f.\x.ifz x then 1 else x * f(x-1))((\x.\x13.ifz x13 then
then 1 else 0 * (\x.ifz x then 1 else x * (\x.(\f.\x.ifz x then 1 else x * f(x-1))(xx))(\x.(\f.\x.ifz x then 1 else x * f(x-1))(xx))(x-1))(((2-1)-1)-1))))
z x then 1 else x * f(x-1))(xx)))(((((3-1)-1)-1)-1)-1)))))
3 * (2*(1*(ifz 0 then 1 else 0 * (\x.ifz x then 1 else x * (\f.\x.i
))))
3 * (2*(1*1))
3 * (2*1)
3 * 2
6

Initial motivation

- Bohm trees interpretation for the (untyped) **λ**-calculus
  - head normal forms are $M \twoheadrightarrow \lambda x_1 x_2 \cdots x_m.\, x\, M_1\, M_2 \cdots M_n$
  - Bohm trees are (possibly infinite) extensions of head normal forms

$$\mathrm{BT}(M) = \lambda x_1 x_2 \cdots x_m.\, x\ \mathrm{BT}(M_1)\ \mathrm{BT}(M_2) \cdots \mathrm{BT}(M_n)$$

  - Bohm trees are a consistent interpretation of the **λ**-calculus

$$\mathrm{BT}(M) = \mathrm{BT}(N) \implies \mathrm{BT}(C[M]) = \mathrm{BT}(C[N])$$

- continuity of Bohm trees

$$\forall b \prec \mathrm{BT}(C[M]),\ \exists a \prec \mathrm{BT}(M),\ b \prec BT(C[a])$$

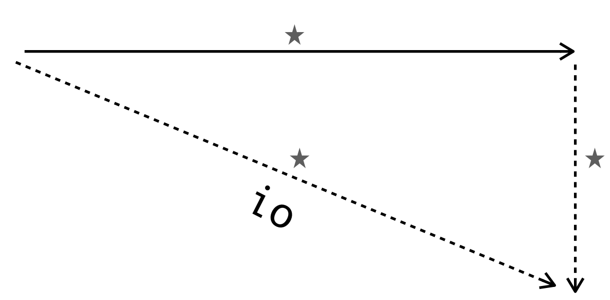
- finite computations of $C[M]$ only need finite computations of M

Initial motivation

- completeness of inside-out reductions $\Longrightarrow$ continuity of BT [Welch 1974]

  - an inside-out reduction does not contract residual of a redex internal to a redex previously contracted.

- completeness of io reductions:



- obvious if strong normalisation

- but when infinite reductions ?? !!

HWλ-calculus

with

exponents

Hyland-Wadsworth λ-calculus

- D-infinity model of the **λ**-calculus [Scott 1969]

$$D_\infty \;=\; \lim_{n\to\infty}\{D_n \mid D_{n+1} = D_n \to D_n\}$$

- Indexed **λ**-calculus [Hyland-Wadsworth 1971; revised JJL 1974]

$$M, N, \ldots ::= x \mid MN \mid \lambda x.M \mid M^n \qquad (n \geq 0)$$

$$(\lambda x.M)^{n+1} N \to M\{x := N^n\}^n$$

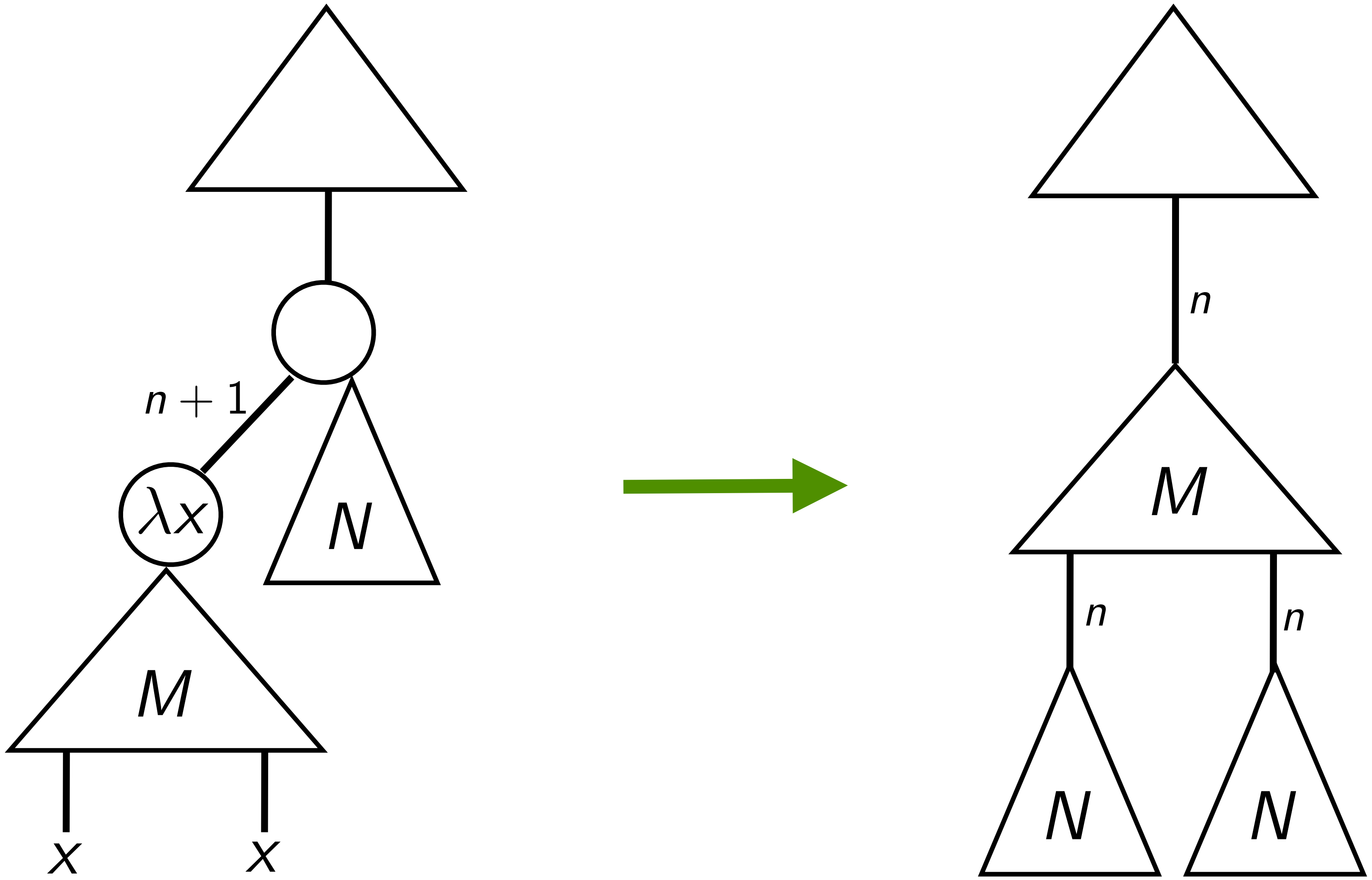
$$M^n\{x := N\} = M\{x := N\}^n$$

$$(M^m)^n = M^p \quad \text{where} \quad p = \min\{m, n\}$$

- An example: $\Delta_n = (\lambda x.(x^{10} x^4)^{20})^n$

$$(\Delta_3\, \Delta_4)^{15} \to (\Delta_2\, \Delta_2)^2 \to (\Delta_1\, \Delta_1)^1 \to (\Delta_0\, \Delta_0)^0$$

Hyland-Wadsworth λ-calculus



Hyland-Wadsworth $\boldsymbol{\lambda}$-calculus

- An example: $\Delta_n = (\lambda x.(x^{10}x^4)^{20})^n$

$$
\begin{aligned}
(\Delta_3\,\Delta_4)^{15} &\to ((x^{10}x^4)^{20}\{x := (\Delta_4)^2\}^2)^{15}\\
&= (x^{10}x^4)^{20}\{x := (\Delta_4)^2\}^2\\
&= (x^{10}x^4)^{20}\{x := \Delta_2\}^2\\
&= ((x^{10}x^4)^{20})^2\{x := \Delta_2\}\\
&= (x^{10}x^4)^2\{x := \Delta_2\}\\
&= ((\Delta_2)^{10}(\Delta_2)^4)^2\\
&= (\Delta_2\Delta_2)^2
\end{aligned}
$$

$$(\Delta_3\,\Delta_4)^{15} \to (\Delta_2\,\Delta_2)^2 \to (\Delta_1\,\Delta_1)^1 \to (\Delta_0\,\Delta_0)^0$$

- In the standard $\boldsymbol{\lambda}$-calculus, we have

$$(\lambda x.x\,x)(\lambda x.x\,x) \to (\lambda x.x\,x)(\lambda x.x\,x) \to \cdots$$

Hyland-Wadsworth λ-calculus

- Let $\Delta = \lambda x.x\,x$

$$(\Delta^3\,\Delta^3)^3 \rightarrow (\Delta^2\,\Delta^2)^2 \rightarrow (\Delta^1\,\Delta^1)^1 \rightarrow (\Delta^0\,\Delta^0)^0$$

- Let the degree of a redex be the exponent of its function part

$$\text{degree}(\,(\lambda x.M)^n N\,) = n$$

- The degree of a redex gives its "computing power"

- Residuals of a redex keep their degree

- Created new redexes have lower degree

Hyland-Wadsworth λ-calculus

- HWλ-calculus is confluent and strongly normalizable
- no infinite reductions
- unique normal form
- the standard λ-calculus can be seen as an infinite limit of HWλ-calculus

Hyland-Wadsworth λ-calculus

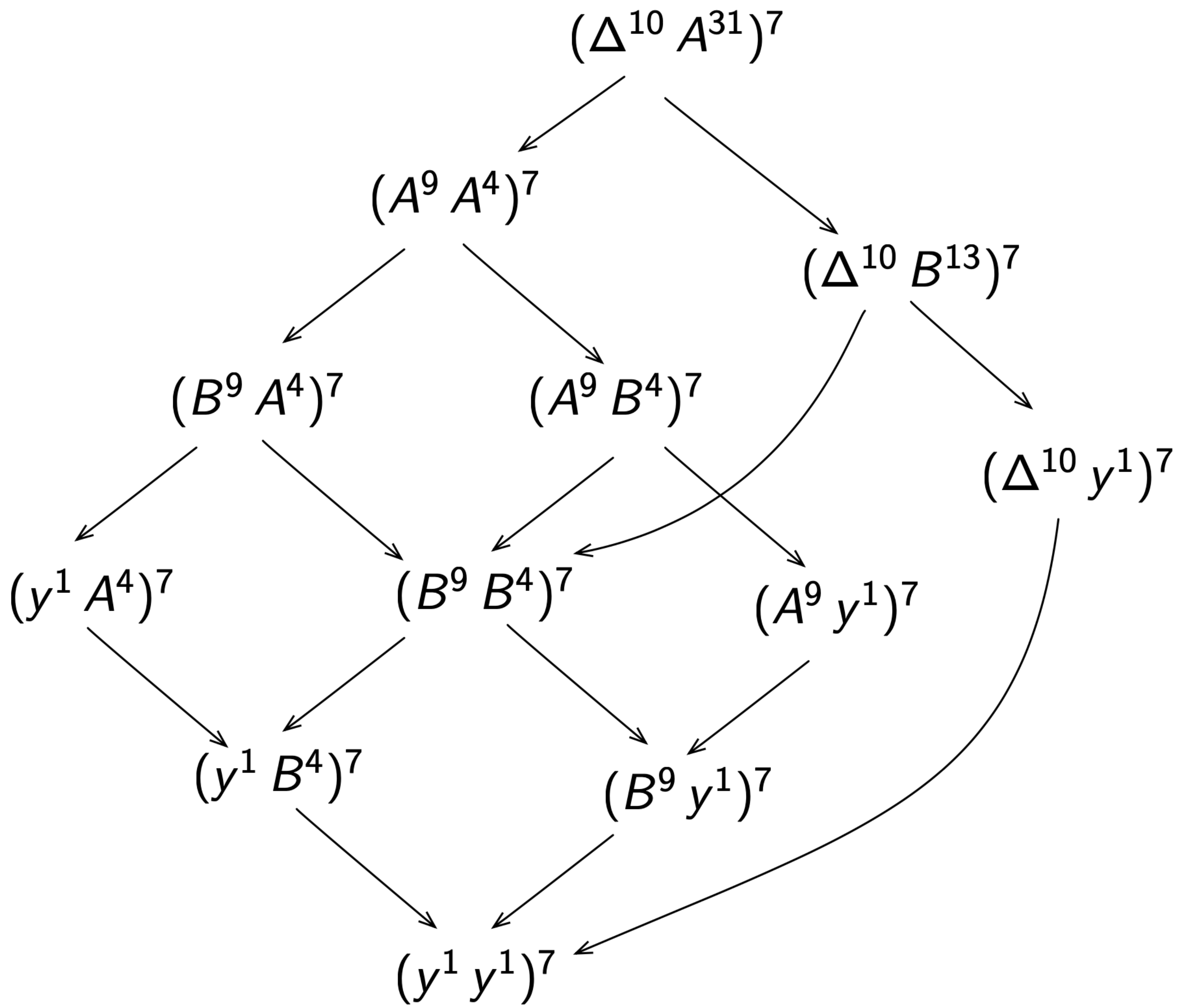
$\Delta = \lambda x.(x^{10}x^4)^{20}$

$F = \lambda f.(f^{27}\, y^5)^{13}$

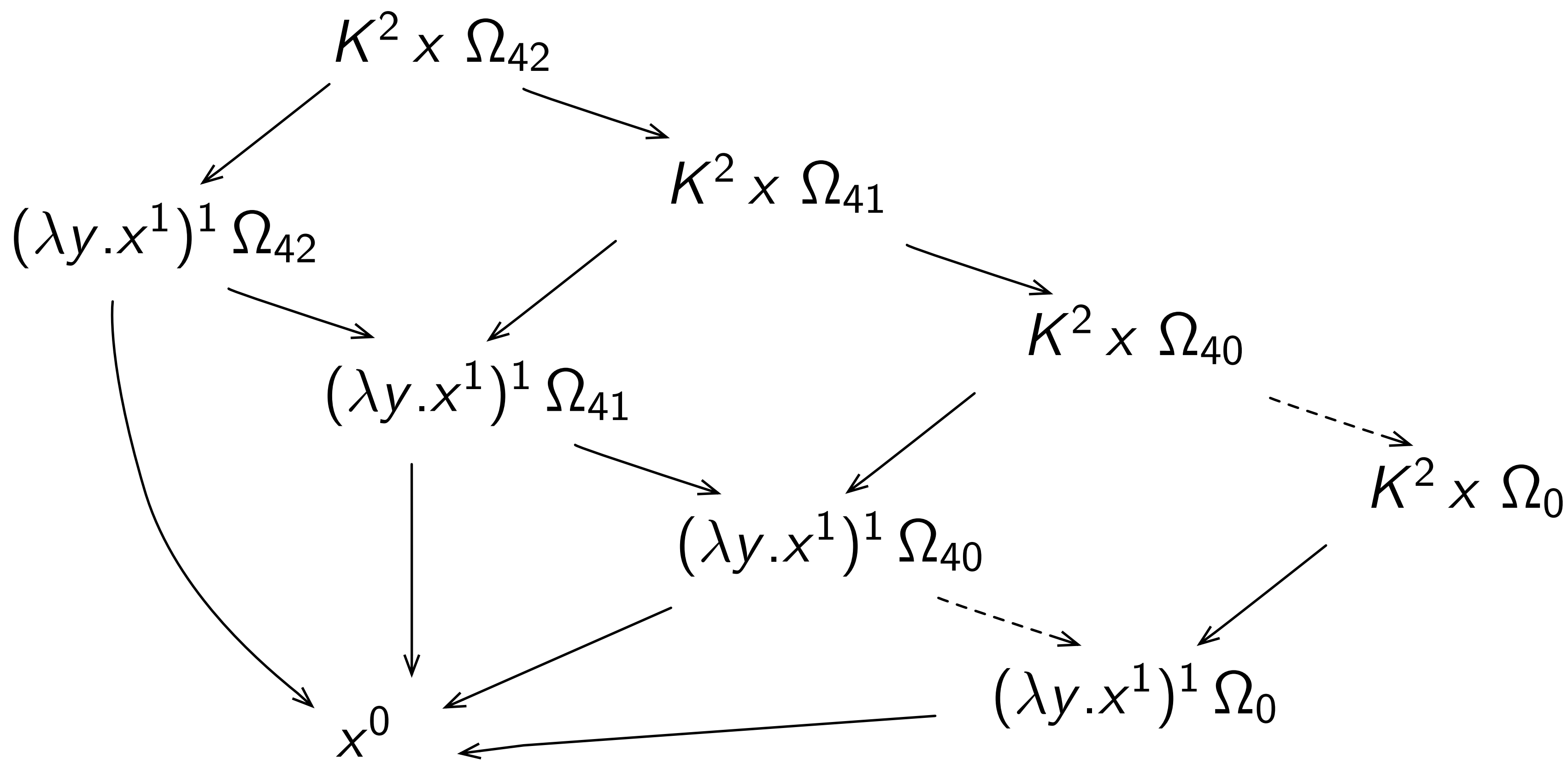
$I = \lambda x.x^8$

$A = F^{42}\, I^2$

$B = I^2\, y^5$



Hyland-Wadsworth λ-calculus



$K = \lambda x.\lambda y.x$

$\Omega_n = (\Delta^n \Delta^n)^n$

Hyland-Wadsworth λ-calculus

$$\underbrace{(\lambda x. \cdots (x^p\ N) \cdots)^{n+1}\ (\lambda y.M)^m}_{n+1} \to \cdots \underbrace{((\lambda y.M)^q\ N')}_{q = \min\{p, n, m\}} \cdots$$

$n+1$ creates $q = \min\{p, n, m\}$

$$\underbrace{((\lambda x.(\lambda y.M)^m)^{n+1} N)^p}_{n+1}\ P \to \underbrace{(\lambda y.M')^q\ P}_{q = \min\{p, n, m\}}$$

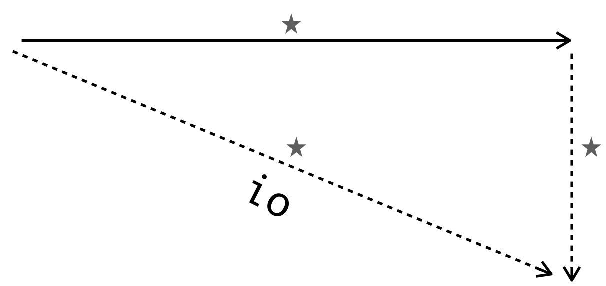
$n+1$ creates $q = \min\{p, n, m\}$

$$\underbrace{((\lambda x.x^p)^{n+1}(\lambda y.M)^m)^q}_{n+1}\ N \to \underbrace{(\lambda y.M)^r}_{r = \min\{p, n, m, q\}}\ N$$

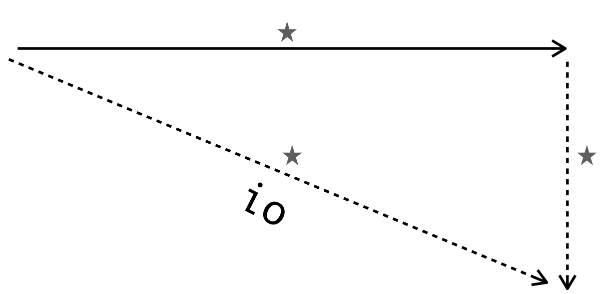
$n+1$ creates $r = \min\{p, n, m, q\}$

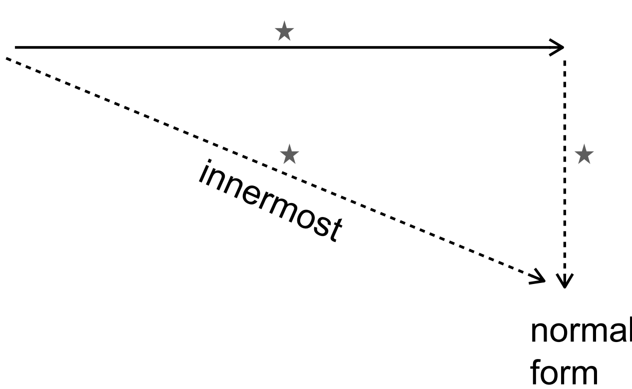
Initial motivation

- completeness of `io` reductions:



- goes to HWλ-calculus





λ-calculus

HWλ-calculus

the labeled λ-calculus

From HWλ-calculus to a labeled λ-calculus

- An abstract set of labels $\{\alpha, \beta, \gamma, ...\}$

- A labeled λ-calculus

$$M, N, ... ::= x \mid MN \mid \lambda x.M \mid M^\alpha$$

$$(\lambda x.M)^\alpha N \rightarrow M\{x := N^{g(\alpha)}\}^{h(\alpha)}$$ where *f, g, h* are 3 unknown functions

$$M^\alpha\{x := N\} = M\{x := N\}^\alpha$$

$$(M^\alpha)^\beta = M^\gamma \quad \text{where} \quad \gamma = f(\alpha, \beta)$$

- consistency of f in $((M^\alpha)^\beta)^\gamma$

$$f(f(\alpha, \beta), \gamma) = f(\alpha, f(\beta, \gamma))$$

The labeled λ-calculus

- An abstract set of labels on alphabet $\mathcal{A} = \{a, b, c, \cdots\}$

$$\alpha, \beta \quad ::= \quad a \mid \alpha\beta \mid \lceil\alpha\rceil \mid \lfloor\alpha\rfloor$$

- A labeled **λ**-calculus

$$M, N, ... ::= x \mid MN \mid \lambda x.M \mid M^{\alpha}$$

$$(\lambda x.M)^{\alpha} N \rightarrow M\{x := N^{\lfloor\alpha\rfloor}\}^{\lceil\alpha\rceil}$$

$$M^{\alpha}\{x := N\} = M\{x := N\}^{\alpha}$$

$$(M^{\alpha})^{\beta} = M^{\alpha\beta}$$

The labeled λ-calculus

- An alphabet of atomic labels $\mathcal{A} = \{a, b, c, \cdots\}$

- A labeled **λ**-calculus [Asperti-Laneve]

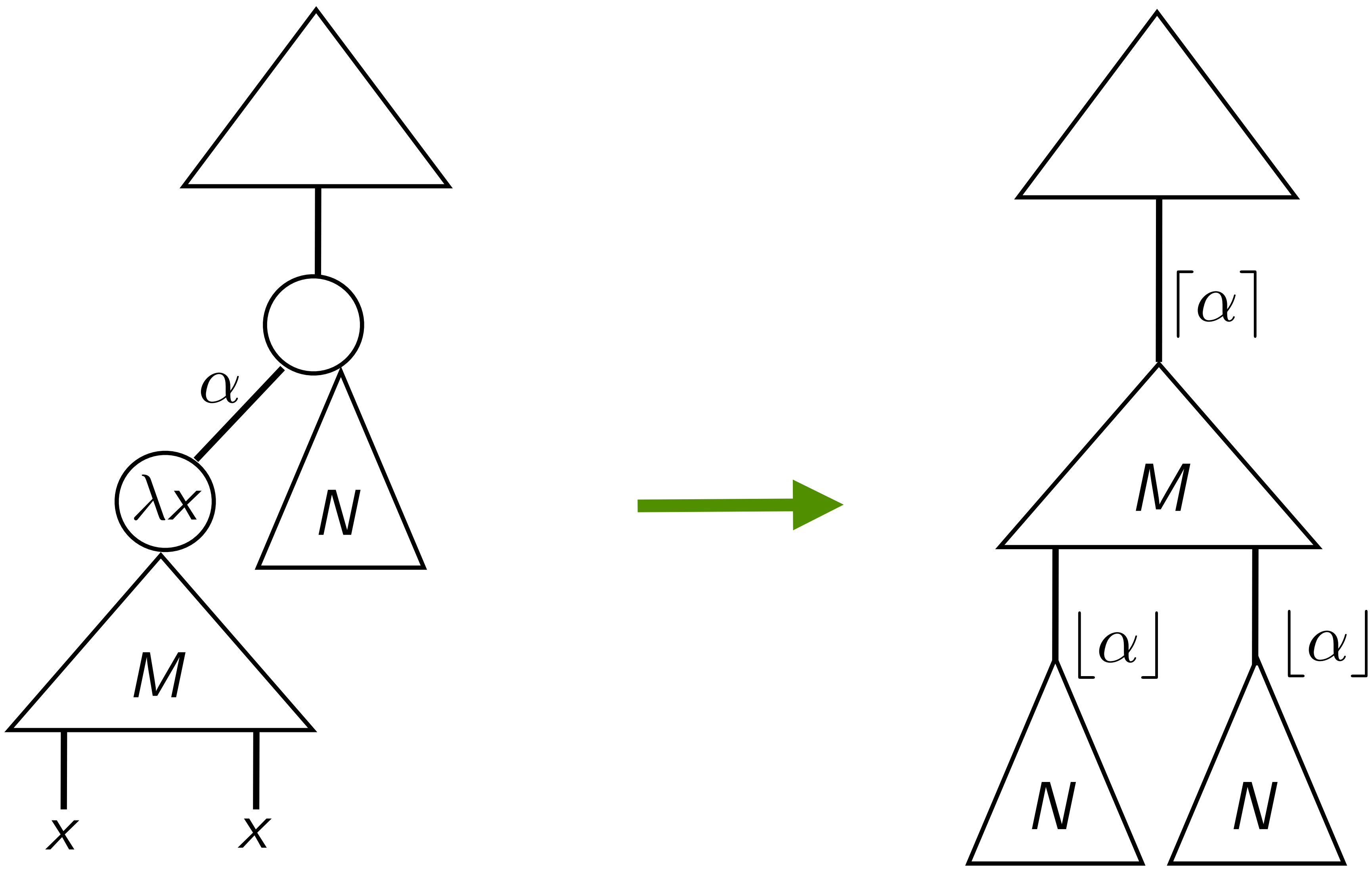
$$M, N, ... ::= x \mid MN \mid \lambda x.M \mid a : M$$

$$(a_1 : a_2 : \cdots a_n : \lambda x.M)N \rightarrow a_1 : a_2 : \cdots a_n : M\{x := a_n : a_{n-1} : \cdots a_1 : N\}$$

$$(a : M)\{x := N\} = a : M\{x := N\}$$

- Correspondence with paths in initial term

The labeled λ-calculus



The labeled λ-calculus

- Let $\Delta = \lambda x.xx,\ \gamma_1 = \lfloor a \rfloor,\ \gamma_2 = \gamma_1 \lfloor \gamma_1 \rfloor$

$$\Delta^a\,\Delta \rightarrow (\Delta^{\gamma_1}\Delta^{\gamma_1})^{\lceil a \rceil} \rightarrow (\Delta^{\gamma_2}\Delta^{\gamma_2})^{\lceil \gamma_1 \rceil \lceil a \rceil} \rightarrow \cdots$$

- Let the name of a redex be the label of its function part

$$\mathrm{name}(\, (\lambda x.M)^\alpha N \,) = \alpha$$

- The name of a redex gives its "origin"

- Residuals of a redex keep their names

- Created new redexes strictly contain the names of their creators

The labeled λ-calculus

- labels over alphabet $\mathcal{A} = \{a, b, c, \cdots\}$

$$\alpha, \beta \quad ::= \quad a \;\mid\; \alpha\beta \;\mid\; \lceil\alpha\rceil \;\mid\; \lfloor\alpha\rfloor$$

atomic

compound

new atomic names (overlined, underlined)

The labeled λ-calculus

- An example: $\underline{\Delta} = \lambda x.(x^c\, x^d)^b, \;\; \Delta = \lambda x.(x^g\, x^h)^f$

$$
\begin{aligned}
\Omega &= \underline{\Delta}^a\, \Delta^e \\
&\rightarrow \Omega_1 = (\Delta^{\gamma_1}\, \Delta^{\delta_1})^{b\lceil a\rceil} && \gamma_1 = e\lfloor a\rfloor c && \delta_1 = e\lfloor a\rfloor d \\
&\rightarrow \Omega_2 = (\Delta^{\gamma_2}\, \Delta^{\delta_2})^{f\lceil \gamma_1\rceil b\lceil a\rceil} && \gamma_2 = \delta_1\lfloor \gamma_1\rfloor g && \delta_2 = \delta_1\lfloor \gamma_1\rfloor h \\
&\rightarrow \Omega_3 = (\Delta^{\gamma_3}\, \Delta^{\delta_3})^{f\lceil \gamma_2\rceil f\lceil \gamma_1\rceil b\lceil a\rceil} && \gamma_3 = \delta_2\lfloor \gamma_2\rfloor g && \delta_3 = \delta_2\lfloor \gamma_2\rfloor h \\
&\rightarrow \cdots
\end{aligned}
$$

- or simpler with partial labels: $\Delta = \lambda x.x\, x$

$$
\begin{aligned}
\Omega &= \Delta^a\, \Delta \\
&\rightarrow \Omega_1 = (\Delta^{\gamma_1}\, \Delta^{\gamma_1})^{\lceil a\rceil} && \gamma_1 = \lfloor a\rfloor \\
&\rightarrow \Omega_2 = (\Delta^{\gamma_2}\, \Delta^{\gamma_2})^{\lceil \gamma_1\rceil \lceil a\rceil} && \gamma_2 = \gamma_1\lfloor \gamma_1\rfloor \\
&\rightarrow \Omega_3 = (\Delta^{\gamma_3}\, \Delta^{\gamma_3})^{\lceil \gamma_2\rceil \lceil \gamma_1\rceil \lceil a\rceil} && \gamma_3 = \gamma_2\lfloor \gamma_2\rfloor \\
&\rightarrow \cdots
\end{aligned}
$$

The labeled λ-calculus

- the labeled calculus is confluent
- the labeled calculus is strongly normalizable when reduction is restricted to a finite set of redex names
- unique normal form when exists
- the standard λ-calculus can be seen as an infinite limit of finite labeled-calculi

The labeled λ-calculus

$\Delta = \lambda x.(x^c x^d)^b$

$F = \lambda f.(f^k\, y^\ell)^j$

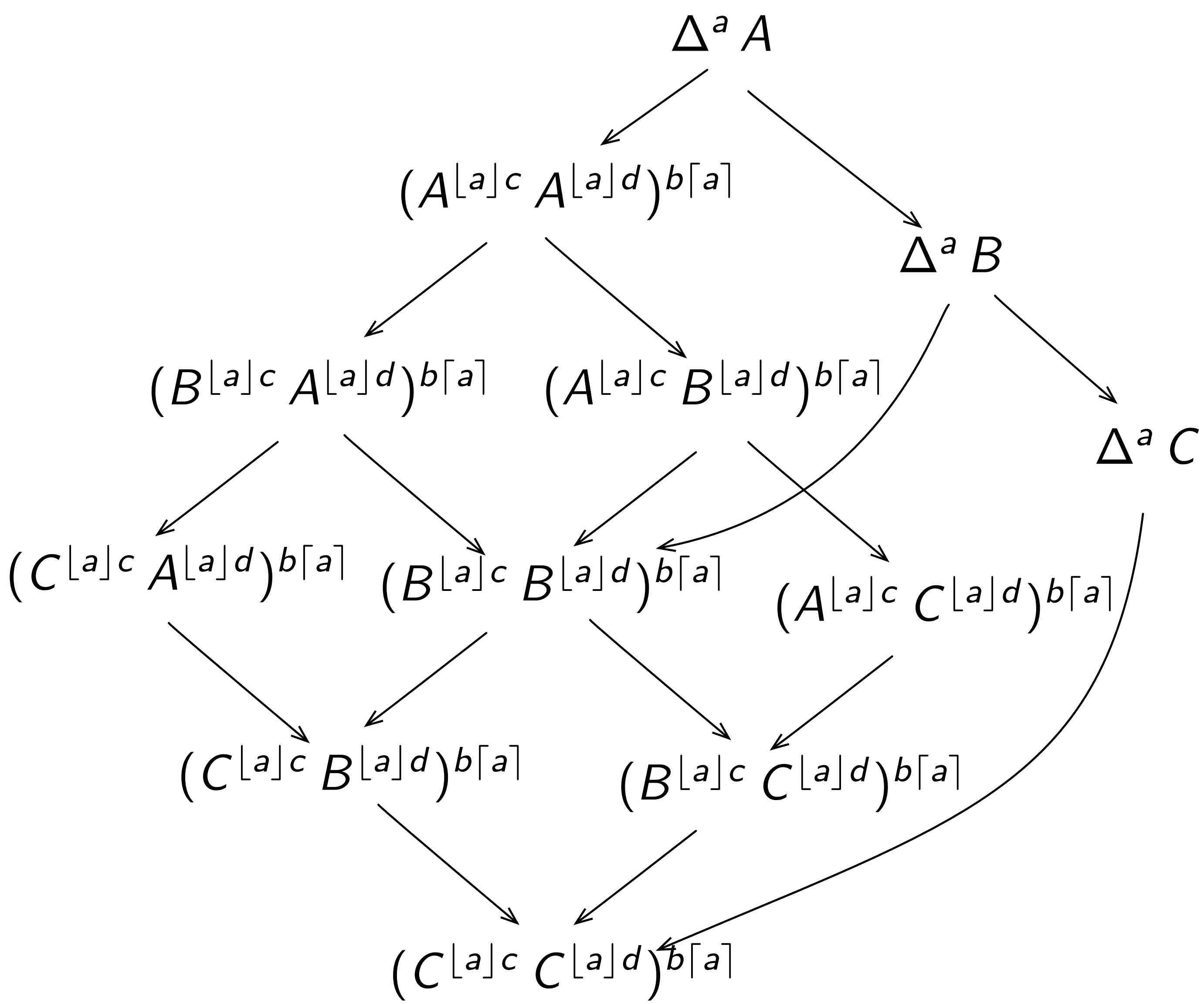
$I = \lambda x.x^v$

$A = (F^i\, I^u)^q$

$B = (I^\gamma\, y^\ell)^q$

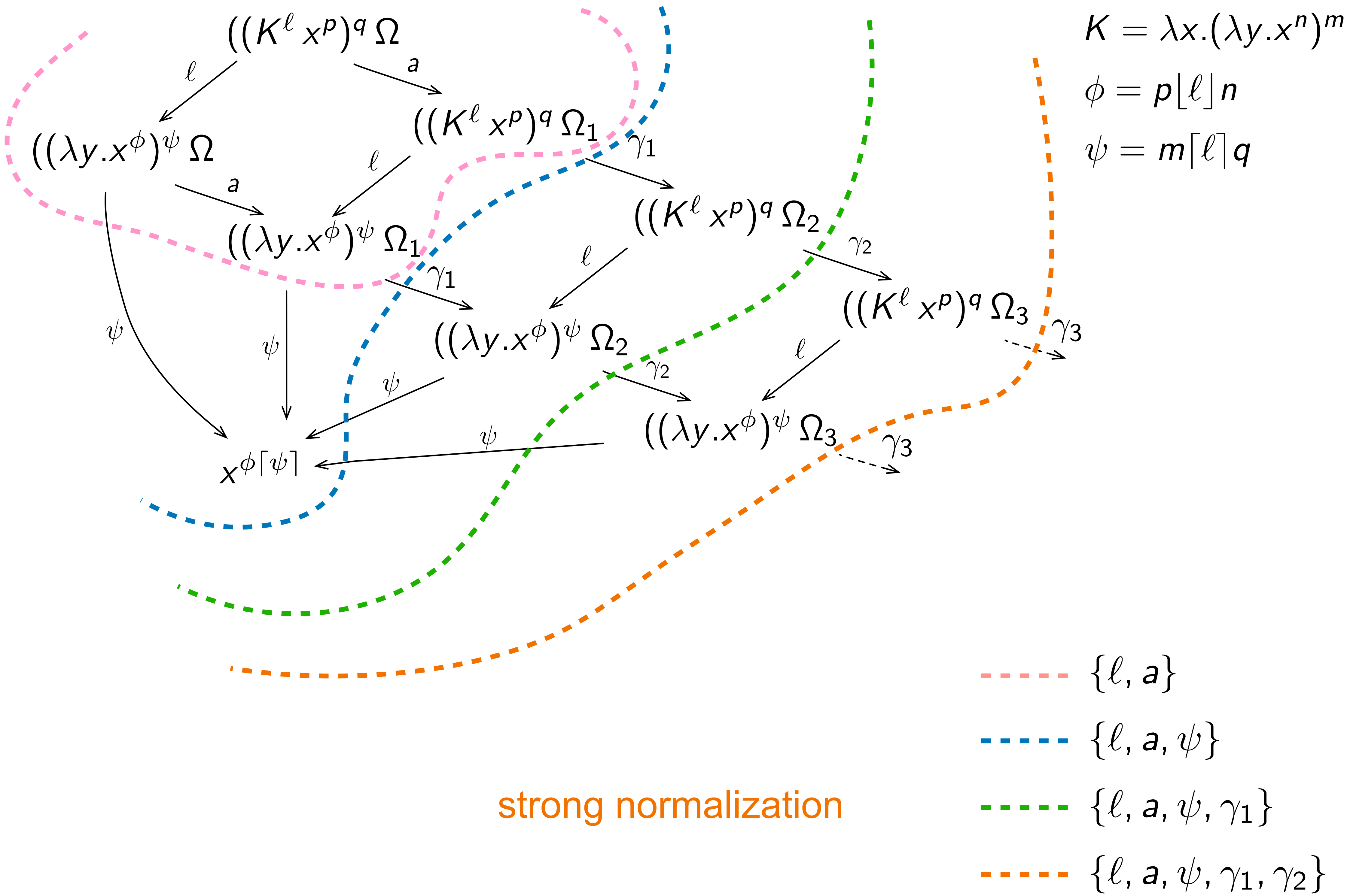
$C = y^{\ell\lfloor\gamma\rfloor v\lceil\gamma\rceil q}$

$\gamma = u\lfloor i\rfloor k$



confluence

The labeled λ-calculus



The labeled λ-calculus

$$(\lambda x. \cdots (x^{\beta}\, N) \cdots)^{\alpha}\, (\lambda y.M)^{\gamma} \rightarrow \cdots ((\lambda y.M)^{\gamma\lfloor\alpha\rfloor\beta}\, N') \cdots$$

α creates $\gamma\lfloor\alpha\rfloor\beta$

$$((\lambda x.(\lambda y.M)^{\gamma})^{\alpha} N)^{\beta}\, P \rightarrow (\lambda y.M')^{\gamma\lceil\alpha\rceil\beta} P$$

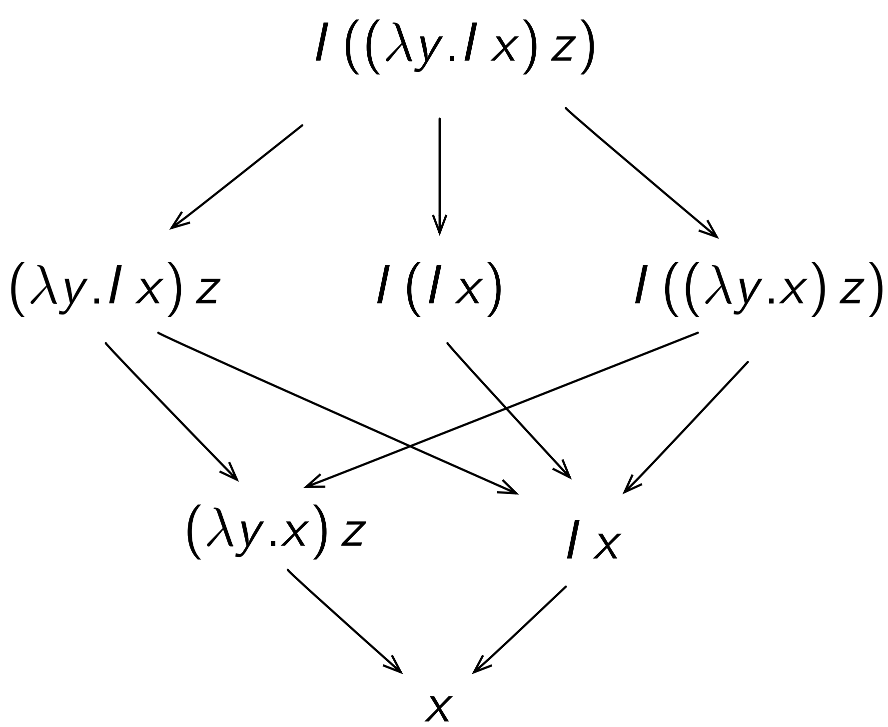
α creates $\gamma\lceil\alpha\rceil\beta$

$$((\lambda x.\, x^{\gamma})^{\alpha}\, (\lambda y.M)^{\delta})^{\beta}\, N \rightarrow (\lambda y.M)^{\delta\lfloor\alpha\rfloor\gamma\lceil\alpha\rceil\beta} N$$

α creates $\delta\lfloor\alpha\rfloor\gamma\lceil\alpha\rceil\beta$

origin

The labeled λ-calculus

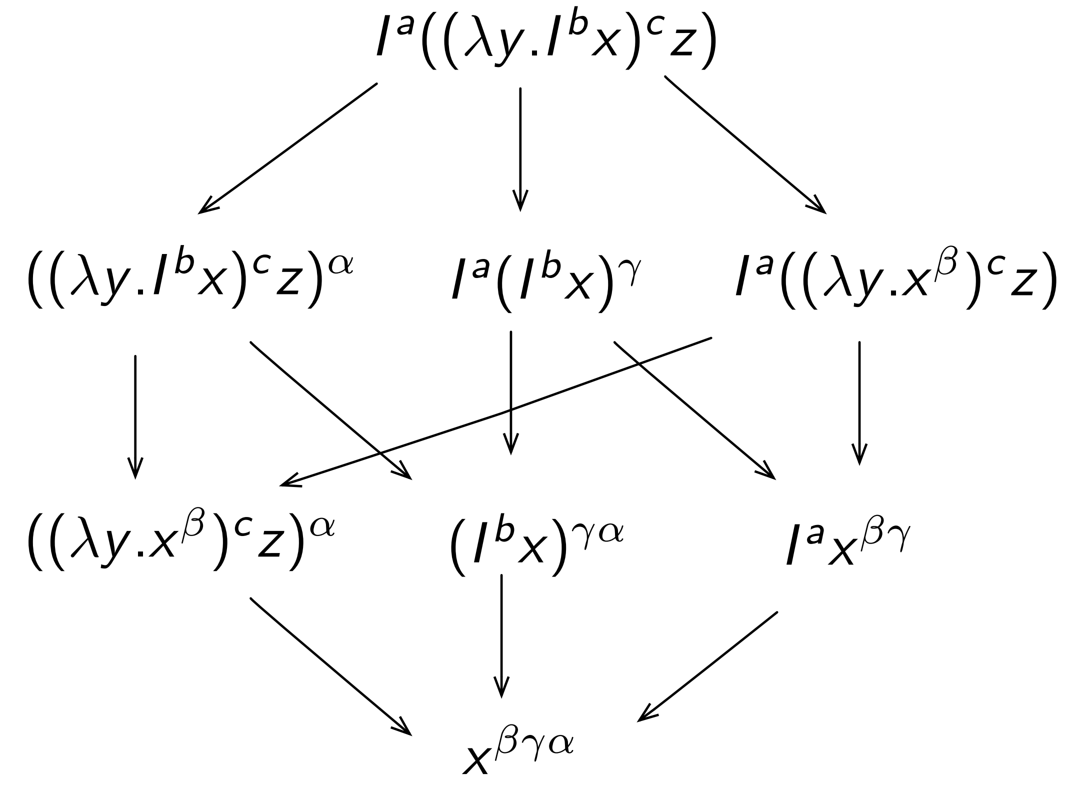


$I = \lambda x.x$

$\alpha = \lfloor a \rfloor \lceil a \rceil$

$\beta = \lfloor b \rfloor \lceil b \rceil$

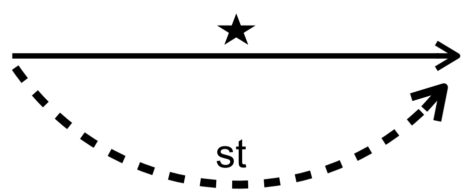
$\gamma = \lceil c \rceil$



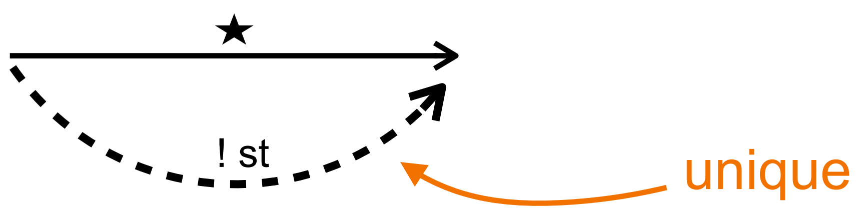
lattice

The labeled λ-calculus

- a **standard** reduction is an outside-in left-to-right reduction strategy
- any reduction can be reordered in a standard reduction `[Curry 1958]`



λ-calculus

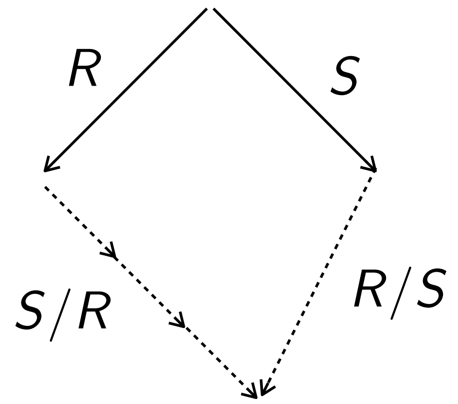


labeled λ-calculus

permutation equivalence

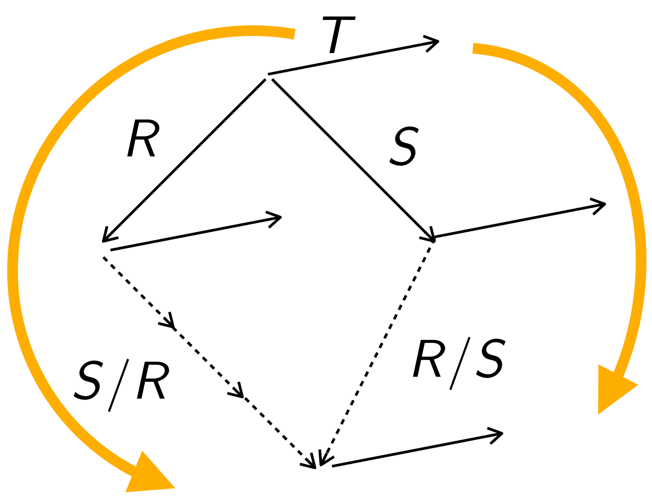
Permutation equivalence

- single-redex reduction steps $M \xrightarrow{R} N$
- residuals S/R of another redex S in M are **disjoint** redexes
  - let $\mathcal{F}$ be a set of disjoint redexes
  - write $\rho : \mathcal{F}$ for any single-redex reduction ρ of redexes of $\mathcal{F}$ in any order
  - these reductions are all cofinal (end on a same term)
- single-redex reductions are locally confluent



Permutation equivalence

- moreover, let T be a another redex in M
- residuals of T on both sides of the permutation are the **same**



$$T/(R \sqcup S) = T/(S \sqcup R)$$

the cube lemma

$$T/(R\,;(S/R)) = T/(S;(R/S))$$

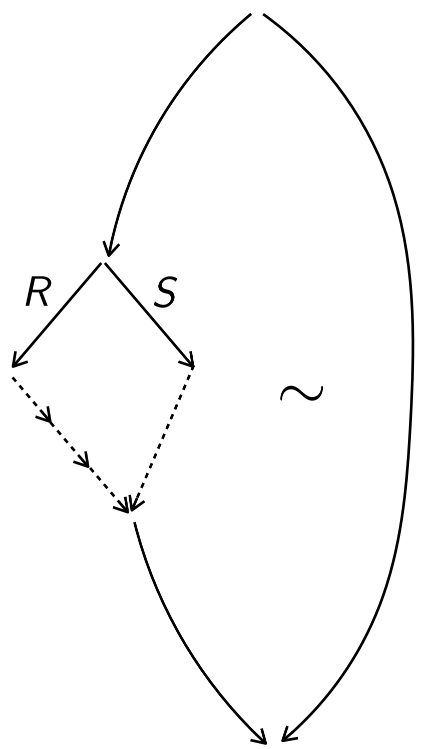
Permutation equivalence

- definition with permutations

$\sim$ is the smallest equivalence relation such that:

$$(i) \quad R \sqcup S \;\sim\; S \sqcup R$$

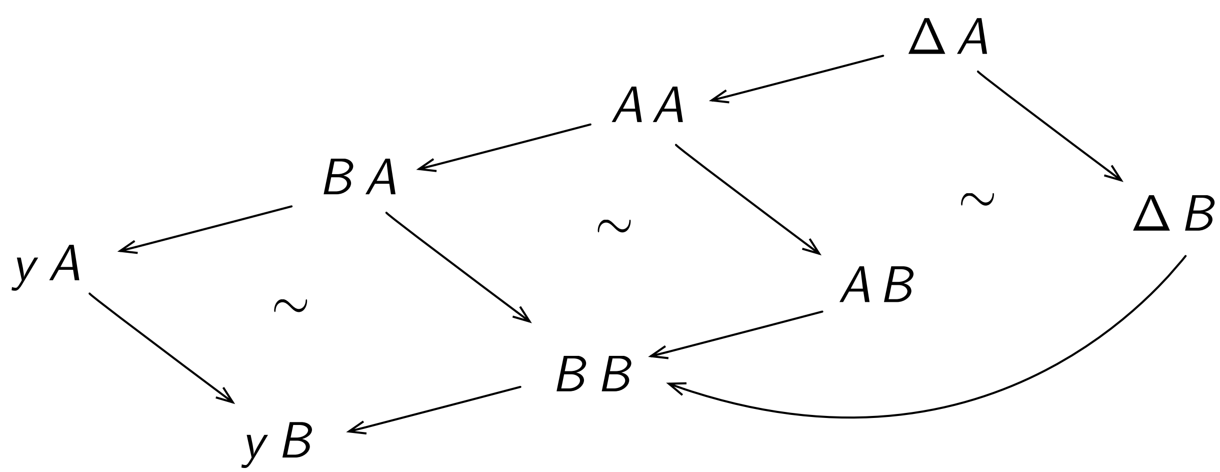
$$(ii) \quad \rho \sim \sigma \implies \tau\,;\rho\,;\upsilon \sim \tau\,;\sigma\,;\upsilon$$



aka parse trees for contex-free languages

Permutation equivalence

- example



$$\Delta = \lambda x.x\,x$$

$$F = \lambda f.f\,y$$

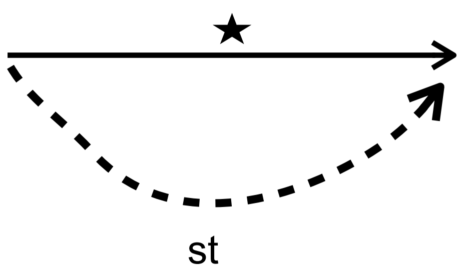
$$I = \lambda x.x$$

$$A = F\,I$$

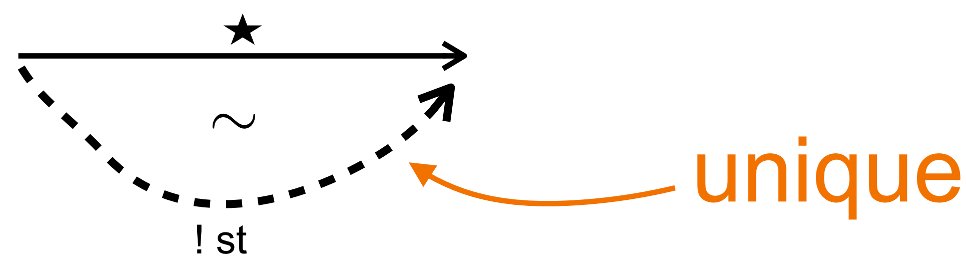
$$B = I\,y$$

Permutation equivalence

- a **standard** reduction is an outside-in left-to-right reduction strategy
- any reduction is equivalent by permutations to a unique standard reduction



λ-calculus



λ-calculus with permutations

- standard reductions are canonical representatives in equivalence classes

Notation

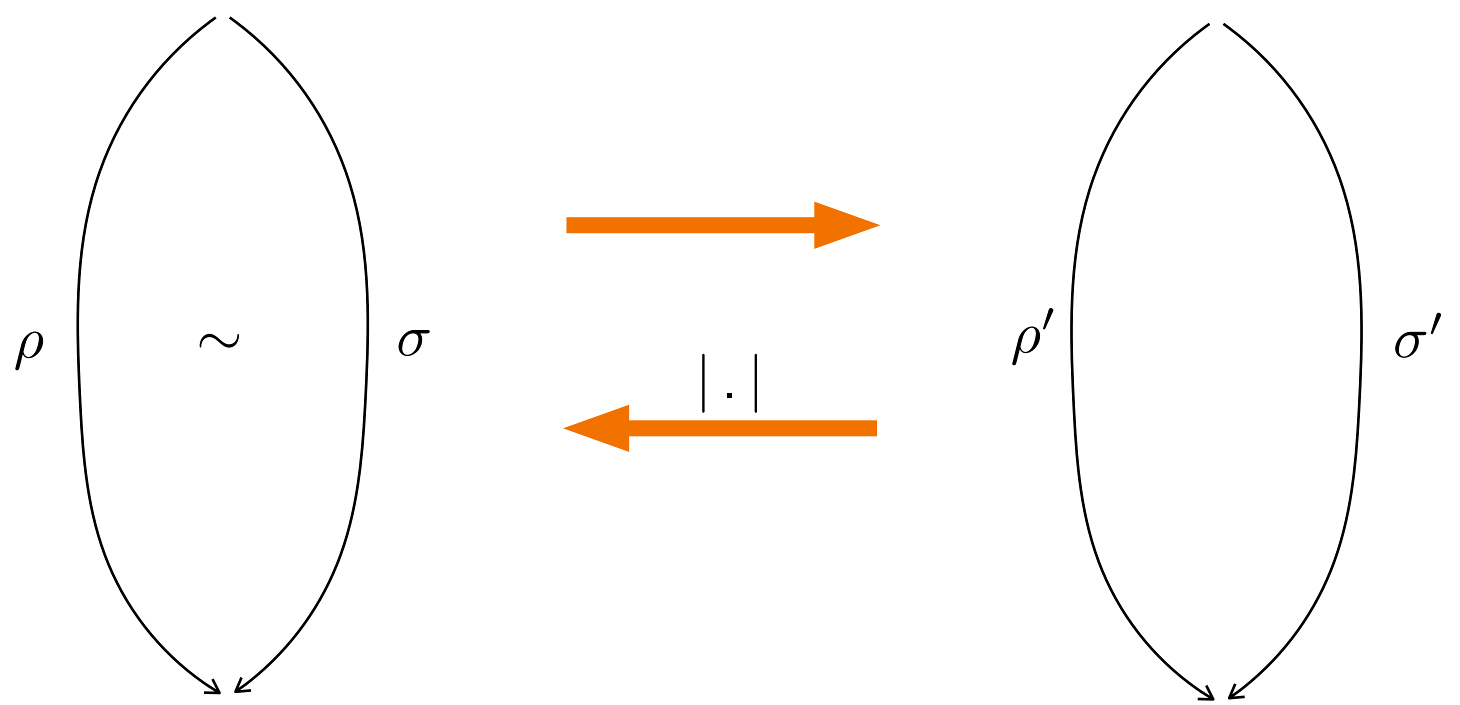
- usual **λ**-calculus $M, N, P, \ldots$
- labeled **λ**-calculus $U, V, W, \ldots$
- forgetful functor $M = |U|$ by erasing labels

Permutation equivalence

- $\sim$ corresponds to the coinitial / cofinal reductions of the labeled **λ**-calculus

Let $\rho = |\rho'|,\ \sigma = |\sigma'|$. Then

$$\rho \sim \sigma \quad \Longleftrightarrow \quad \mathsf{org}(\rho') = \mathsf{org}(\sigma') \ \wedge \ \mathsf{end}(\rho') = \mathsf{end}(\sigma')$$



λ-calculus

$\rho \sim \sigma$

labeled **λ**-calculus

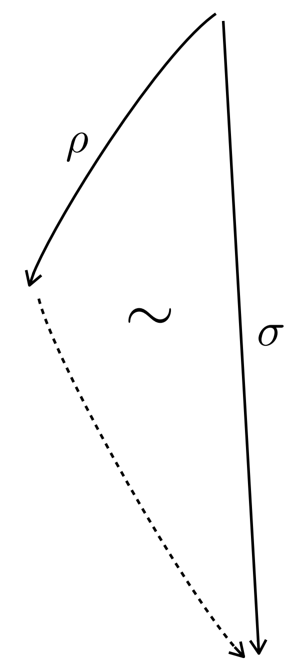
$\mathsf{org}(\rho') = \mathsf{org}(\sigma')$

$\mathsf{end}(\rho') = \mathsf{end}(\sigma')$

Prefix modulo permutations

- $\leq$ is simply defined by:

$$\rho \leq \sigma \iff \exists \tau,\ \rho;\tau \sim \sigma$$



Prefix modulo permutations

- properties of prefix up-to $\sim$

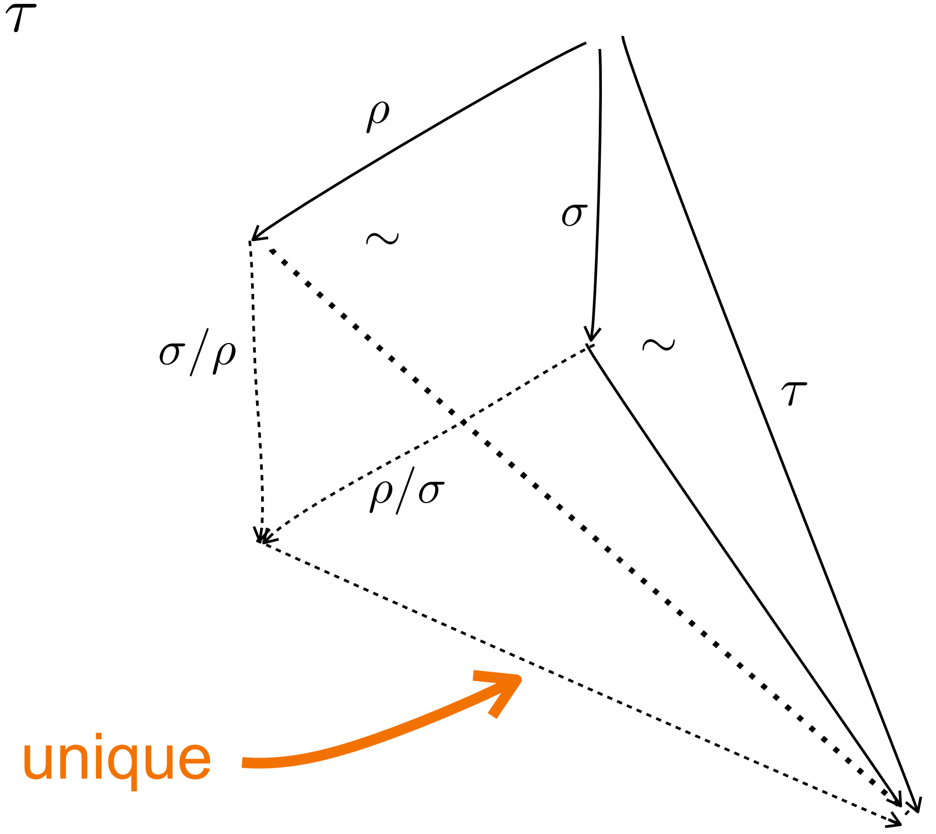
$$(i) \quad \rho \leq \rho \sqcup \sigma$$

$$(ii) \quad \sigma \leq \rho \sqcup \sigma$$

$$(iii) \quad \rho \leq \tau,\ \sigma \leq \tau \implies \rho \sqcup \sigma \leq \tau$$

sup-lattice

pushout

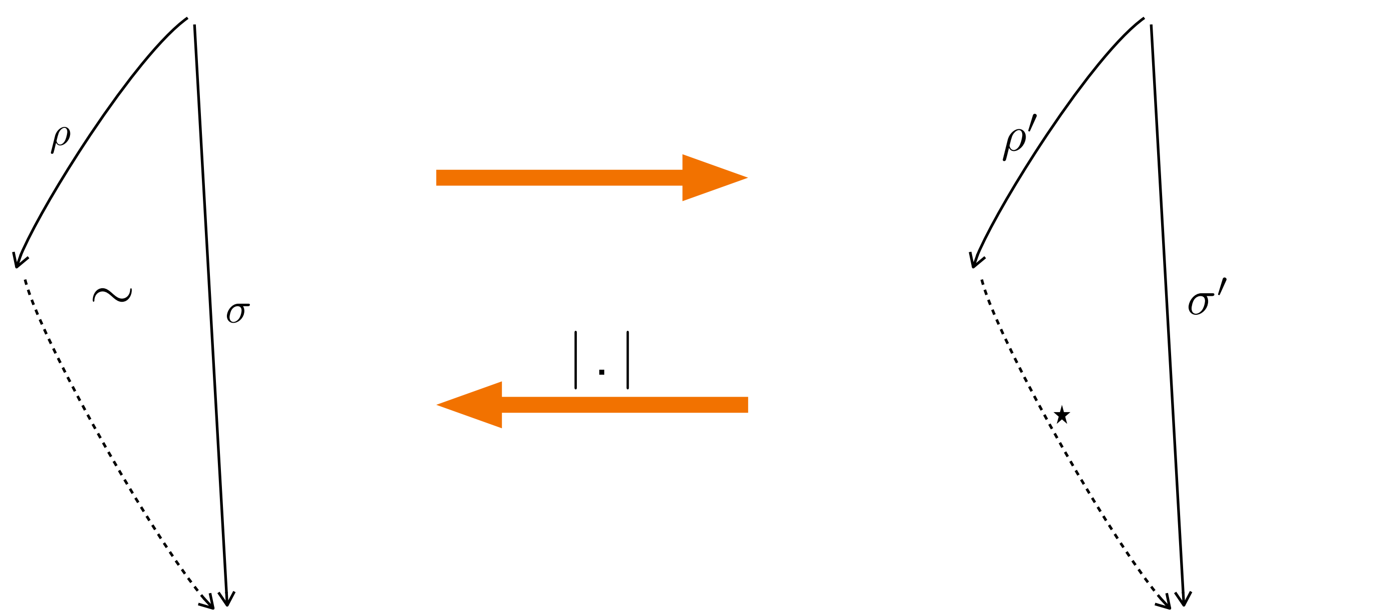


Prefix modulo permutations

- $\leq$ corresponds to reductions of the labeled **λ**-calculus

Let $\rho = |\rho'|$, $\sigma = |\sigma'|$. Then

$$\rho \leq \sigma \iff \mathsf{org}(\rho') = \mathsf{org}(\sigma') \ \wedge \ \mathsf{end}(\rho') \longrightarrow\!\!\!\!\twoheadrightarrow \ \mathsf{end}(\sigma')$$



λ-calculus

$\rho \leq \sigma$

labeled **λ**-calculus

$\mathsf{org}(\rho') = \mathsf{org}(\sigma')$

$\mathsf{end}(\rho') \twoheadrightarrow \mathsf{end}(\sigma')$

redex families

Residuals modulo permutations

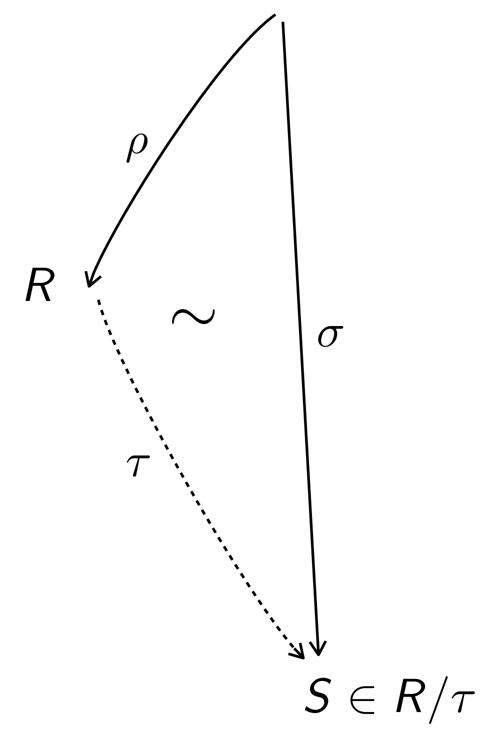
- h-redex $\langle \rho, R \rangle$ is a pair made of a reduction and a redex in its final term

[h-redexes capture histories of redexes]

- a h-redex $\langle \sigma, S \rangle$ is a residual of another h-redex $\langle \rho, R \rangle$ when

$$\exists \tau, \quad \rho ; \tau \sim \sigma \quad \wedge \quad S \in R/\tau$$

- we write then $\langle \rho, R \rangle \lesssim \langle \sigma, S \rangle$



Residuals modulo permutations

- properties of residuals of h-redexes

$$(i) \quad \langle \rho, R \rangle \lesssim \langle \sigma, S \rangle \iff \rho \leq \sigma \ \wedge \ S \in R/(\sigma/\rho)$$

$$(ii) \quad \langle \rho, R \rangle \lesssim \langle \rho, R \rangle$$

$$(iii) \quad \langle \rho, R \rangle \lesssim \langle \sigma, S \rangle \lesssim \langle \rho, R \rangle \iff \rho \sim \sigma \ \wedge \ R = S$$

$$(iv) \quad \langle \rho, R \rangle \lesssim \langle \sigma, S \rangle \lesssim \langle \tau, T \rangle \implies \langle \rho, R \rangle \lesssim \langle \tau, T \rangle$$

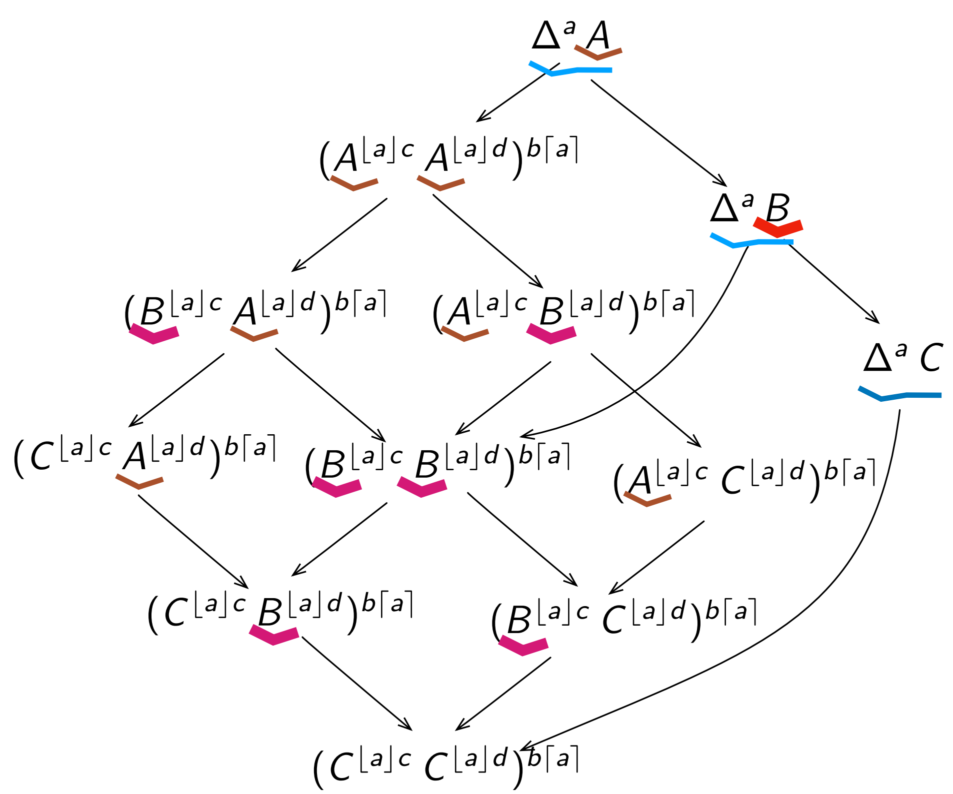
$$(v) \quad \langle \rho, R \rangle \lesssim \langle \tau, T \rangle \ \wedge \ \rho \leq \sigma \leq \tau \implies \exists! S, \ \langle \rho, R \rangle \lesssim \langle \sigma, S \rangle \lesssim \langle \tau, T \rangle$$

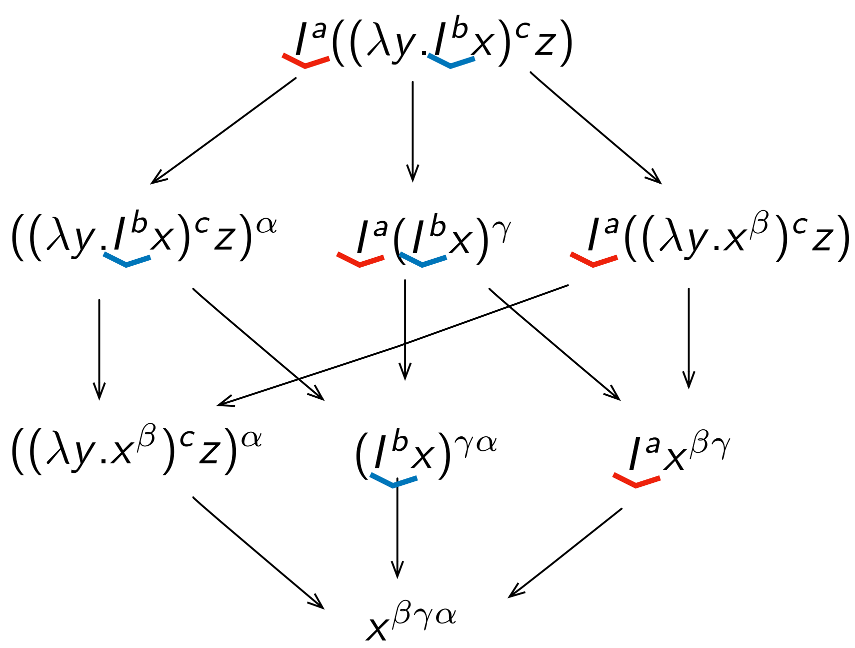
$$(vi) \quad \langle \rho, R \rangle \lesssim \langle \sigma, S \rangle \iff \langle \tau; \rho, \ R \rangle \lesssim \langle \tau; \sigma, \ S \rangle$$

- residuals of h-redexes are consistent with permutation equivalence

Residuals modulo permutations

- residuals of h-redexes correspond to names of redexes in :





$\Delta = \lambda x.(x^c x^d)^b$ $\quad A = (F^i\, I^u)^q$

$F = \lambda f.(f^k\, y^\ell)^j$ $\quad B = (I^\gamma\, y^\ell)^q$

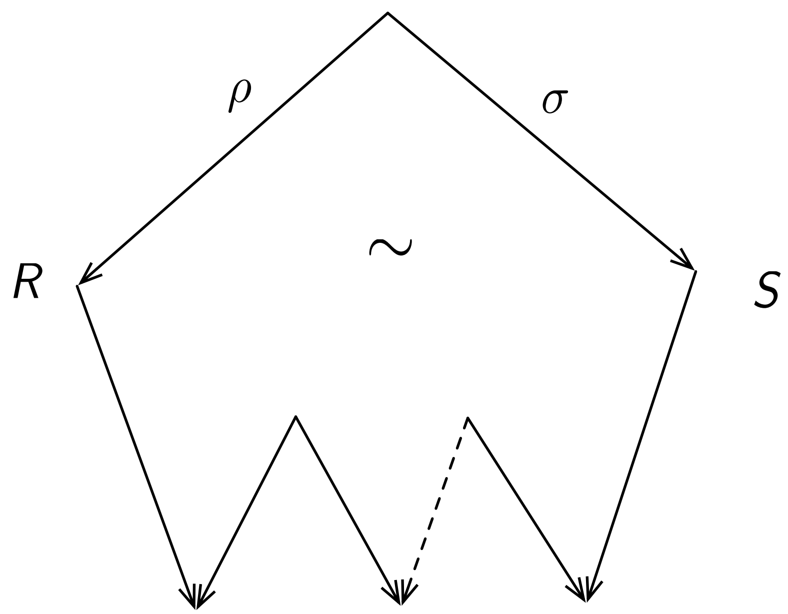
$I = \lambda x.x^v$ $\quad C = y^{\ell \lfloor \gamma \rfloor v \lceil \gamma \rceil q}$

Redex families

- the family relation $\simeq$ between h-redexes is defined by :

$$(i) \quad \langle \rho, R \rangle \lesssim \langle \sigma, S \rangle \implies \langle \rho, R \rangle \simeq \langle \sigma, S \rangle \simeq \langle \rho, R \rangle$$

$$(ii) \quad \langle \rho, R \rangle \simeq \langle \sigma, S \rangle \simeq \langle \tau, T \rangle \implies \langle \rho, R \rangle \simeq \langle \tau, T \rangle$$



- symmetric + transitive closure of residuals modulo permutations

Redex families

- from now on, we only consider standard reductions
- then the extraction relation $\lhd$ on h-redexes is defined as follows

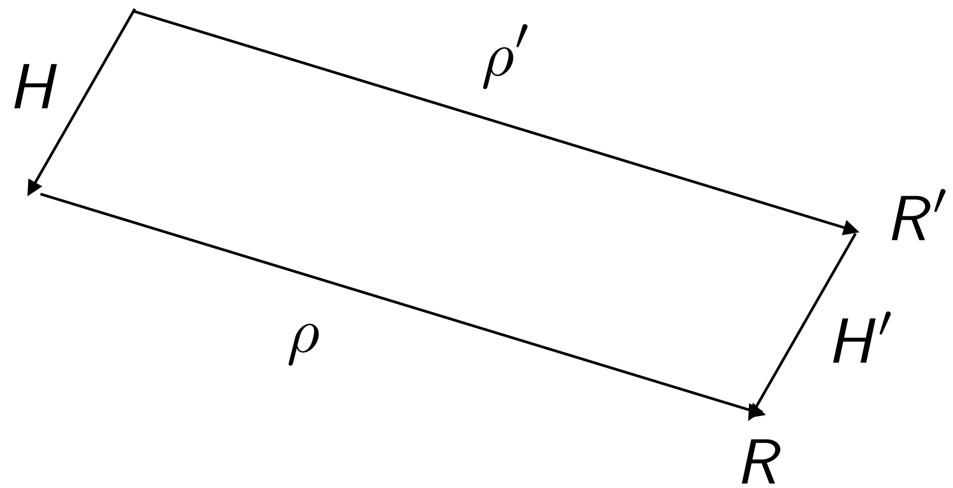
$$(i) \quad \langle o, R \rangle \lhd \langle o, R \rangle$$

$$(ii) \quad \langle \rho, R \rangle \lhd \langle \sigma, S \rangle \implies \langle \rho', R' \rangle \lhd \langle H; \sigma, S \rangle$$

where $\langle \rho', R' \rangle$ is defined by cases analysis on ρ w.r.t. H

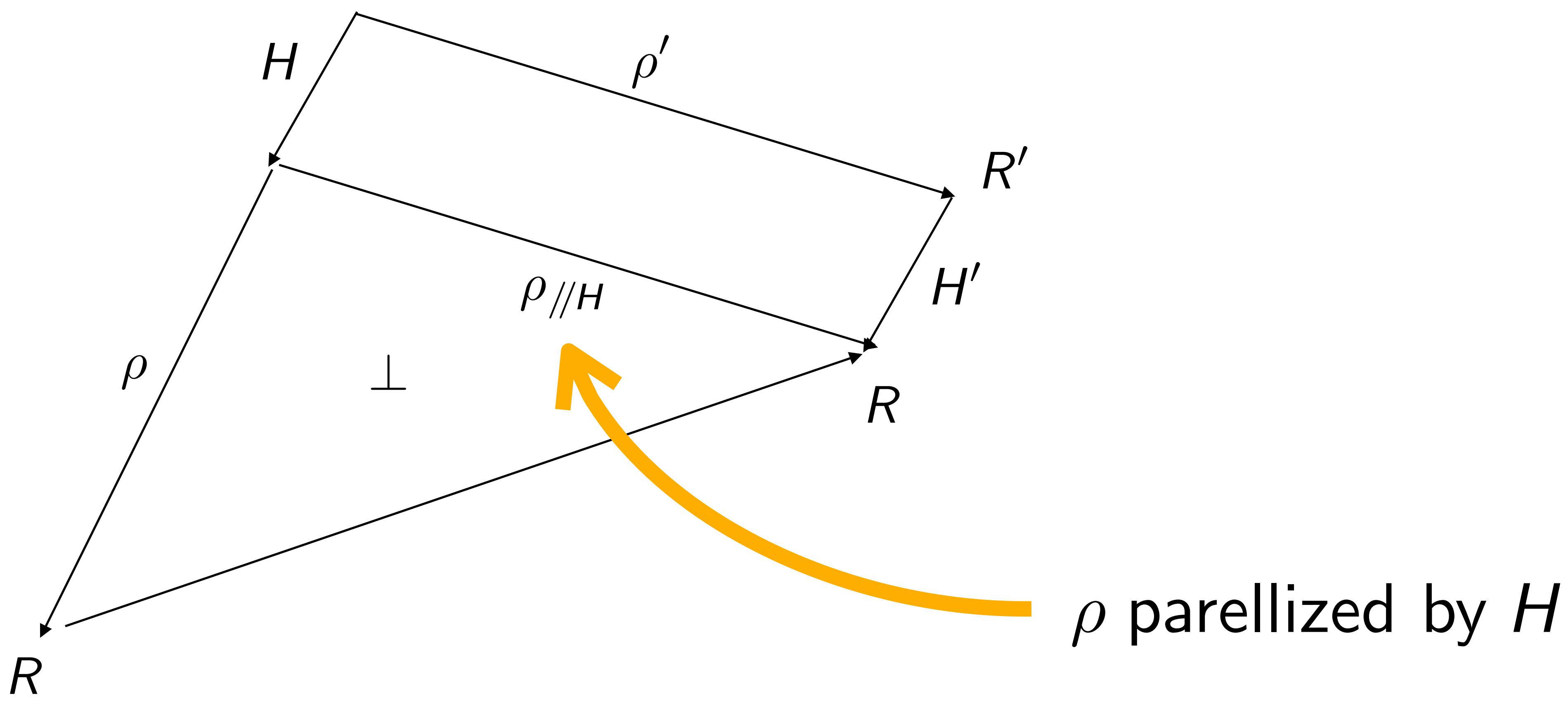
Redex families

- **Case 1:** ρ is in body of *H* or disjoint to the right of the contractum of *H* then ρ' is isomorphic to ρ



Redex families

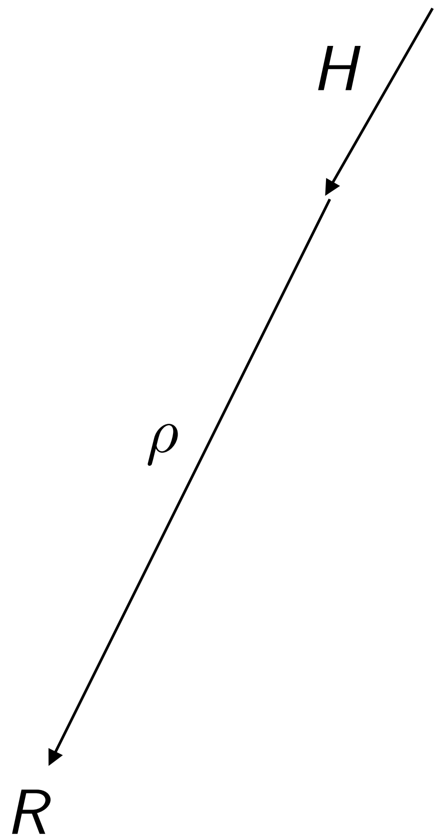
- **Case 2:** ρ is internal to an instance of a copy of the argument of *H* then ρ' is isomorphic to ρ in the argument of *H*



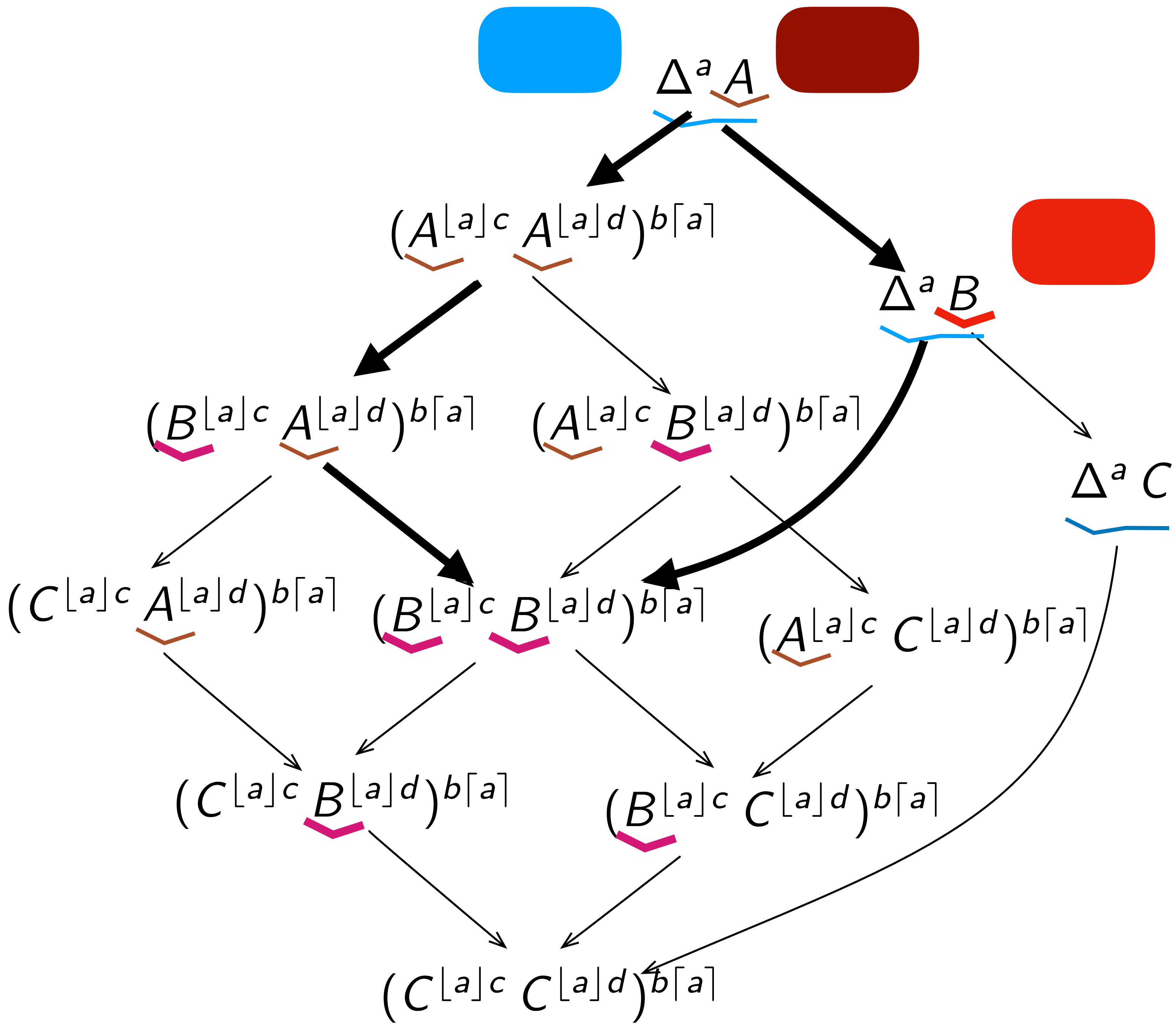
Redex families

- **Otherwise** (*H* necessary for *R*)

$$\rho' = H; \rho \quad \wedge \quad R' = R$$



Redex families



Redex families

- the family relation $\simeq$ can be decided by extraction

$$\langle \rho, R \rangle \simeq \langle \sigma, S \rangle \iff \langle \tau, T \rangle \lhd \langle \rho, R \rangle \;\wedge\; \langle \tau, T \rangle \lhd \langle \sigma, S \rangle \quad \text{for some } \langle \tau, T \rangle$$

- in fact $\langle \tau, T \rangle$ is unique and is the canonical representative of its family

- $\langle \tau, T \rangle$ is unique in family with minimum length of (standard) reduction

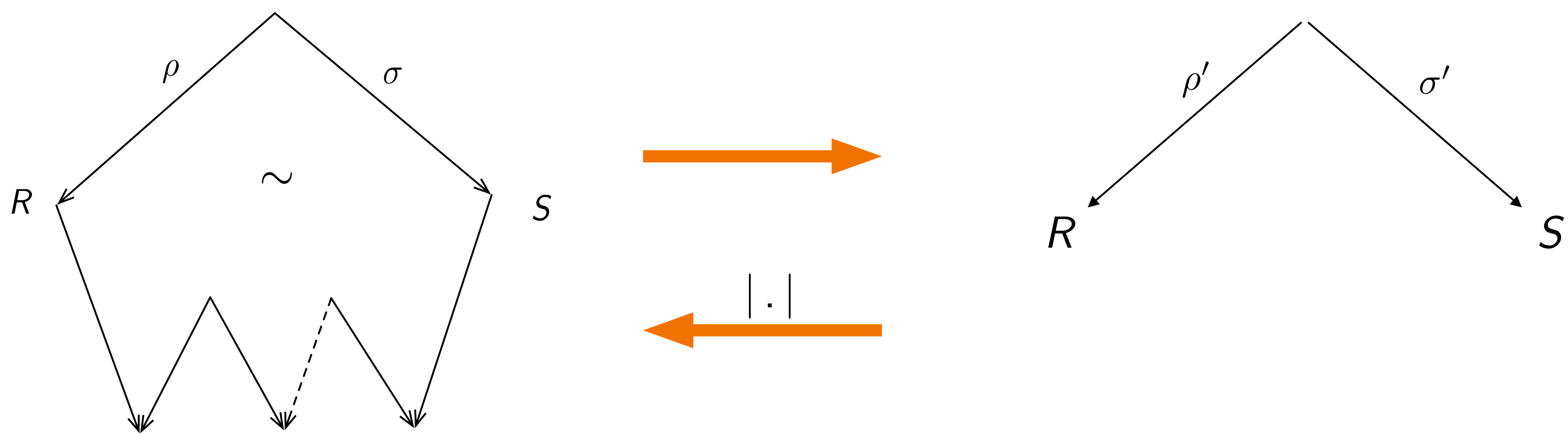
redexes are stable in the λ-calculus

sequentiality

Redex families

- the family relation $\simeq$ corresponds to names in the labeled calculus

$$\langle \rho, R \rangle \simeq \langle \sigma, S \rangle \iff \langle \tau, T \rangle \lhd \langle \rho, R \rangle \;\wedge\; \langle \tau, T \rangle \lhd \langle \sigma, S \rangle \quad \text{for some } \langle \tau, T \rangle$$



λ-calculus

$\langle \rho, R \rangle \simeq \langle \sigma, S \rangle$

labeled **λ**-calculus

$\mathsf{name}(R) = \mathsf{name}(S)$

when initial labeling with distinct letters

Extra properties

- algebraic laws with parallel reductions of redexes
- residuals of parallel reductions
- optimality of family complete reductions
- family complete reductions are the duplication complete
- reductions with ultra sharing [Lamping]
- connections with linear logic (without boxes)
- generalization to other systems (interaction systems, …)

Conclusion

- real implementations of sharing (more than call-by-need) ?

  [non exponential implementations]

- subsets where possible manageable sharing (weak calculi, others?)

- intuitive proofs of strong normalization

- simplification of the extraction process

- history-based information flow

- incremental computations (makefiles [Vesta], neural networks)