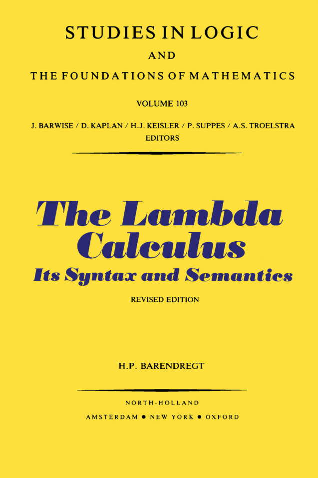
Mechanising Böhm Trees and $\lambda\eta$-Completeness

Chun Tian[1] Michael Norrish[1]

[1]School of Computing, The Australian National University (ANU)

September 29, 2025

Project Context

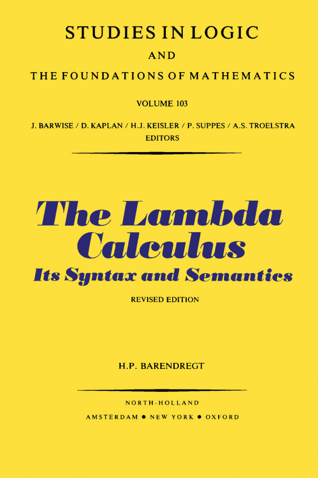


Long term, slow motion campaign to mechanise ∼600pp of famous fundamental computer science.

Title notwithstanding, very much the untyped λ-calulus.

Proofs mostly from this original; some use of more recent contributions (*e.g.*, Takahashi).

Project Context



Long term, slow motion campaign to mechanise ∼600pp of famous fundamental computer science.

Title notwithstanding, very much the untyped λ-calulus.

Proofs mostly from this original; some use of more recent contributions (*e.g.*, Takahashi).

Earlier work proved standardisation, finiteness of developments, CR, . . .

Untyped (type-free) λ-calculus and λ-theories

The set of λ-terms Λ is defined inductively:

- $x \in \Lambda$; (x is an arbitrary *variable*)
- $M \in \Lambda \Rightarrow (\lambda x.\, M) \in \Lambda$;
- $M, N \in \Lambda \Rightarrow (MN) \in \Lambda$;

Assert α-conversion, a syntactic identity :

- $\lambda x.\, M \equiv \lambda y.\, M[x := y]$

where y is not free or bound in M, so $(\lambda x.\, x) \equiv (\lambda y.\, y)$

Untyped (type-free) λ-calculus and λ-theories

The set of λ-terms Λ is defined inductively:

- $x \in \Lambda$; (x is an arbitrary *variable*)
- $M \in \Lambda \Rightarrow (\lambda x.\, M) \in \Lambda$;
- $M, N \in \Lambda \Rightarrow (MN) \in \Lambda$;

The theory $\boldsymbol{\lambda}$ has as formulas $M = N$ where $M, N \in \Lambda$ and is axiomatized by the following axioms (rules):

- $(\lambda x.\, M)N = M[x := N]$; ($\beta$-conversion)
- $M = M$;
- $M = N \Rightarrow N = M$;
- $M = N \wedge N = L \Rightarrow M = L$;
- $M = N \Rightarrow MZ = NZ$;
- $M = N \Rightarrow ZM = ZN$;
- $M = N \Rightarrow \lambda x.\, M = \lambda x.\, N$.

Assert α-conversion, a syntactic identity :

- $\lambda x.\, M \equiv \lambda y.\, M[x := y]$

where y is not free or bound in M, so $(\lambda x.\, x) \equiv (\lambda y.\, y)$

Untyped (type-free) λ-calculus and λ-theories

The set of λ-terms Λ is defined inductively:

- $x \in \Lambda$; (x is an arbitrary *variable*)
- $M \in \Lambda \Rightarrow (\lambda x.\, M) \in \Lambda$;
- $M, N \in \Lambda \Rightarrow (MN) \in \Lambda$;

The theory $\boldsymbol{\lambda}$ has as formulas $M = N$ where $M, N \in \Lambda$ and is axiomatized by the following axioms (rules):

- $(\lambda x.\, M)N = M[x := N]$;   ($\beta$-conversion)
- $M = M$;
- $M = N \Rightarrow N = M$;
- $M = N \wedge N = L \Rightarrow M = L$;
- $M = N \Rightarrow MZ = NZ$;
- $M = N \Rightarrow ZM = ZN$;
- $M = N \Rightarrow \lambda x.\, M = \lambda x.\, N$.

Assert α-conversion, a syntactic identity :

- $\lambda x.\, M \equiv \lambda y.\, M[x := y]$

where y is not free or bound in M, so $(\lambda x.\, x) \equiv (\lambda y.\, y)$

The theory $\boldsymbol{\lambda\eta}$ is defined by adding one rule into $\boldsymbol{\lambda}$:

- $\lambda x.\, Mx = M$ (x is not free in M);   (η-conversion)

Untyped (type-free) λ-calculus and λ-theories

The set of λ-terms Λ is defined inductively:

- $x \in \Lambda$; (x is an arbitrary *variable*)
- $M \in \Lambda \Rightarrow (\lambda x.\, M) \in \Lambda$;
- $M, N \in \Lambda \Rightarrow (MN) \in \Lambda$;

The theory $\boldsymbol{\lambda}$ has as formulas $M = N$ where $M, N \in \Lambda$ and is axiomatized by the following axioms (rules):

- $(\lambda x.\, M)N = M[x := N]$; ($\beta$-conversion)
- $M = M$;
- $M = N \Rightarrow N = M$;
- $M = N \wedge N = L \Rightarrow M = L$;
- $M = N \Rightarrow MZ = NZ$;
- $M = N \Rightarrow ZM = ZN$;
- $M = N \Rightarrow \lambda x.\, M = \lambda x.\, N$.

Assert α-conversion, a syntactic identity :

- $\lambda x.\, M \equiv \lambda y.\, M[x := y]$

where y is not free or bound in M, so $(\lambda x.\, x) \equiv (\lambda y.\, y)$

The theory $\boldsymbol{\lambda\eta}$ is defined by adding one rule into $\boldsymbol{\lambda}$:

- $\lambda x.\, Mx = M$ (x is not free in M); (η-conversion)

Provability in $\boldsymbol{\lambda}$ of an equation $M = N$ is denoted by $\boldsymbol{\lambda} \vdash M = N$, or $M =_\beta N$.
Similarily, provability in $\boldsymbol{\lambda\eta}$ is denoted by $\boldsymbol{\lambda\eta} \vdash M = N$, or $M =_{\beta\eta} N$.

Other λ-theories $\mathcal{T}$ are $\boldsymbol{\lambda}$ with different extra axioms, e.g. $\boldsymbol{\lambda} + (P = Q)$ is the theory adding $P = Q$ into $\boldsymbol{\lambda}$. In this case, e.g., $\boldsymbol{\lambda} + (P = Q) \vdash \lambda x.\, P = \lambda x.\, Q$.

Consistency and Completeness of λ-theories

Consistency ("theory is not useless/vacuous")

A formal theory $\mathcal{T}$ (with equations as formulas) is consistent (notation: $\mathrm{Con}(\mathcal{T})$) if $\mathcal{T}$ does not prove every closed equation. Else $\mathcal{T}$ is inconsistent.
If equal in $\mathcal{T}$, Church-Rosser (formalised in many systems) would give combinators S and K a common reduct. As both are in β-normal form, the common reduct would be themselves. But $\mathsf{S} \not\equiv \mathsf{K}$, so $\boldsymbol{\lambda} \nvdash \mathsf{S} = \mathsf{K}$, and so $\mathrm{Con}(\boldsymbol{\lambda})$ (and $\mathrm{Con}(\boldsymbol{\lambda\eta})$ similarly).

Consistency and Completeness of λ-theories

Consistency ("theory is not useless/vacuous")

A formal theory $\mathcal{T}$ (with equations as formulas) is consistent (notation: $\mathrm{Con}(\mathcal{T})$) if $\mathcal{T}$ does not prove every closed equation. Else $\mathcal{T}$ is inconsistent.
If equal in $\mathcal{T}$, Church-Rosser (formalised in many systems) would give combinators S and K a common reduct. As both are in β-normal form, the common reduct would be themselves. But $\mathsf{S} \not\equiv \mathsf{K}$, so $\boldsymbol{\lambda} \nvdash \mathsf{S} = \mathsf{K}$, and so $\mathrm{Con}(\boldsymbol{\lambda})$ (and $\mathrm{Con}(\boldsymbol{\lambda\eta})$ similarly).

(Hilbert-Post) Completeness of $\boldsymbol{\lambda\eta}$ ("you can prove anything that's right" or "Equational theory is 'full' ")

Suppose M, N have $\beta\eta$-normal forms. Then either $\boldsymbol{\lambda\eta} \vdash M = N$ or $\boldsymbol{\lambda\eta} + (M = N)$ is inconsistent.
This is an (easy) corollary of Böhm's separability theorem from 1968[a], never formalised.

[a] C. Böhm. Alcune proprietà delle forme β-η-normali nel λ-k-calcolo. *Pubblicazioni dell'Instituto per le Applicazioni del Calcolo*, 696:1–19, 1968

Outline of this work

- We (first ever) successfully formalised (mechanised) $\lambda\eta$-completeness in HOL4, following Barendregt[1].
- We did NOT fully formalise Böhm's separability theorem (full version is still in progress), but only obtained (with less effort) a restricted version (with extra antecedents), sufficient to prove completeness.
- From this restricted version of the separability theorem to $\lambda\eta$-completeness, we have a novel proof, differing from Barendregt (and Böhm).
- The modern proof of the separability theorem (retold by Barendregt) involves a coinductive data structure called the Böhm tree, which is hard to formalise. We formally defined it in a "first-order" style, and used it to prove the separability theorem.
- Our formal Böhm trees require a novel (smart) way of allocating fresh names, possibly useful for other purposes.

[1] H. P. Barendregt. *The Lambda Calculus, its Syntax and Semantics*, volume 40 of *Studies in Logic*. North-Holland Publishing Company, 1984

Preliminaries: λ-terms by Nominal Datatype

The existing[2] λ-calculus mechanisation in HOL4 provides the type `term` with three constructors:

- `VAR` x for the λ-term made of a single variable x (whose type is `string`);
- `LAM` x t for the abstraction $\lambda x.\, t$ where t is another λ-term and x is a string;
- $t \cdot t'$ (or `APP` t t') for an applications such as $t\, t'$ where t and t' are λ-terms.

The type `term` is *nominal*: terms which are α-equivalent are equal, e.g.,

$$(\lambda x.\, x) = (\lambda y.\, y)$$

or (as a theorem in HOL4):

$\vdash$ `LAM "x" (VAR "x") = LAM "y" (VAR "y")`

[2] M. Norrish. Mechanising λ-calculus using a classical first order theory of terms with permutations. *Higher-Order and Symbolic Computation*, 19(2-3):169–195, Sept. 2006

Preliminaries: λ-terms by Nominal Datatype

The existing[2] λ-calculus mechanisation in HOL4 provides the type `term` with three constructors:

- `VAR` x for the λ-term made of a single variable x (whose type is `string`);
- `LAM` x t for the abstraction $\lambda x.\, t$ where t is another λ-term and x is a string;
- $t \cdot t'$ (or `APP` t t') for an applications such as $t\, t'$ where t and t' are λ-terms.

The type `term` is *nominal*: terms which are α-equivalent are equal, e.g.,

$$(\lambda x.\, x) = (\lambda y.\, y)$$

or (as a theorem in HOL4):

$$\vdash \texttt{LAM}\ \text{“x”}\ (\texttt{VAR}\ \text{“x”})\ \texttt{=}\ \texttt{LAM}\ \text{“y”}\ (\texttt{VAR}\ \text{“y”})$$

Proof (by the following basic theorem derived from the nominal package):

$$\begin{aligned}&\vdash \texttt{LAM}\ u\ t_1\ \texttt{=}\ \texttt{LAM}\ v\ t_2 \iff\\ &\quad u\ \texttt{=}\ v \land t_1\ \texttt{=}\ t_2 \lor u \neq v \land u \ \sharp\ t_2 \land t_1\ \texttt{=}\ \texttt{tpm}\ [(u,v)]\ t_2\end{aligned}$$

[2] M. Norrish. Mechanising λ-calculus using a classical first order theory of terms with permutations. *Higher-Order and Symbolic Computation*, 19(2-3):169–195, Sept. 2006

Preliminaries: free names and substitutions

- The set of free names occurring in a term M is $\mathrm{FV}\ M$. $x \notin \mathrm{FV}(M)$ is denoted by $x\ \sharp\ M$.
- The result of substituting N for the free occurrences of x in M (textbook notation $M[x := N]$) is denoted by $[N/x]\ M$. For example,

Lemma (Barendregt 2.1.16 (Substitution lemma))

If $x \not\equiv y$ and $x \notin \mathrm{FV}(L)$, then $M[x := N][y := L] \equiv M[y := L][x := N[y := L]]$.

$\vdash x \neq y \land x\ \sharp\ L \Rightarrow [L/y]\ ([N/x]\ M) = [[L/y]\ N/x]\ ([L/y]\ M)$

Preliminaries: free names and substitutions

- The set of free names occurring in a term M is $\mathrm{FV}\ M$. $x \notin \mathrm{FV}(M)$ is denoted by $x\ \sharp\ M$.
- The result of substituting N for the free occurrences of x in M (textbook notation $M[x := N]$) is denoted by $[N/x]\ M$. For example,

Lemma (Barendregt 2.1.16 (Substitution lemma))

If $x \not\equiv y$ *and* $x \notin \mathrm{FV}(L)$, *then* $M[x := N][y := L] \equiv M[y := L][x := N[y := L]]$.

$\vdash x \neq y \land x\ \sharp\ L \Rightarrow [L/y]\ ([N/x]\ M) = [[L/y]\ N/x]\ ([L/y]\ M)$

We also use iterated substitution ($\mathtt{ISUB}$):

$$M\ \mathtt{ISUB}\ [] \stackrel{\text{def}}{=} M$$
$$M\ \mathtt{ISUB}\ ((s,x)::sxs) \stackrel{\text{def}}{=} [s/x]\ M\ \mathtt{ISUB}\ sxs$$

Head Reduction and Head Normal Forms

One-step head-reduction is inductively defined by:

$$\frac{}{(\lambda v.\, M)N \xrightarrow{h} M[v := N]} \qquad \frac{M_1 \xrightarrow{h} M_2}{\lambda v.M_1 \xrightarrow{h} \lambda v.\, M_2} \qquad \frac{M_1 \xrightarrow{h} M_2 \qquad M_1 \text{ is not abstraction}}{M_1 N \xrightarrow{h} M_2 N}$$

Head Reduction and Head Normal Forms

One-step head-reduction is inductively defined by:

$$\frac{}{(\lambda v.\, M)N \xrightarrow{h} M[v := N]} \qquad \frac{M_1 \xrightarrow{h} M_2}{\lambda v. M_1 \xrightarrow{h} \lambda v.\, M_2} \qquad \frac{M_1 \xrightarrow{h} M_2 \qquad M_1 \text{ is not abstraction}}{M_1 N \xrightarrow{h} M_2 N}$$

- For any λ-term, the above rules uniquely determine a head reduction path, either finite or infinite.

Head Reduction and Head Normal Forms

One-step head-reduction is inductively defined by:

$$\frac{}{(\lambda v.\, M)N \xrightarrow{h} M[v := N]} \qquad \frac{M_1 \xrightarrow{h} M_2}{\lambda v. M_1 \xrightarrow{h} \lambda v.\, M_2} \qquad \frac{M_1 \xrightarrow{h} M_2 \qquad M_1 \text{ is not abstraction}}{M_1 N \xrightarrow{h} M_2 N}$$

- For any λ-term, the above rules uniquely determine a head reduction path, either finite or infinite.
- A term M is in head normal form (hnf) if M is of the form $M \equiv \lambda x_0 x_1 \ldots x_{n-1}.\, y\, M_0 M_1 \cdots M_{m-1}$.

Head Reduction and Head Normal Forms

One-step head-reduction is inductively defined by:

$$\frac{}{(\lambda v.\, M)N \xrightarrow{h} M[v := N]} \qquad \frac{M_1 \xrightarrow{h} M_2}{\lambda v.M_1 \xrightarrow{h} \lambda v.\, M_2} \qquad \frac{M_1 \xrightarrow{h} M_2 \qquad M_1 \text{ is not abstraction}}{M_1 N \xrightarrow{h} M_2 N}$$

- For any λ-term, the above rules uniquely determine a head reduction path, either finite or infinite.
- A term M is in head normal form (hnf) if M is of the form $M \equiv \lambda x_0 x_1 \ldots x_{n-1}.\, y\, M_0 M_1 \cdots M_{m-1}$.
- A term M has hnf if it's β-equivalent to a head normal form.

Lemma (Barendregt 11.4.8, corollary of the Standardisation Theorem)

A λ-term has hnf iff its head reduction path is finite:

$\vdash \forall M.$ `has_hnf` $M \iff$ `finite (head_reduction_path` M`)`

- Multi-step head reduction ($M \twoheadrightarrow_h N$) is the RTC of one-step head reduction ($M \rightarrow_h N$).

Principal Head Normal Forms

A term may be β-equivalent to multiple head normal forms (hnf), the principal hnf is particularly important (in the definition of Böhm trees).

Definition (Barendregt 8.3.20)

If M has a hnf, then the last term of the terminating head reduction of M is called the principal head normal form (principal hnf) of M.

$$\texttt{principal\_hnf} \stackrel{\text{def}}{=} \texttt{last} \circ \texttt{head\_reduction\_path}$$

Some properties of principal hnf

$$\vdash \texttt{has\_hnf}\ M \Rightarrow (\texttt{principal\_hnf}\ M = N \iff M \twoheadrightarrow_h N \land \texttt{hnf}\ N)$$

$$\vdash \texttt{has\_hnf}\ M \Rightarrow \mathrm{FV}\ (\texttt{principal\_hnf}\ M) \subseteq \mathrm{FV}\ M$$

$$\vdash \texttt{hnf}\ t \Rightarrow \texttt{principal\_hnf}\ (\texttt{LAMl}\ xs\ t \cdot\cdot\ \texttt{MAP VAR}\ xs) = t$$

Solvable Terms; Wadsworth's Theorem

Definition (Barendregt 8.3.1)

A closed term M is solvable if there exist $N_1 \cdots N_n$ such that $M\,N_1 \cdots N_n =_\beta \mathsf{I}\ (= \lambda x.x)$. An arbitrary M is solvable if a closure $\lambda\vec{x}.\,M$ of M is solvable (this is independent of the choice of $\vec{x}$).

$$\texttt{solvable}\ M \stackrel{\text{def}}{=} \exists\, M'\ Ns.\ M' \in \texttt{closures}\ M \wedge M' \bullet\bullet\ Ns =_\beta \texttt{I}$$
$$\texttt{closures}\ M \stackrel{\text{def}}{=} \{\texttt{LAMl}\ vs\ M \mid vs \mid \texttt{ALL\_DISTINCT}\ vs \wedge \mathrm{FV}\ M \subseteq \texttt{set}\ vs\}$$

An example of unsolvable term $((\lambda x.\,xx)(\lambda x.\,xx))$:

$\vdash$ `unsolvable` Ω

$\vdash \Omega$ = `LAM "x" (VAR "x" • VAR "x") • LAM "x" (VAR "x" • VAR "x")`

Theorem (Barendregt 8.3.14, Wadsworth)

A λ-term is solvable iff it has hnf:

$\vdash$ `solvable` $M \iff$ `has_hnf` M

Solvable Terms; Wadsworth's Theorem

Definition (Barendregt 8.3.1)

A closed term M is solvable if there exist $N_1 \cdots N_n$ such that $M\,N_1 \cdots N_n =_\beta \mathsf{I}\ (= \lambda x.x)$. An arbitrary M is solvable if a closure $\lambda\vec{x}.\,M$ of M is solvable (this is independent of the choice of $\vec{x}$).

$$\texttt{solvable}\ M \stackrel{\text{def}}{=} \exists\, M'\ Ns.\ \ M' \in \texttt{closures}\ M \land M' \bullet\bullet\ Ns =_\beta \texttt{I}$$
$$\texttt{closures}\ M \stackrel{\text{def}}{=} \{\texttt{LAMl}\ vs\ M \mid vs \mid \texttt{ALL\_DISTINCT}\ vs \land \mathrm{FV}\ M \subseteq \texttt{set}\ vs\}$$

An example of unsolvable term $((\lambda x.\,xx)(\lambda x.\,xx))$:

$\vdash$ $\texttt{unsolvable}\ \Omega$
$\vdash$ $\Omega = \texttt{LAM}\ \text{“x”}\ (\texttt{VAR}\ \text{“x”} \bullet \texttt{VAR}\ \text{“x”}) \bullet \texttt{LAM}\ \text{“x”}\ (\texttt{VAR}\ \text{“x”} \bullet \texttt{VAR}\ \text{“x”})$

Theorem (Barendregt 8.3.14, Wadsworth)

A λ-term is solvable iff it has hnf:

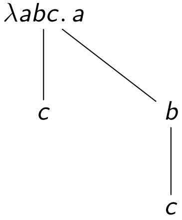
$\vdash$ $\texttt{solvable}\ M \iff \texttt{has\_hnf}\ M$

Two first mechanisations of this announced today!
(Talk after this on paper by Lancelot *et al.*)

Informal Böhm Trees (textbook definitions)

- If a term M is unsolvable, then its Böhm tree, denoted by $\mathrm{BT}(M)$, is $\perp$.
- Otherwise the term has principal hnf $\lambda\vec{x}.\, yM_0 \cdots M_{m-1}$. The root of $\mathrm{BT}(M)$ is $\lambda\vec{x}.\, y$, and the subtrees are $\mathrm{BT}(M_0), \ldots \mathrm{BT}(M_{m-1})$.

Böhm tree examples for $\mathsf{S} \equiv \lambda abc.\, ac(bc)$,

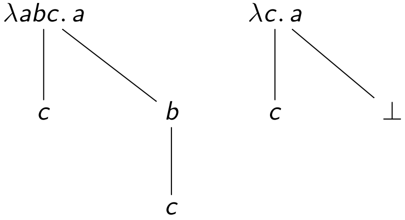


[3]Note that $\mathsf{Y}f =_\beta f(\mathsf{Y}f)$ and $\mathsf{Y} =_\beta \lambda f.\, f(\mathsf{Y}f)$.

Informal Böhm Trees (textbook definitions)

- If a term M is unsolvable, then its Böhm tree, denoted by $\mathrm{BT}(M)$, is $\perp$.
- Otherwise the term has principal hnf $\lambda\vec{x}.\, yM_0 \cdots M_{m-1}$. The root of $\mathrm{BT}(M)$ is $\lambda\vec{x}.\, y$, and the subtrees are $\mathrm{BT}(M_0), \ldots \mathrm{BT}(M_{m-1})$.

Böhm tree examples for $\mathsf{S} \equiv \lambda abc.\, ac(bc)$, $\mathsf{S}a\Omega = \lambda c.\, ac\Omega$

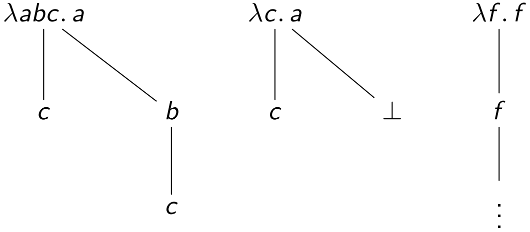


[3]Note that $\mathsf{Y}f =_\beta f(\mathsf{Y}f)$ and $\mathsf{Y} =_\beta \lambda f.\, f(\mathsf{Y}f)$.

Informal Böhm Trees (textbook definitions)

- If a term M is unsolvable, then its Böhm tree, denoted by $\mathrm{BT}(M)$, is $\perp$.
- Otherwise the term has principal hnf $\lambda\vec{x}.\,yM_0\cdots M_{m-1}$. The root of $\mathrm{BT}(M)$ is $\lambda\vec{x}.\,y$, and the subtrees are $\mathrm{BT}(M_0),\ldots\mathrm{BT}(M_{m-1})$.

Böhm tree examples for $\mathsf{S} \equiv \lambda abc.\,ac(bc)$, $\mathsf{S}a\Omega = \lambda c.\,ac\Omega$ and $\mathsf{Y} \equiv \lambda f.\,(\lambda x.\,f(xx))(\lambda x.\,f(xx))$.[3]

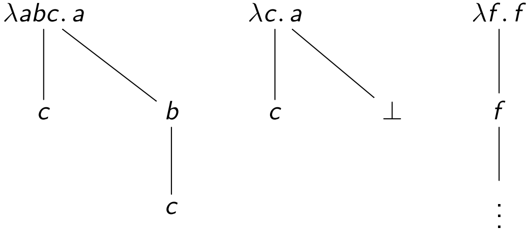


[3] Note that $\mathsf{Y}f =_\beta f(\mathsf{Y}f)$ and $\mathsf{Y} =_\beta \lambda f.\,f(\mathsf{Y}f)$.

Informal Böhm Trees (textbook definitions)

- If a term M is unsolvable, then its Böhm tree, denoted by $\mathrm{BT}(M)$, is $\perp$.
- Otherwise the term has principal hnf $\lambda\vec{x}.\, yM_0 \cdots M_{m-1}$. The root of $\mathrm{BT}(M)$ is $\lambda\vec{x}.\, y$, and the subtrees are $\mathrm{BT}(M_0), \ldots \mathrm{BT}(M_{m-1})$.

Böhm tree examples for $\mathsf{S} \equiv \lambda abc.\, ac(bc)$, $\mathsf{S}a\Omega = \lambda c.\, ac\Omega$ and $\mathsf{Y} \equiv \lambda f.\, (\lambda x.\, f(xx))(\lambda x.\, f(xx))$.[3]



By α-equivalence, different (informal) Böhm trees can be generated from the same term, by choosing different bound variables. (Barendregt suggests Böhm Trees should use de Bruijn indices.)

[3] Note that $\mathsf{Y}f =_\beta f(\mathsf{Y}f)$ and $\mathsf{Y} =_\beta \lambda f.\, f(\mathsf{Y}f)$.

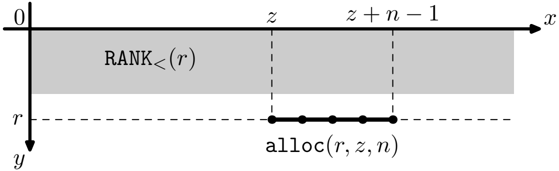
Rank-based fresh name allocation (1)

Given any finite set of names (as strings) X, it's easy to define "`NEWS` n X" which returns a list of n names excluding X (simply because the set of all strings is infinite.) But this is not enough.
The set of all strings is actually countably infinite, therefore can be filled into a 2-dimensional space, indexed by two natural numbers. To define Böhm tree formally, we need "`RNEWS` r n X", which returns n names at row r while excluding X. This funcion is based on `alloc`:

Definition (`alloc`)

The allocation function `alloc` allocates n names in the row r, starting at position (r, z). Thus the n allocated names are at coordinates $(r, z), (r, z + 1), \ldots, (r, z + n - 1)$:

$$\texttt{alloc}\ r\ z\ n \stackrel{\text{def}}{=} \texttt{GENLIST}\ (\lambda\, i.\ \texttt{n2s}\ (r \otimes (z + i)))\ n$$



Rank-based fresh name allocation (2)

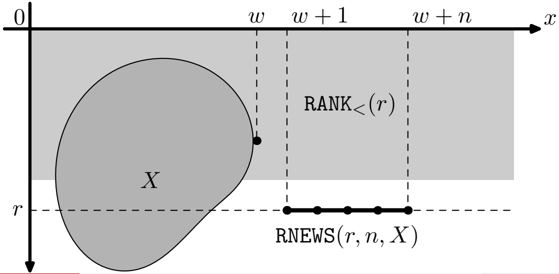
Definition (RNEWS and $\mathtt{RANK}_<$)

$$\mathtt{RNEWS}\ r\ n\ X \stackrel{\text{def}}{=} (\mathtt{let}\ z = \mathtt{SUC}\ (\mathtt{string\_width}\ X)\ \mathtt{in}\ \mathtt{alloc}\ r\ z\ n)$$

$$\mathtt{string\_width}\ X \stackrel{\text{def}}{=} \mathtt{MAX\_SET}\ (\mathtt{IMAGE}\ (\mathtt{nsnd} \circ \mathtt{s2n})\ X)$$

$\mathtt{RANK}_<\ r$ is the set of all names whose row is smaller than r:

$$\mathtt{RANK}_<\ r \stackrel{\text{def}}{=} \{v \mid \exists i\ j.\ v = \mathtt{n2s}\ (i \otimes j) \land i < r\}$$



Böhm Trees: The Formal definition

```
BT_generator X (M,r) ≝
  if solvable M then
    (let
       M₀ = principal_hnf M;
       n = LAMl_size M₀;
       vs = RNEWS r n X;
       M₁ = principal_hnf (M₀ ·· MAP VAR vs);
       Ms = hnf_children M₁;
       y = hnf_headvar M₁;
       l = MAP (λe. (e,SUC r)) Ms
     in
       (SOME (vs,y),fromList l))
  else (NONE,[||])

BT X ≝ ltree_unfold (BT_generator X)
```

The type of generated Böhm tree is "`BT_node ltree`", where `BT_node` is `(string list × string) option`.

Steps for generating one Böhm node (when M is solvable):

1. M_0 is the principal hnf of M, in form (*alpha*-equivalent to) `LAMl` vs `(VAR` y `··` Ms`)`;

Böhm Trees: The Formal definition

```
BT_generator X (M,r) ≝
  if solvable M then
    (let
       M₀ = principal_hnf M;
       n = LAMl_size M₀;
       vs = RNEWS r n X;
       M₁ = principal_hnf (M₀ ·· MAP VAR vs);
       Ms = hnf_children M₁;
       y = hnf_headvar M₁;
       l = MAP (λe. (e,SUC r)) Ms
     in
       (SOME (vs,y),fromList l))
  else (NONE,[||])

BT X ≝ ltree_unfold (BT_generator X)
```

The type of generated Böhm tree is "`BT_node ltree`", where `BT_node` is `(string list` $\times$ `string) option`.

Steps for generating one Böhm node (when M is solvable):

1. M_0 is the principal hnf of M, in form (*alpha*-equivalent to) `LAMl` vs `(VAR` y `··` Ms`)`;
2. n is the length of vs;

Böhm Trees: The Formal definition

```
BT_generator X (M,r) ≝
  if solvable M then
    (let
       M₀ = principal_hnf M;
       n = LAMl_size M₀;
       vs = RNEWS r n X;
       M₁ = principal_hnf (M₀ ·· MAP VAR vs);
       Ms = hnf_children M₁;
       y = hnf_headvar M₁;
       l = MAP (λe. (e,SUC r)) Ms
     in
       (SOME (vs,y),fromList l))
  else (NONE,[||])

BT X ≝ ltree_unfold (BT_generator X)
```

The type of generated Böhm tree is "`BT_node ltree`", where `BT_node` is `(string list × string) option`.

Steps for generating one Böhm node (when M is solvable):

1. M_0 is the principal hnf of M, in form (*alpha*-equivalent to) `LAMl vs (VAR y ·· Ms)`;
2. n is the length of vs;
3. vs is allocated at row r, excluding X;

Böhm Trees: The Formal definition

```
BT_generator X (M,r) ≝
  if solvable M then
    (let
       M₀ = principal_hnf M;
       n = LAMl_size M₀;
       vs = RNEWS r n X;
       M₁ = principal_hnf (M₀ ·· MAP VAR vs);
       Ms = hnf_children M₁;
       y = hnf_headvar M₁;
       l = MAP (λe. (e,SUC r)) Ms
     in
       (SOME (vs,y),fromList l))
  else (NONE,[||])

BT X ≝ ltree_unfold (BT_generator X)
```

The type of generated Böhm tree is "BT_node ltree", where BT_node is (string list × string) option.

Steps for generating one Böhm node (when M is solvable):

1. M_0 is the principal hnf of M, in form (*alpha*-equivalent to) LAMl vs (VAR y ·· Ms);
2. n is the length of vs;
3. vs is allocated at row r, excluding X;
4. M_1 is in form of VAR y ·· Ms;

Böhm Trees: The Formal definition

```
BT_generator X (M,r) ≝
  if solvable M then
    (let
       M₀ = principal_hnf M;
       n = LAMl_size M₀;
       vs = RNEWS r n X;
       M₁ = principal_hnf (M₀ ·· MAP VAR vs);
       Ms = hnf_children M₁;
       y = hnf_headvar M₁;
       l = MAP (λe. (e,SUC r)) Ms
     in
       (SOME (vs,y),fromList l))
  else (NONE,[||])

BT X ≝ ltree_unfold (BT_generator X)
```

The type of generated Böhm tree is "`BT_node ltree`", where `BT_node` is `(string list × string) option`.

Steps for generating one Böhm node (when M is solvable):

1. M_0 is the principal hnf of M, in form (*alpha*-equivalent to) `LAMl` vs (`VAR` y ·· Ms);
2. n is the length of vs;
3. vs is allocated at row r, excluding X;
4. M_1 is in form of `VAR` y ·· Ms;
5. Ms is finally obtained from M_1;

Böhm Trees: The Formal definition

```
BT_generator X (M,r) ≝
  if solvable M then
    (let
       M₀ = principal_hnf M;
       n = LAMl_size M₀;
       vs = RNEWS r n X;
       M₁ = principal_hnf (M₀ ·· MAP VAR vs);
       Ms = hnf_children M₁;
       y = hnf_headvar M₁;
       l = MAP (λe. (e,SUC r)) Ms
     in
       (SOME (vs,y),fromList l))
  else (NONE,[||])

BT X ≝ ltree_unfold (BT_generator X)
```

The type of generated Böhm tree is "`BT_node ltree`", where `BT_node` is `(string list × string) option`.

Steps for generating one Böhm node (when M is solvable):

1. M_0 is the principal hnf of M, in form (*alpha*-equivalent to) `LAMl` vs `(VAR` y `··` Ms`)`;
2. n is the length of vs;
3. vs is allocated at row r, excluding X;
4. M_1 is in form of `VAR` y `··` Ms;
5. Ms is finally obtained from M_1;
6. y is finally obtained from M_1;

Böhm Trees: The Formal definition

```
BT_generator X (M,r) ≝
  if solvable M then
    (let
       M₀ = principal_hnf M;
       n = LAMl_size M₀;
       vs = RNEWS r n X;
       M₁ = principal_hnf (M₀ ·· MAP VAR vs);
       Ms = hnf_children M₁;
       y = hnf_headvar M₁;
       l = MAP (λe. (e,SUC r)) Ms
     in
       (SOME (vs,y),fromList l))
  else (NONE,[||])

BT X ≝ ltree_unfold (BT_generator X)
```

The type of generated Böhm tree is "`BT_node ltree`", where `BT_node` is `(string list × string) option`.

Steps for generating one Böhm node (when M is solvable):

1. M_0 is the principal hnf of M, in form (*alpha*-equivalent to) `LAMl` vs (`VAR` y ·· Ms);
2. n is the length of vs;
3. vs is allocated at row r, excluding X;
4. M_1 is in form of `VAR` y ·· Ms;
5. Ms is finally obtained from M_1;
6. y is finally obtained from M_1;
7. l is the list of hnf children as seeds for generating subtrees, each paired with $r+1$;

Böhm Trees: The Formal definition

```
BT_generator X (M,r) ≝
  if solvable M then
    (let
       M₀ = principal_hnf M;
       n = LAMl_size M₀;
       vs = RNEWS r n X;
       M₁ = principal_hnf (M₀ ·· MAP VAR vs);
       Ms = hnf_children M₁;
       y = hnf_headvar M₁;
       l = MAP (λe. (e,SUC r)) Ms
     in
       (SOME (vs,y),fromList l))
  else (NONE,[||])

BT X ≝ ltree_unfold (BT_generator X)
```

The type of generated Böhm tree is "BT_node ltree", where BT_node is (string list × string) option.

Steps for generating one Böhm node (when M is solvable):

1. M_0 is the principal hnf of M, in form (*alpha*-equivalent to) LAMl vs (VAR y ·· Ms);
2. n is the length of vs;
3. vs is allocated at row r, excluding X;
4. M_1 is in form of VAR y ·· Ms;
5. Ms is finally obtained from M_1;
6. y is finally obtained from M_1;
7. l is the list of hnf children as seeds for generating subtrees, each paired with $r + 1$;
8. Instead of LAMl vs y, the tree node is SOME (vs,y) (or NONE for $\perp$).

Subterm (dual concept of Böhm tree)

```
subterm X M [] r ≝ SOME (M,r)
subterm X M (i::is) r ≝
  if solvable M then
    (let
       M₀ = principal_hnf M;
       n = LAMl_size M₀;
       vs = RNEWS r n X;
       M₁ = principal_hnf (M₀ ·· MAP VAR vs);
       Ms = hnf_children M₁;
       m = LENGTH Ms
     in
       if i < m then subterm X (EL i Ms) is (SUC r) else NONE)
  else NONE
```

If M is already in hnf, say `LAMl` vs $(y \cdot\cdot Ms)$, then "`subterm` X M $[i]$ r" essentially returns the i-th child of Ms, i.e. `EL` i Ms, paired with $r+1$.

Subterm (dual concept of Böhm tree)

```
subterm X M [] r ≝ SOME (M,r)
subterm X M (i::is) r ≝
  if solvable M then
    (let
       M₀ = principal_hnf M;
       n = LAMl_size M₀;
       vs = RNEWS r n X;
       M₁ = principal_hnf (M₀ ·· MAP VAR vs);
       Ms = hnf_children M₁;
       m = LENGTH Ms
     in
       if i < m then subterm X (EL i Ms) is (SUC r) else NONE)
  else NONE
```

If M is already in hnf, say `LAMl` vs $(y \cdot\cdot Ms)$, then "`subterm` X M $[i]$ r" essentially returns the i-th child of Ms, i.e. `EL` i Ms, paired with $r+1$.

Connection between Böhm trees and subterms (Barendregt 10.1.15):

$$
\begin{aligned}
\vdash\ & \mathtt{FINITE}\ X \land \mathrm{FV}\ M \subseteq X \cup \mathtt{RANK}_{<}\ r \land \mathtt{subterm}\ X\ M\ is\ r \neq \mathtt{NONE} \Rightarrow \\
& \mathtt{ltree\_lookup}\ (\mathtt{BT'}\ X\ M\ r)\ is \neq \mathtt{NONE} \land \\
& \mathtt{BT}\ X\ (\mathtt{THE}\ (\mathtt{subterm}\ X\ M\ is\ r)) = \mathtt{THE}\ (\mathtt{ltree\_lookup}\ (\mathtt{BT'}\ X\ M\ r)\ is)
\end{aligned}
$$

Basic properties of Böhm trees

- Two β-equivalent terms must have literally the same Böhm tree (Barendregt 10.1.6):

  $\vdash$ $\texttt{FINITE}\ X \land \mathrm{FV}\ M \subseteq X \cup \texttt{RANK}_{<}\ r \land \mathrm{FV}\ N \subseteq X \cup \texttt{RANK}_{<}\ r \land M =_\beta N \Rightarrow$
  $\texttt{BT'}\ X\ M\ r = \texttt{BT'}\ X\ N\ r$

Basic properties of Böhm trees

- Two β-equivalent terms must have literally the same Böhm tree (Barendregt 10.1.6):

  $\vdash \texttt{FINITE}\ X \land \mathrm{FV}\ M \subseteq X \cup \texttt{RANK}_{<}\ r \land \mathrm{FV}\ N \subseteq X \cup \texttt{RANK}_{<}\ r \land M =_{\beta} N \Rightarrow$
  $\texttt{BT'}\ X\ M\ r = \texttt{BT'}\ X\ N\ r$

- If a term is in β-normal form (bnf), then its Böhm tree must be finite, and without any $\perp$s:

  $\vdash \texttt{FINITE}\ X \land \mathrm{FV}\ M \subseteq X \cup \texttt{RANK}_{<}\ r \land \texttt{bnf}\ M \Rightarrow \texttt{ltree\_finite}\ (\texttt{BT'}\ X\ M\ r)$

  $\vdash \texttt{FINITE}\ X \land \mathrm{FV}\ M \subseteq X \cup \texttt{RANK}_{<}\ r \land \texttt{bnf}\ M \land p \in \texttt{ltree\_paths}\ (\texttt{BT'}\ X\ M\ r) \Rightarrow$
  $\texttt{ltree\_el}\ (\texttt{BT'}\ X\ M\ r)\ p \neq \texttt{SOME}\ \perp$

Basic properties of Böhm trees

- Two β-equivalent terms must have literally the same Böhm tree (Barendregt 10.1.6):

  $\vdash \texttt{FINITE}\ X \wedge \mathrm{FV}\ M \subseteq X \cup \texttt{RANK}_{<}\ r \wedge \mathrm{FV}\ N \subseteq X \cup \texttt{RANK}_{<}\ r \wedge M =_{\beta} N \Rightarrow$
  $\texttt{BT'}\ X\ M\ r = \texttt{BT'}\ X\ N\ r$

- If a term is in β-normal form (bnf), then its Böhm tree must be finite, and without any $\bot$s:

  $\vdash \texttt{FINITE}\ X \wedge \mathrm{FV}\ M \subseteq X \cup \texttt{RANK}_{<}\ r \wedge \texttt{bnf}\ M \Rightarrow \texttt{ltree\_finite}\ (\texttt{BT'}\ X\ M\ r)$

  $\vdash \texttt{FINITE}\ X \wedge \mathrm{FV}\ M \subseteq X \cup \texttt{RANK}_{<}\ r \wedge \texttt{bnf}\ M \wedge p \in \texttt{ltree\_paths}\ (\texttt{BT'}\ X\ M\ r) \Rightarrow$
  $\texttt{ltree\_el}\ (\texttt{BT'}\ X\ M\ r)\ p \neq \texttt{SOME}\ \bot$

- Free names of hnf children (in subterms) have the same "shape" as the parent term: (this is essential for (co)induction proofs of Böhm tree properties)

  $\vdash \texttt{FINITE}\ X \wedge \mathrm{FV}\ M \subseteq X \cup \texttt{RANK}_{<}\ r \wedge \texttt{solvable}\ M \wedge M_0 = \texttt{principal\_hnf}\ M \wedge$
  $n = \texttt{LAMl\_size}\ M_0 \wedge m = \texttt{hnf\_children\_size}\ M_0 \wedge vs = \texttt{RNEWS}\ r\ n\ X \wedge$
  $M_1 = \texttt{principal\_hnf}\ (M_0 \cdot\cdot \texttt{MAP VAR}\ vs) \wedge Ms = \texttt{hnf\_children}\ M_1 \wedge h < m \Rightarrow$
  $\mathrm{FV}\ (\texttt{EL}\ h\ Ms) \subseteq X \cup \texttt{RANK}_{<}\ (\texttt{SUC}\ r)$

Böhm Transformations and the "Böhm out" technique

Atomic transforms add a variable to the right, or perform a substitution.
A transformation is a list of these composed together:

$$\texttt{solving\_transform}\ f \stackrel{\mathsf{def}}{=} (\exists x.\ f = (\lambda p.\ p \bullet \texttt{VAR}\ x)) \lor \exists x\ N.\ f = [N/x]$$

$$\texttt{Boehm\_transform}\ \pi \stackrel{\mathsf{def}}{=} \texttt{EVERY}\ \texttt{solving\_transform}\ \pi$$

$$\texttt{apply}\ \pi \stackrel{\mathsf{def}}{=} \texttt{FOLDR}\ (\circ)\ \texttt{I}\ \pi$$

Böhm Transformations and the "Böhm out" technique

Atomic transforms add a variable to the right, or perform a substitution.
A transformation is a list of these composed together:

$$\texttt{solving\_transform}\ f \stackrel{\text{def}}{=} (\exists x.\ f = (\lambda p.\ p \bullet \texttt{VAR}\ x)) \lor \exists x\ N.\ f = [N/x]$$

$$\texttt{Boehm\_transform}\ \pi \stackrel{\text{def}}{=} \texttt{EVERY}\ \texttt{solving\_transform}\ \pi$$

$$\texttt{apply}\ \pi \stackrel{\text{def}}{=} \texttt{FOLDR}\ (\circ)\ \texttt{I}\ \pi$$

Transformed equal terms remain equal:

$$\vdash \texttt{Boehm\_transform}\ \pi \land M =_\beta N \Rightarrow \texttt{apply}\ \pi\ M =_\beta \texttt{apply}\ \pi\ N$$

"Böhm out" technique for single term (Barendregt 10.3.6)

A "ready" term is one that head-reduces to a term with y free at the head of an application, and where y doesn't appear elsewhere. (A prime target for a substitution IOW.)

$$\begin{array}{l} \texttt{is\_ready}\ M \stackrel{\text{def}}{=} \\ \quad \texttt{unsolvable}\ M \lor \exists y\ Ns.\ M \twoheadrightarrow_h \texttt{VAR}\ y \cdot\cdot\ Ns \land \texttt{EVERY}\ (\lambda e.\ y \sharp e)\ Ns \\ \texttt{is\_ready'}\ M \stackrel{\text{def}}{=} \texttt{is\_ready}\ M \land \texttt{solvable}\ M \end{array}$$

We can construct a Böhm transformation that

- makes the result is_ready'; and
- given a path p, guarantees that original and transformed subterms along p differ only by one closed substitution.

$$\begin{array}{l} \vdash \texttt{FINITE}\ X \land \mathrm{FV}\ M \subseteq X \cup \texttt{RANK}_{<}\ r \land p \neq [] \land \texttt{subterm}\ X\ M\ p\ r \neq \texttt{NONE} \Rightarrow \\ \quad \exists \pi.\ \texttt{Boehm\_transform}\ \pi \land \texttt{is\_ready'}\ (\texttt{apply}\ \pi\ M) \land \mathrm{FV}\ (\texttt{apply}\ \pi\ M) \subseteq X \cup \texttt{RANK}_{<}\ (\texttt{SUC}\ r) \land \\ \qquad \exists v\ P. \\ \qquad\quad \texttt{closed}\ P \land \\ \qquad\quad \forall q.\ q \preccurlyeq p \land q \neq [] \Rightarrow \\ \qquad\qquad \texttt{subterm}\ X\ (\texttt{apply}\ \pi\ M)\ q\ r \neq \texttt{NONE} \land \\ \qquad\qquad \texttt{subterm'}\ X\ (\texttt{apply}\ \pi\ M)\ q\ r = [P/v]\ (\texttt{subterm'}\ X\ M\ q\ r) \end{array}$$

"Böhm out" technique for multiple λ-terms (1)

Two Böhm tree nodes (either from the same or different trees) are head equivalent if:

- They are either both $\bot$ or both not $\bot$;
- In case they are not $\bot$, the two tree nodes can be written as $\langle \lambda x_1 \cdots x_n . \, y, m \rangle$ and $\langle \lambda x_1 \cdots x_{n'} . \, y', m' \rangle$, with m and m' the number of children at that node, and it is required that $y \equiv y'$ and $n - m = n' - m'$.

"Böhm out" technique for multiple λ-terms (1)

Two Böhm tree nodes (either from the same or different trees) are head equivalent if:

- They are either both $\bot$ or both not $\bot$;
- In case they are not $\bot$, the two tree nodes can be written as $\langle \lambda x_1 \cdots x_n.\, y, m \rangle$ and $\langle \lambda x_1 \cdots x_{n'}.\, y', m' \rangle$, with m and m' the number of children at that node, and it is required that $y \equiv y'$ and $n - m = n' - m'$.

This is a very local notion; only comparing superficial structure:

- the head variables must be equal; and
- extra variables in the binding list correspond to extra children (smells of η!)

"Böhm out" technique for multiple λ-terms (1)

Two Böhm tree nodes (either from the same or different trees) are head equivalent if:

- They are either both $\bot$ or both not $\bot$;
- In case they are not $\bot$, the two tree nodes can be written as $\langle \lambda x_1 \cdots x_n.\, y, m \rangle$ and $\langle \lambda x_1 \cdots x_{n'}.\, y', m' \rangle$, with m and m' the number of children at that node, and it is required that $y \equiv y'$ and $n - m = n' - m'$.

This is a very local notion; only comparing superficial structure:

- the head variables must be equal; and
- extra variables in the binding list correspond to extra children (smells of η!)

Extend this to equivalence at a particular path p:

$$\texttt{subtree\_equiv}\ X\ \ M\ \ N\ \ p\ \ r \stackrel{\text{def}}{=}$$
$$\texttt{ltree\_equiv}\ (\texttt{ltree\_el}\ (\texttt{BT'}\ X\ \ M\ \ r)\ \ p)\ (\texttt{ltree\_el}\ (\texttt{BT'}\ X\ \ N\ \ r)\ \ p)$$

and to terms that have hnfs (`equivalent`).

"Böhm out" technique for multiple λ-terms (2)

Let $\mathcal{F}$ be finite non-empty set of λ-terms, $\forall M \in \mathcal{F}.\, \alpha \in \mathrm{BT}(M)$ (shared path of all involved Böhm trees). Then there exists a Böhm transformation π such that:

- $\forall M \in \mathcal{F}.\, M^\pi$ is ready (and solvable);
- $\forall M \in \mathcal{F}.\, \alpha \in \mathrm{BT}(M^\pi)$;
- $\forall M \in \mathcal{F}, \beta \preceq \alpha.$
  M_β solvable iff M^π_β solvable;
- $\forall M, N \in \mathcal{F}, \beta \preceq \alpha.$
  $M \sim_\beta N$ iff $M^\pi \sim_\beta N^\pi$.

"Böhm out" technique for multiple λ-terms (2)

Let $\mathcal{F}$ be finite non-empty set of λ-terms, $\forall M \in \mathcal{F}.\, \alpha \in \mathrm{BT}(M)$ (shared path of all involved Böhm trees). Then there exists a Böhm transformation π such that:

- $\forall M \in \mathcal{F}.\, M^\pi$ is ready (and solvable);
- $\forall M \in \mathcal{F}.\, \alpha \in \mathrm{BT}(M^\pi)$;
- $\forall M \in \mathcal{F}, \beta \preceq \alpha.$ M_β solvable iff M^π_β solvable;
- $\forall M, N \in \mathcal{F}, \beta \preceq \alpha.$ $M \sim_\beta N$ iff $M^\pi \sim_\beta N^\pi$.

$$
\begin{array}{l}
\vdash \texttt{FINITE}\ X \wedge p \neq [] \wedge 0 < r \wedge Ms \neq [] \wedge \\
\quad \bigcup\ (\texttt{IMAGE}\ \mathrm{FV}\ (\texttt{set}\ Ms)) \subseteq X \cup \texttt{RANK}_<\ r \wedge \\
\quad \texttt{EVERY}\ (\lambda M.\ p \in \texttt{ltree\_paths}\ (\texttt{BT'}\ X\ M\ r))\ Ms \Rightarrow \\
\quad \exists \pi.\ \texttt{Boehm\_transform}\ \pi \wedge \texttt{EVERY is\_ready'}\ (\texttt{apply}\ \pi\ Ms) \wedge \\
\qquad \texttt{EVERY}\ (\lambda M.\ \mathrm{FV}\ M \subseteq X \cup \texttt{RANK}_<\ r)\ (\texttt{apply}\ \pi\ Ms) \wedge \\
\qquad \texttt{EVERY}\ (\lambda M.\ p \in \texttt{ltree\_paths}\ (\texttt{BT'}\ X\ M\ r))\ (\texttt{apply}\ \pi\ Ms) \wedge \\
\qquad (\forall q\ M. \\
\qquad\quad \texttt{MEM}\ M\ Ms \wedge q \preccurlyeq p \Rightarrow \\
\qquad\quad (\texttt{solvable}\ (\texttt{subterm'}\ X\ M\ q\ r) \iff \\
\qquad\quad\ \texttt{solvable}\ (\texttt{subterm'}\ X\ (\texttt{apply}\ \pi\ M)\ q\ r))) \wedge \\
\qquad \forall q\ M\ N. \\
\qquad\quad \texttt{MEM}\ M\ Ms \wedge \texttt{MEM}\ N\ Ms \wedge q \preccurlyeq p \Rightarrow \\
\qquad\quad (\texttt{subtree\_equiv}\ X\ M\ N\ q\ r \iff \\
\qquad\quad\ \texttt{subtree\_equiv}\ X\ (\texttt{apply}\ \pi\ M)\ (\texttt{apply}\ \pi\ N)\ q\ r)
\end{array}
$$

(The above lemma has a single big proof of 4,000+ lines.)

Separability of λ-terms (a restricted version)

Theorem

If M, N both have bnf (and also have the same set of Böhm tree paths) and $M \neq_\beta N$, then for any two terms P and Q there exists Böhm transformation π such that $M^\pi =_\beta P$ and $N^\pi =_\beta Q$:

$$\begin{array}{l} \vdash \texttt{FINITE}\ X \wedge \mathrm{FV}\ M \cup \mathrm{FV}\ N \subseteq X \cup \texttt{RANK}_<\ r \wedge 0 < r \wedge \\ \quad \texttt{ltree\_paths}\ (\texttt{BT'}\ X\ M\ r) = \texttt{ltree\_paths}\ (\texttt{BT'}\ X\ N\ r) \wedge \texttt{has\_bnf}\ M \wedge \\ \quad \texttt{has\_bnf}\ N \wedge M \neq_\beta N \Rightarrow \\ \quad \forall P\ Q.\ \exists \pi.\ \texttt{Boehm\_transform}\ \pi \wedge \texttt{apply}\ \pi\ M =_\beta P \wedge \texttt{apply}\ \pi\ N =_\beta Q \end{array}$$

If the antecedent $\texttt{ltree\_paths}\ (\texttt{BT'}\ X\ M\ r) = \texttt{ltree\_paths}\ (\texttt{BT'}\ X\ N\ r)$ gets removed, then the resulting theorem is Böhm's original separability theorem[4].

[4] C. Böhm. Alcune proprietà delle forme β-η-normali nel λ-k-calcolo. *Pubblicazioni dell'Instituto per le Applicazioni del Calcolo*, 696:1–19, 1968

η-separability and $\lambda\eta$-completeness

Theorem (η-separability)

Two distinct terms having $\beta\eta$-normal forms are η-separable:

$$\vdash \texttt{has\_benf}\ M \wedge \texttt{has\_benf}\ N \wedge M \neq_{\beta\eta} N \Rightarrow$$
$$\forall P\ Q.\ \exists\pi.\ \texttt{Boehm\_transform}\ \pi \wedge \texttt{apply}\ \pi\ M =_{\beta\eta} P \wedge \texttt{apply}\ \pi\ N =_{\beta\eta} Q$$

Proof of $\lambda\eta$-completess by η-separability.

Fix P and Q. Assuming $M \neq_{\beta\eta} N$, the goal is to prove $\lambda\eta + (M = N) \vdash P = Q$. By η-separability, there exists a Böhm transformation π such that $M^\pi =_{\beta\eta} P$ and $N^\pi =_{\beta\eta} Q$, therefore $\lambda\eta + (M = N) \vdash M^\pi = P$ and $\lambda\eta + (M = N) \vdash N^\pi = Q$. By induction on π, $\lambda\eta + (M = N) \vdash M^\pi = N^\pi$ can be proved (the base case is $M = N$, an axiom in $\lambda\eta + (M = N)$). Therefore $\lambda\eta + (M = N) \vdash P = Q$ by symmetry and transitivity of (all) λ-equational theories. $\square$

Removing Our Restriction

Given different terms M and N, need to deal with restriction
`ltree_paths (BT'` X M r`) = ltree_paths (BT'` X N r`)`

Thanks to $\boldsymbol{\eta}$, any M is equal to $\lambda x.\, M\, x$ with x fresh.

Removing Our Restriction

Given different terms M and N, need to deal with restriction
`ltree_paths (BT'` X M r`) = ltree_paths (BT'` X N r`)`

Thanks to $\boldsymbol{\eta}$, any M is equal to $\lambda x.\, M\, x$ with x fresh.

When M has a hnf, this gives the corresponding
Böhm tree extra paths to the right and/or downwards.

Removing Our Restriction

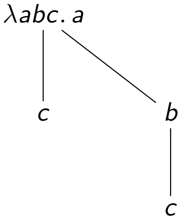
Given different terms M and N, need to deal with restriction
`ltree_paths (BT'` X M r`) = ltree_paths (BT'` X N r`)`

Thanks to $\boldsymbol{\eta}$, any M is equal to $\lambda x.\, M\,x$ with x fresh.

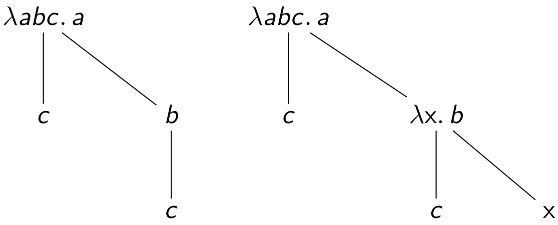
When M has a hnf, this gives the corresponding
Böhm tree extra paths to the right and/or downwards.

So, if finite `BT'` X M r and `BT'` X N r don't share
the same paths they can be η-converted so that they do!

From (restricted) separability to η-separability



From (restricted) separability to η-separability



From (restricted) separability to η-separability

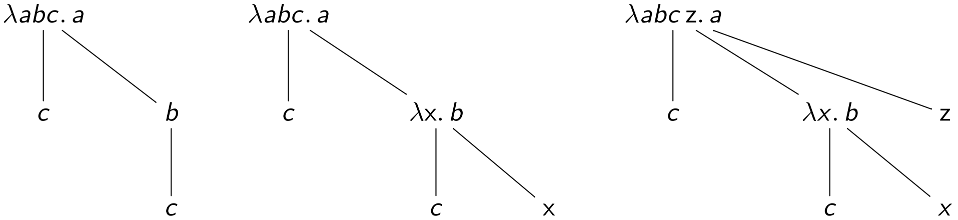


Figure: One-step η-expansion of (informal) Böhm trees.

Conclusion and future work

Done:

- Proved famous results; mostly following Barendregt
- Developed technology for working with Böhm trees

Not Done Yet:

- Other Böhm-like trees (Lévy-Longo trees, Berarducci trees, etc.) can be formalised similarly.
- Generally: denotational semantics for λ-calculus (D_∞, P_ω)
- Polishing the result:
  - inequal terms without $\beta\eta$-normal lead to inconsistency;
  - Böhm's original statement showing all terms can be separated
  - ...