Taylor Expansion, at work

Davide Barbarossa and **Giulio Manzonetto**

giulio.manzonetto@lipn.univ-paris13.fr

LIPN, Université Sorbonne Paris-Nord

January 28, 2022

The Global Picture

Program Approximation | Calculi | Denotational Semantics

λ-Calculus

The Global Picture

Program Approximation

Calculi

Denotational Semantics

λ-Calculus

Scott's Semantics

The Global Picture

Program Approximation | Calculi | Denotational Semantics

$$\begin{tikzcd} & \lambda\text{-Calculus} \arrow[dl, bend right] \arrow[dr, bend left] & \\ \text{Böhm Trees} & & \text{Scott's Semantics} \end{tikzcd}$$

The Global Picture

Program Approximation | Calculi | Denotational Semantics

λ-Calculus

Böhm Trees

Scott's Semantics

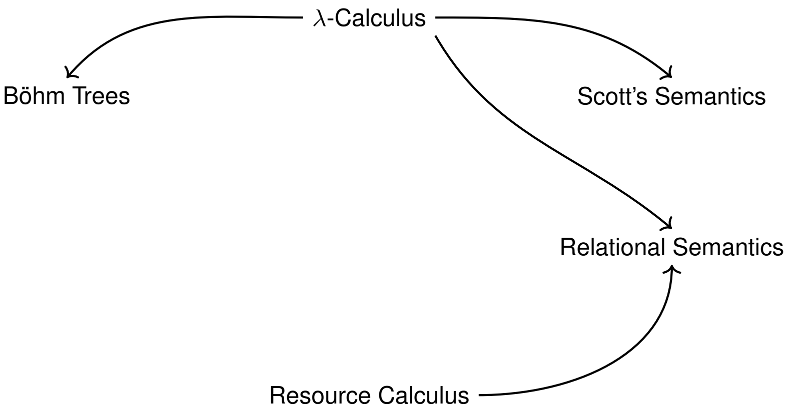
Relational Semantics

The Global Picture

Program Approximation

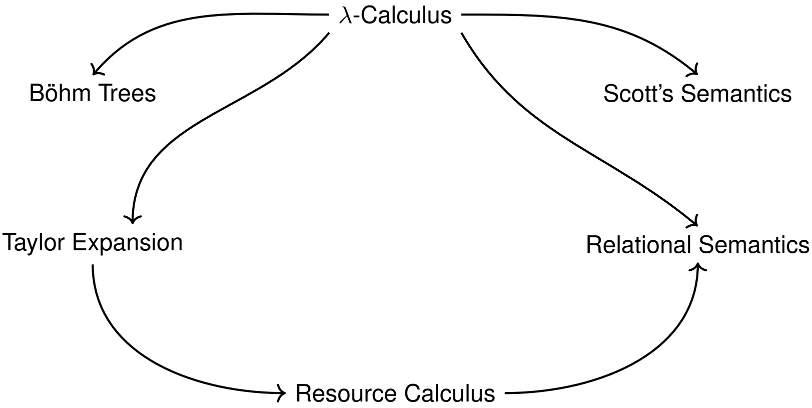
Calculi

Denotational Semantics



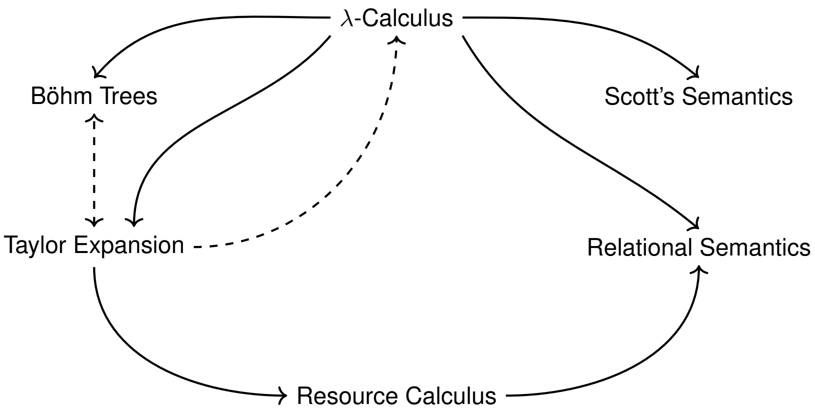
The Global Picture

Program Approximation | Calculi | Denotational Semantics



The Global Picture

Program Approximation | Calculi | Denotational Semantics



How to Handle the Complexity of Software?

An operating system can be huge, e.g. Linux is about 12 million lines of code.

```
let k = ref 1.5 let alpha = ref 0.0055 let edgenum = ref 1 let help = ref false let save = ref false let syn = ref false let sa = ref false let sources = ref
□ (* default values for linear logic edge detector *) let threshold = ref 0.01 let ndirs = ref 16 (* common default values *) let filename = ref "" module An
nealingArg = struct let specl = [ ("-help", Arg.Set help, " display this help message"); ("-k", Arg.Float (fun f -> k := f), " k value (default = 1.5)"); ("-a
lpha", Arg.Float (fun f -> alpha := f), " alpha value (default = 0.0055)"); ("-n", Arg.Int (fun f -> edgenum := f), " minimum number of edge in an edgel (defa
ult = 1)"); ("-s", Arg.Set save, " save edge map"); ("-syn", Arg.Set syn, " MFA's synchronous version"); ("-sa", Arg.Set sa, " Simulated Annealing algorithm"
) ] let usage = "Caml's eye\n" ^ "Usage: " ^ Sys.argv.(0) ^ " [options] file\n" ^ " file\t\t ps file containing edges\n\n" ^ "Options:" let get_par () = let a
non s = sources := !sources @ [s] in let _ = Arg.parse specl anon usage in let _ = if ( ( List.length !sources ) == 1) then let _ = filename := List.hd !sourc
es in let len = String.length !filename in if ( len > 3 ) && ( (String.sub !filename ( len - 3 ) 3 ) = ".ps" ) then filename := String.sub !filename 0 (len -
3) else invalid_arg ( "wrong " ^ !filename ^ " extension." ) else ( Arg.usage specl usage ; exit 0 ) in if ( !help ) then (* user needs help *) (Arg.usage spe
cl usage; exit 0) else () end module LogLinArg = struct let specl = [ ("-help", Arg.Set help, " display this help message"); ("-ndirs", Arg.Int ( fun n -> ndi
rs := n ), " number of directions (only even values) (default = 16)"); ("-t", Arg.Float ( fun f -> threshold := f ), " threshold value (default = 0.01)") ] le
t usage = "Linear Logic Edge Detector\n" ^ "Usage: " ^ Sys.argv.(0) ^ " [options] file\n" ^ " file\t\t (pgm/pbm/ppm) file containing ascii or binary image\n\n
" ^ "Options:" let get_par () = let anon s = sources := !sources @ [s] in let _ = Arg.parse specl anon usage in let _ = if ( ( List.length !sources ) == 1) th
en let _ = filename := List.hd !sources in let len = String.length !filename in if ( len > 4 ) && ( ( (String.sub !filename ( len - 4 ) 4 ) = ".pgm" ) || ( (S
tring.sub !filename ( len - 4 ) 4 ) = ".pbm" ) || ( (String.sub !filename ( len - 4 ) 4 ) = ".ppm" ) ) then () else invalid_arg ( "wrong " ^ !filename ^ " ext
ension." ) else ( Arg.usage specl usage ; exit 0 ) in if ( !help ) then (* user needs help *) (Arg.usage specl usage; exit 0) else () end module ParallelArg =
 struct let specl = [ ("-help", Arg.Set help, " display this help message"); ("-k", Arg.Float (fun f -> k := f), " k value (default = 1.5)"); ("-alpha", Arg.F
loat (fun f -> alpha := f), " alpha value (default = 0.0055)"); ("-n", Arg.Int (fun f -> edgenum := f), " minimum number of edge in an edgel (default = 1)");
          ("-p4pg", Arg.Unit (fun () -> ()), " set procgroup file" );             ("-p4wd", Arg.Unit (fun () -> ()), " set working directory\n" ) ] let usage
 = "Caml's eyes: a parallel version of camleye.\n" ^ "Usage: " ^ Sys.argv.(0) ^ " [options] file\n" ^ "Options:\n" ^ " file\t\t ps file containing edges" let
get_par () = let anon s = sources := !sources @ [s] in let _ = Arg.parse specl anon usage in let _ = if ( ( List.length !sources ) == 1) then let _ = filename
 := List.hd !sources in let len = String.length !filename in if ( len > 3 ) && ( (String.sub !filename ( len - 3 ) 3 ) = ".ps" ) then filename := String.sub !
```

How to Handle the Complexity of Software?

An operating system can be huge, e.g. Linux is about 12 million lines of code.

```
let k = ref 1.5 let alpha = ref 0.0055 let edgenum = ref 1 let help = ref false let save = ref false let syn = ref false let sa = ref false let sources = ref
□ (* default values for linear logic edge detector *) let threshold = ref 0.01 let ndirs = ref 16 (* common default values *) let filename = ref "" module An
nealingArg = struct let specl = [ ("-help", Arg.Set help, " display this help message"); ("-k", Arg.Float (fun f -> k := f), " k value (default = 1.5)"); ("-a
lpha", Arg.Float (fun f -> alpha := f), " alpha value (default = 0.0055)"); ("-n", Arg.Int (fun f -> edgenum := f), " minimum number of edge in an edgel (defa
ult = 1)"); ("-s", Arg.Set save, " save edge map"); ("-syn", Arg.Set syn, " MFA's synchronous version"); ("-sa", Arg.Set sa, " Simulated Annealing algorithm"
) ] let usage = "Caml's eye\n" ^ "Usage: " ^ Sys.argv.(0) ^ " [options] file\n" ^ " file\t\t ps file containing edges\n\n" ^ "Options:" let get_par () = let a
non s = sources := !sources @ [s] in let _ = Arg.parse specl anon usage in let _ = if ( ( List.length !sources ) == 1) then let _ = filename := List.hd !sourc
es in let len = String.length !filename in if ( len > 3 ) && ( (String.sub !filename ( len - 3 ) 3 ) = ".ps" ) then filename := String.sub !filename 0 (len -
3) else invalid_arg ( "wrong " ^ !filename ^ " extension." ) else ( Arg.usage specl usage ; exit 0 ) in if ( !help ) then (* user needs help *) (Arg.usage spe
cl usage; exit 0) else () end module LogLinArg = struct let specl = [ ("-help", Arg.Set help, " display this help message"); ("-ndirs", Arg.Int ( fun n -> ndi
rs := n ), " number of directions (only even values) (default = 16)"); ("-t", Arg.Float ( fun f -> threshold := f ), " threshold value (default = 0.01)") ] le
t usage = "Linear Logic Edge Detector\n" ^ "Usage: " ^ Sys.argv.(0) ^ " [options] file\n" ^ " file\t\t (pgm/pbm/ppm) file containing ascii or binary image\n\n
" ^ "Options:" let get_par () = let anon s = sources := !sources @ [s] in let _ = Arg.parse specl anon usage in let _ = if ( ( List.length !sources ) == 1) th
en let _ = filename := List.hd !sources in let len = String.length !filename in if ( len > 4 ) && ( ( (String.sub !filename ( len - 4 ) 4 ) = ".pgm" ) || ( (S
tring.sub !filename ( len - 4 ) 4 ) = ".pbm" ) || ( (String.sub !filename ( len - 4 ) 4 ) = ".ppm" ) ) then () else invalid_arg ( "wrong " ^ !filename ^ " ext
ension." ) else ( Arg.usage specl usage ; exit 0 ) in if ( !help ) then (* user needs help *) (Arg.usage specl usage; exit 0) else () end module ParallelArg =
 struct let specl = [ ("-help", Arg.Set help, " display this help message"); ("-k", Arg.Float (fun f -> k := f), " k value (default = 1.5)"); ("-alpha", Arg.F
loat (fun f -> alpha := f), " alpha value (default = 0.0055)"); ("-n", Arg.Int (fun f -> edgenum := f), " minimum number of edge in an edgel (default = 1)");
          ("-p4pg", Arg.Unit (fun () -> ()), " set procgroup file" );            ("-p4wd", Arg.Unit (fun () -> ()), " set working directory\n" ) ] let usage
 = "Caml's eyes: a parallel version of camleye.\n" ^ "Usage: " ^ Sys.argv.(0) ^ " [options] file\n" ^ "Options:\n" ^ " file\t\t ps file containing edges" let
get_par () = let anon s = sources := !sources @ [s] in let _ = Arg.parse specl anon usage in let _ = if ( ( List.length !sources ) == 1) then let _ = filename
 := List.hd !sources in let len = String.length !filename in if ( len > 3 ) && ( (String.sub !filename ( len - 3 ) 3 ) = ".ps" ) then filename := String.sub !
```

Denotational Semantics

1. Define a program interpretation satisfying compositionality.

The Theory of Program Approximation

1. Decompose a program into elementary “bricks” (its approximants),
2. Retrieve the whole program behaviour performing a “limit” of its approximants.

The Crucial Point — How to Handle Recursion?

Scott's continuity

"A finite portion of the output of a program must be generated by a finite portion of its input."

Kleene Fixed Point Theorem

Let $\mathcal{D} = (D, \leq, \perp)$ be a domain. Every Scott-continuous function

$$f : \mathcal{D} \to \mathcal{D}$$

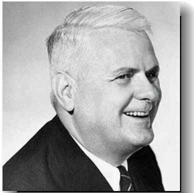
has a least fixed point $\mathrm{lfp}(f)$ that can calculated as follows:

$$\mathrm{lfp}(f) = \bigvee_{n \in \mathbb{N}} f^n(\perp)$$

Example. The factorial is the least fixed point of the higher-order program:

```
fun f → fun n → if n = 0 then 1 else n * (f (n - 1)))
```

The Theory of Program Approximation



First developed for untyped λ-calculus (Church, 1932)

Based on a primitive notion of function.

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

Computation becomes substitution $(\lambda x.M)N \to_\beta M[N/x]$.

Continous Semantics (Scott, 1969)

$\mathcal{D}_\infty$: First denotational model of λ-calculus.

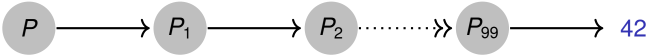
Böhm tree semantics (Barendregt, 1977)

Tree-like representation for program execution.

"Syntactic model" of λ-calculus.

Possible behaviours of a program

Classification	Behaviour	Result
normalizable	$P \to P_1 \to P_2 \twoheadrightarrow_{97} P_{99} \to 42$	completely defined



Possible behaviours of a program

Classification	Behaviour	Result
normalizable	$P \to P_1 \to P_2 \twoheadrightarrow_{97} P_{99} \to 42$	completely defined
unsolvable	$P \to P_1 \to P \twoheadrightarrow_{97} P_1 \to \cdots$	undefined

$$\begin{tikzcd} P \arrow[r, bend left] & P_1 \arrow[l, bend left] \end{tikzcd}$$

Possible behaviours of a program

Classification	Behaviour	Result
normalizable	$P \to P_1 \to P_2 \twoheadrightarrow_{97} P_{99} \to 42$	completely defined
unsolvable	$P \to P_1 \to P \twoheadrightarrow_{97} P_1 \to \cdots$	undefined
solvable	$P \to o_1 P_1 \to o_1(o_2 P_2) \to o_1(o_2(o_3 P_3))$ $\twoheadrightarrow_{\infty} o_1(o_2(o_3(\cdots o_n))\cdots)$	stable parts (infinitary)

$$P \longrightarrow P_1 \longrightarrow P_2 \longrightarrow P_3 \cdots\cdots\cdots \ggg \ \pi$$

$$3, \qquad 1 \qquad 4 \qquad 15926535\cdots$$

The Böhm Tree Semantics

Given a program P, its Böhm tree $\mathrm{BT}(P)$ is defined as follows:

- If P is completely undefined (unsolvable), then

$$\mathrm{BT}(P) = \bot,$$

- Otherwise $P \twoheadrightarrow \text{output}\ P_1 \cdots P_k$ and

$$\mathrm{BT}(P) = \begin{tikzcd}[column sep=small] & \text{output} \arrow[dl, dash] \arrow[dr, dash] & \\ \mathrm{BT}(P_1) & \cdots & \mathrm{BT}(P_k) \end{tikzcd}$$

Example

$$\begin{array}{c} \mathrm{BT}(\mathbf{Y}) \\ \shortparallel \\ \lambda f.f \\ | \\ f \\ | \\ f \\ | \\ f \\ | \\ \vdots \end{array}$$

The Böhm Tree Semantics

$$\mathcal{B} \vdash P = P' \iff \mathrm{BT}(P) = \mathrm{BT}(P')$$

The Böhm Tree Semantics

Given a program P, its Böhm tree $\mathrm{BT}(P)$ is defined as follows:

- If P is completely undefined (unsolvable), then

$$\mathrm{BT}(P) = \perp,$$

- Otherwise $P \twoheadrightarrow \text{output}\ P_1 \cdots P_k$ and

$$\mathrm{BT}(P) = \begin{array}{c} \text{output} \\ \diagup \qquad \diagdown \\ \mathrm{BT}(P_1) \quad \cdots \quad \mathrm{BT}(P_k) \end{array}$$

$$\mathrm{BT}(P) = \bigvee \mathcal{A}(P)$$

Example

$$\mathrm{BT}(\mathbf{Y}) = \perp$$

The Böhm Tree Semantics

$$\mathcal{B} \vdash P = P' \iff \mathrm{BT}(P) = \mathrm{BT}(P')$$

The Böhm Tree Semantics

Given a program P, its Böhm tree $\mathrm{BT}(P)$ is defined as follows:

- If P is completely undefined (unsolvable), then

$$\mathrm{BT}(P) = \bot,$$

- Otherwise $P \twoheadrightarrow \mathrm{output}\ P_1 \cdots P_k$ and

$$\mathrm{BT}(P) = \begin{array}{ccc} & \mathrm{output} & \\ \diagup & & \diagdown \\ \mathrm{BT}(P_1) & \cdots & \mathrm{BT}(P_k) \end{array}$$

$$\mathrm{BT}(P) = \bigvee \mathcal{A}(P)$$

Example

$$\begin{array}{c} \mathrm{BT}(\mathbf{Y}) \\ \shortparallel \\ \lambda f.f \\ | \\ \bot \end{array}$$

The Böhm Tree Semantics

$$\mathcal{B} \vdash P = P' \iff \mathrm{BT}(P) = \mathrm{BT}(P')$$

The Böhm Tree Semantics

Given a program P, its Böhm tree $\mathrm{BT}(P)$ is defined as follows:

- If P is completely undefined (unsolvable), then

$$\mathrm{BT}(P) = \bot,$$

- Otherwise $P \twoheadrightarrow \mathrm{output}\ P_1 \cdots P_k$ and

$$\mathrm{BT}(P) = \begin{array}{ccc} & \mathrm{output} & \\ \swarrow & & \searrow \\ \mathrm{BT}(P_1) & \cdots & \mathrm{BT}(P_k) \end{array}$$

$$\mathrm{BT}(P) = \bigvee \mathcal{A}(P)$$

Example

$$\begin{array}{c} \mathrm{BT}(\mathbf{Y}) \\ \parallel \\ \lambda f.f \\ | \\ f \\ | \\ \bot \end{array}$$

The Böhm Tree Semantics

$$\mathcal{B} \vdash P = P' \iff \mathrm{BT}(P) = \mathrm{BT}(P')$$

The Böhm Tree Semantics

Given a program P, its Böhm tree $\mathrm{BT}(P)$ is defined as follows:

- If P is completely undefined (unsolvable), then

$$\mathrm{BT}(P) = \bot,$$

- Otherwise $P \twoheadrightarrow \mathrm{output}\ P_1 \cdots P_k$ and

$$\mathrm{BT}(P) = \begin{array}{ccc} & \mathrm{output} & \\ \swarrow & & \searrow \\ \mathrm{BT}(P_1) & \cdots & \mathrm{BT}(P_k) \end{array}$$

$$\mathrm{BT}(P) = \bigvee \mathcal{A}(P)$$

Example

$$\begin{array}{c} \mathrm{BT}(\mathbf{Y}) \\ \shortparallel \\ \lambda f.f \\ | \\ f \\ | \\ f \\ | \\ \bot \end{array}$$

The Böhm Tree Semantics

$$\mathcal{B} \vdash P = P' \iff \mathrm{BT}(P) = \mathrm{BT}(P')$$

The Böhm Tree Semantics

Given a program P, its Böhm tree $\mathrm{BT}(P)$ is defined as follows:

- If P is completely undefined (unsolvable), then

$$\mathrm{BT}(P) = \bot,$$

- Otherwise $P \twoheadrightarrow \text{output } P_1 \cdots P_k$ and

$$\mathrm{BT}(P) = \begin{array}{ccc} & \text{output} & \\ \swarrow & & \searrow \\ \mathrm{BT}(P_1) & \cdots & \mathrm{BT}(P_k) \end{array}$$

$$\mathrm{BT}(P) = \bigvee \mathcal{A}(P)$$

Example

$$\begin{array}{c} \mathrm{BT}(\mathbf{Y}) \\ \shortparallel \\ \lambda f.f \\ | \\ f \\ | \\ f \\ | \\ f \\ | \\ \bot \end{array}$$

The Böhm Tree Semantics

$$\mathcal{B} \vdash P = P' \iff \mathrm{BT}(P) = \mathrm{BT}(P')$$

The Böhm Tree Semantics

Given a program P, its Böhm tree $\mathrm{BT}(P)$ is defined as follows:

- If P is completely undefined (unsolvable), then

$$\mathrm{BT}(P) = \bot,$$

- Otherwise $P \twoheadrightarrow \text{output } P_1 \cdots P_k$ and

$$\mathrm{BT}(P) = \begin{array}{ccc} & \text{output} & \\ \swarrow & & \searrow \\ \mathrm{BT}(P_1) & \cdots & \mathrm{BT}(P_k) \end{array}$$

$$\mathrm{BT}(P) = \bigvee \mathcal{A}(P)$$

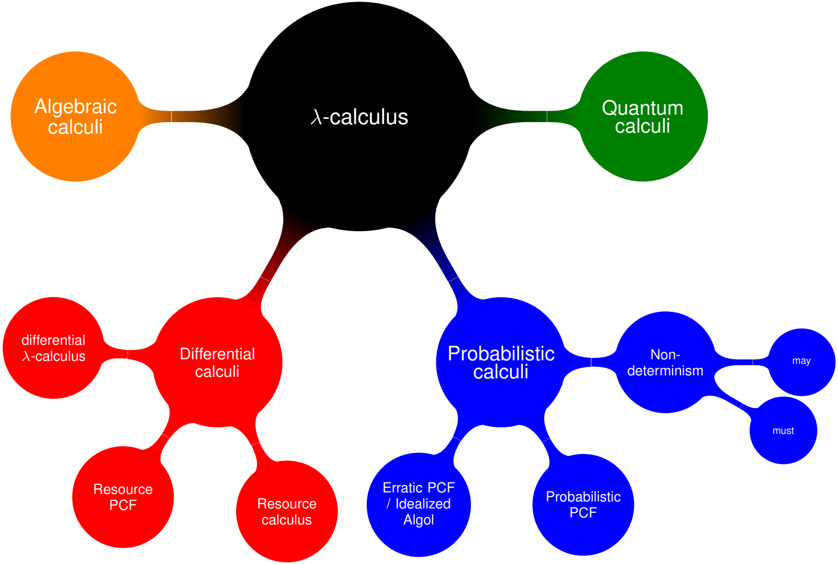
Example

$$\begin{array}{c} \mathrm{BT}(\mathbf{Y}) \\ \parallel \\ \lambda f.f \\ | \\ f \\ | \\ f \\ | \\ f \\ | \\ \vdots \end{array}$$

The Böhm Tree Semantics

$$\mathcal{B} \vdash P = P' \iff \mathrm{BT}(P) = \mathrm{BT}(P')$$

The Mainstream Programming Languages



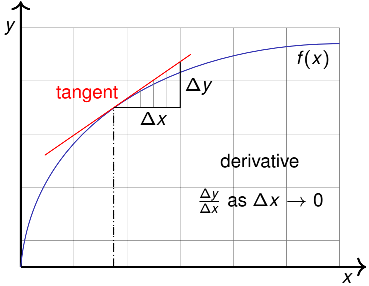
The origin of Linear Logic Quantitative Semantics

- Girard: Normal functors, power series and λ-calculus. Ann. Pure Appl. Log., 1988.
- Girard: Linear Logic. Theor. Comput. Sci. 50: 1-102 (1987)

As remarked in the LL paper: A notion of differentiation is at hand in some of these models. . .

The differential λ-calculus — Ehrhard & Regnier 2003

Mathematical Analysis

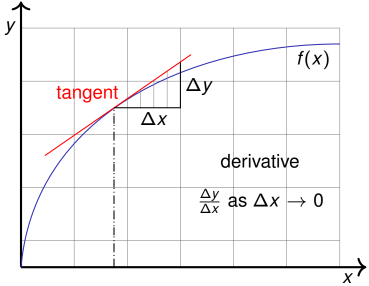


Taylor expansion

$$f(x) = \sum_{n=0}^{\infty} \frac{f^{(n)}(a)}{n!}(x-a)^n$$

The differential λ-calculus — Ehrhard & Regnier 2003

Mathematical Analysis



Taylor expansion

$$f(x) = \sum_{n=0}^{\infty} \frac{f^{(n)}(a)}{n!}(x-a)^n$$

Theory of Programming Languages

The differential λ-calculus

$$D(\lambda x.M) \cdot N \to$$

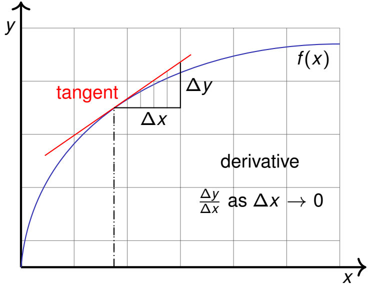
$$\lambda x.\left(\frac{\partial M}{\partial x} \cdot N\right)$$

linear substitution of N

for one occurrence of x in M

The differential λ-calculus — Ehrhard & Regnier 2003

Mathematical Analysis



Theory of Programming Languages

The differential λ-calculus

$$D(\lambda x.M) \cdot N \to$$

$$\lambda x.\left(\frac{\partial M}{\partial x} \cdot N\right)$$

linear substitution of N
for one occurrence of x in M

Taylor expansion

$$f(x) = \sum_{n=0}^{\infty} \frac{f^{(n)}(a)}{n!}(x-a)^n$$

Taylor expansion $\mathcal{T}(-)$

$$P\,x = \sum_{n=0}^{\infty} \frac{1}{n!}(\mathrm{D}^n(P) \cdot (x, \ldots, x))0$$

The ambitious goal: to replace the theory of program approximation based on continuity and Böhm trees with the theory of resource consumption based on Taylor expansion.

The Resource Calculus Λ^r

Resource approximants:

$$
\begin{array}{rcl}
t & ::= & x \mid \lambda x.t \mid t\,b \\
b & ::= & [t_1, \ldots, t_n] \qquad \text{where } n \geq 0
\end{array}
$$

The Resource Calculus Λ^r

Resource approximants:

$$\begin{array}{rcll} t & ::= & x \mid \lambda x.t \mid t\,b & \\ b & ::= & [t_1, \ldots, t_n] & \text{where } n \geq 0 \end{array}$$

Resource Terms

The Resource Calculus Λ^r

Resource approximants:

$$\begin{array}{rcl} t & ::= & x \mid \lambda x.t \mid t\, b \\ b & ::= & [t_1, \ldots, t_n] \quad \text{where } n \geq 0 \end{array}$$

Resource Terms

The Resource Calculus Λ^r

Resource approximants:

$$
\begin{array}{lcll}
t & ::= & x \mid \lambda x.t \mid t\,b & \\
b & ::= & [t_1, \ldots, t_n] & \text{where } n \geq 0
\end{array}
$$

Bags

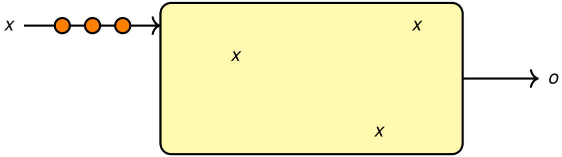
The Resource Calculus Λ^r

Resource approximants:

$$\begin{array}{rcll} t & ::= & x \mid \lambda x.t \mid t\, b & \\ b & ::= & [t_1, \ldots, t_n] & \text{where } n \geq 0 \end{array}$$

Reduction:

$$(\lambda x.t)[s_1, s_2, s_3] \to ?$$



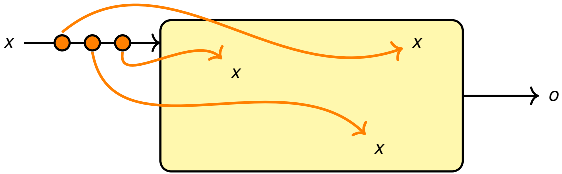
The Resource Calculus Λ^r

Resource approximants:

$$\begin{array}{rcll} t & ::= & x \mid \lambda x.t \mid t\,b & \\ b & ::= & [t_1, \ldots, t_n] & \text{where } n \geq 0 \end{array}$$

Reduction:

$$(\lambda x.t)[s_1, s_2, s_3] \to t\langle s_1/x_1, s_2/x_2, s_3/x_1\rangle$$



The Resource Calculus Λ^r

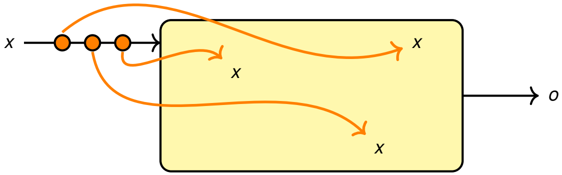
Resource approximants:

$$\begin{array}{rcll} t & ::= & x \mid \lambda x.t \mid t\, b & \\ b & ::= & [t_1, \dots, t_n] & \text{where } n \geq 0 \\ \mathbb{T} & ::= & t_1 + \cdots + t_n & \end{array}$$

Formal sums

Reduction:

$$(\lambda x.t)[s_1, s_2, s_3] \to \sum_{\sigma \in \mathfrak{S}_3} t\langle s_1/x_{\sigma(1)}, s_2/x_{\sigma(2)}, s_3/x_{\sigma(1)}\rangle$$



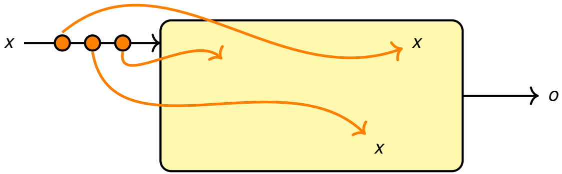
The Resource Calculus Λ^r

Resource approximants:

$$
\begin{array}{lcll}
t & ::= & x \mid \lambda x.t \mid t\,b & \\
b & ::= & [t_1, \ldots, t_n] & \text{where } n \geq 0 \\
\mathbb{T} & ::= & t_1 + \cdots + t_n &
\end{array}
$$

Reduction:

$$(\lambda x.t)[s_1, s_2, s_3] \to \ ?$$



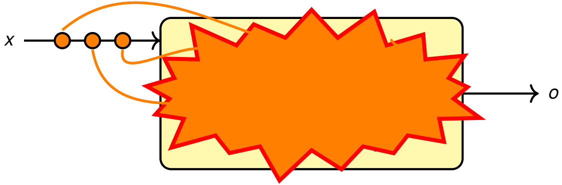
The Resource Calculus Λ^r

Resource approximants:

$$
\begin{array}{rcll}
t & ::= & x \mid \lambda x.t \mid t\,b & \\
b & ::= & [t_1,\ldots,t_n] & \text{where } n \geq 0 \\
\mathbb{T} & ::= & t_1 + \cdots + t_n &
\end{array}
$$

Reduction:

$$(\lambda x.t)[s_1, s_2, s_3] \to 0$$



All constructors are linear:

$$
\begin{array}{rcl}
\lambda x.\sum_i t_i & := & \sum_i \lambda x.t_i \\
(\sum_i t_i)(\sum_j b_j) & := & \sum_{i,j} t_i b_j \\
[\sum_i t_i,\ldots] & := & \sum_i [t_i,\ldots]
\end{array}
$$

The Resource Calculus Λ^r — Examples

Let $\mathrm{I} = \lambda x.x$ be the identity.

The linear fragment of λ-calculus is embeddable:

$$(\lambda fgh.f[g][h])[x][y][z] \to (\lambda gh.x[g][h])[y][z] \to (\lambda h.x[y][h])[z] \to x[y][z]$$

Resource terms may experience starvation:

$$(\lambda x.x[x])[\lambda x.x[x],\ \lambda x.x[x]] \to (\lambda x.x[x])[\lambda x.x[x]] \to 0$$

Resource terms may experience surfeit:

$$(\lambda fg.f)[x][y] \to (\lambda g.x)[y] \to 0$$

Non-determinism may arise along the reduction:

$$(\lambda f.f[f])[y,z] \to y[z] + z[y]$$

The Resource Calculus Λ^r — Examples

Let $\mathrm{I} = \lambda x.x$ be the identity.

The linear fragment of λ-calculus is embeddable:

$$(\lambda fgh.f[g][h])[x][y][z] \to (\lambda gh.x[g][h])[y][z] \to (\lambda h.x[y][h])[z] \to x[y][z]$$

Resource terms may experience starvation:

$$(\lambda x.x[x])[\lambda x.x[x],\ \lambda x.x[x]] \to (\lambda x.x[x])[\lambda x.x[x]] \to 0$$

Resource terms may experience surfeit:

$$(\lambda fg.f)[x][y] \to (\lambda g.x)[y] \to 0$$

Non-determinism may arise along the reduction:

$$(\lambda f.f[f])[y, z] \to y[z] + z[y]$$

The Resource Calculus Λ^r — Examples

Let $\mathrm{I} = \lambda x.x$ be the identity.

The linear fragment of λ-calculus is embeddable:

$$(\lambda fgh.f[g][h])[x][y][z] \to (\lambda gh.x[g][h])[y][z] \to (\lambda h.x[y][h])[z] \to x[y][z]$$

Resource terms may experience starvation:

$$(\lambda x.x[x])[\lambda x.x[x],\ \lambda x.x[x]] \to (\lambda x.x[x])[\lambda x.x[x]] \to 0$$

Resource terms may experience surfeit:

$$(\lambda fg.f)[x][y] \to (\lambda g.x)[y] \to 0$$

Non-determinism may arise along the reduction:

$$(\lambda f.f[f])[y,z] \to y[z] + z[y]$$

The Resource Calculus Λ^r — Examples

Let $\mathrm{I} = \lambda x.x$ be the identity.

The linear fragment of λ-calculus is embeddable:

$$(\lambda fgh.f[g][h])[x][y][z] \to (\lambda gh.x[g][h])[y][z] \to (\lambda h.x[y][h])[z] \to x[y][z]$$

Resource terms may experience starvation:

$$(\lambda x.x[x])[\lambda x.x[x],\ \lambda x.x[x]] \to (\lambda x.x[x])[\lambda x.x[x]] \to 0$$

Resource terms may experience surfeit:

$$(\lambda fg.f)[x][y] \to (\lambda g.x)[y] \to 0$$

Non-determinism may arise along the reduction:

$$(\lambda f.f[f])[y,z] \to y[z] + z[y]$$

The Resource Calculus Λ^r

Resource approximants:

$$\begin{array}{rcll} t & ::= & x \mid \lambda x.t \mid t\, b & \\ b & ::= & [t_1, \dots, t_n] & \text{where } n \geq 0 \\ \mathbb{T} & ::= & t_1 + \cdots + t_n & \end{array}$$

Reduction:

$(\lambda x.t)[s_1, \dots, s_n] \twoheadrightarrow \mathbb{T} \neq 0 \quad \Rightarrow \quad t$ must use each s_i exactly once in the reduction to a value.

$t \twoheadrightarrow c(\!|0|\!) = 0 \quad \Leftarrow \quad$ otherwise, the whole program t becomes an empty program 0.

Main Properties

- Strong Normalization: Trivial, because there is no duplication. □
- Confluence: Locally confluent + strongly normalizing. □
- Linearity: Nothing gets erased in a non-zero reduction sequence. □

Taylor Expansion : λ-terms $\quad\mapsto\quad$ (infinite) series of resource approximants

The Taylor Expansion of a λ-term:

$$MN \quad\mapsto\quad \sum_{k=0}^{\infty} \frac{1}{k!} M[\underbrace{N,\dots,N}_{k \text{ times}}] \qquad \Big(\cong \sum_{k=0}^{\infty} \frac{1}{k!} (\mathrm{D}^k(M)\cdot(N,\dots,N))0\Big)$$

Examples:

$$\begin{aligned}
\mathcal{T}(\lambda x.x) &= \{\lambda x.x\} \\
\mathcal{T}(\lambda x.xx) &= \{\lambda x.x[x^n] \mid n \in \mathbb{N}\} \\
\mathcal{T}(\Omega) &= \{(\lambda x.x[x^{n_0}])[\lambda x.x[x^{n_1}],\dots,\lambda x.x[x^{n_k}]] \mid k, n_0,\dots,n_k \in \mathbb{N}\} \\
\mathcal{T}(\Delta_f) &= \{\lambda x.f[x^n][x^k] \mid n,k \in \mathbb{N}\} \\
\mathcal{T}(\mathrm{Y}) &= \{\lambda f.t[s_1,\dots,s_k] \mid k \in \mathbb{N},\ t, s_1,\dots,s_k \in \mathcal{T}(\Delta_f)\}
\end{aligned}$$

where $\mathrm{Y} = \lambda f.\Delta_f\Delta_f$ and $\Delta_f = \lambda x.f(xx)$.

Taylor Expansion : λ-terms $\mapsto$ sets of resource approximants

The Taylor Expansion of a λ-term:

$$\mathcal{T}(x) = \{x\}, \qquad \mathcal{T}(\lambda x.M) = \{\lambda x.t \mid t \in \mathcal{T}(M)\},$$
$$\mathcal{T}(MN) = \bigcup_{k\in\mathbb{N}} \{t[s_1,\dots,s_k] \mid t \in \mathcal{T}(M), s_1,\dots,s_k \in \mathcal{T}(N)\}.$$

Examples:

$$\begin{aligned}
\mathcal{T}(\lambda x.x) &= \{\lambda x.x\}\\
\mathcal{T}(\lambda x.xx) &= \{\lambda x.x[x^n] \mid n \in \mathbb{N}\}\\
\mathcal{T}(\Omega) &= \{(\lambda x.x[x^{n_0}])[\lambda x.x[x^{n_1}],\dots,\lambda x.x[x^{n_k}]] \mid k, n_0,\dots,n_k \in \mathbb{N}\}\\
\mathcal{T}(\Delta_f) &= \{\lambda x.f[x^n][x^k] \mid n,k \in \mathbb{N}\}\\
\mathcal{T}(\mathrm{Y}) &= \{\lambda f.t[s_1,\dots,s_k] \mid k \in \mathbb{N},\ t,s_1,\dots,s_k \in \mathcal{T}(\Delta_f)\}
\end{aligned}$$

where $\mathrm{Y} = \lambda f.\Delta_f\Delta_f$ and $\Delta_f = \lambda x.f(xx)$.

Taylor Expansion : λ-terms $\mapsto$ sets of resource approximants

The Taylor Expansion of a λ-term:

$$\begin{array}{rclrcl} \mathcal{T}(x) & = & \{x\}, & \mathcal{T}(\lambda x.M) & = & \{\lambda x.t \mid t \in \mathcal{T}(M)\}, \\ \mathcal{T}(MN) & = & \multicolumn{4}{l}{\bigcup_{k\in\mathbb{N}} \{t[s_1,\dots,s_k] \mid t \in \mathcal{T}(M), s_1,\dots,s_k \in \mathcal{T}(N)\}.} \end{array}$$

Examples:

$$\begin{array}{rcl} \mathcal{T}(\lambda x.x) & = & \{\lambda x.x\} \\ \mathcal{T}(\lambda x.xx) & = & \{\lambda x.x[x^n] \mid n \in \mathbb{N}\} \\ \mathcal{T}(\Omega) & = & \{(\lambda x.x[x^{n_0}])[\lambda x.x[x^{n_1}],\dots,\lambda x.x[x^{n_k}]] \mid k, n_0,\dots,n_k \in \mathbb{N}\} \\ \mathcal{T}(\Delta_f) & = & \{\lambda x.f[x^n][x^k] \mid n,k \in \mathbb{N}\} \\ \mathcal{T}(\mathrm{Y}) & = & \{\lambda f.t[s_1,\dots,s_k] \mid k \in \mathbb{N},\ t, s_1,\dots,s_k \in \mathcal{T}(\Delta_f)\} \end{array}$$

where $\mathrm{Y} = \lambda f.\Delta_f\Delta_f$ and $\Delta_f = \lambda x.f(xx)$.

The Dynamics of Taylor Expansion

Computing the normal form:

$$\mathrm{NF}(\mathcal{T}(M)) = \bigcup \{\mathrm{nf}(t) \mid t \in \mathcal{T}(M)\}$$

Examples

- $\mathcal{T}(\lambda x.x)$, $\mathcal{T}(\lambda x.xx)$, $\mathcal{T}(\lambda x.f(xx))$ are already in normal form.
- $\mathrm{NF}(\mathcal{T}(\mathrm{Y})) = \{\lambda f.f[], \lambda f.f[f[]], \lambda f.f[f[], f[]]], \lambda f.f[f[f[]], f[f[f[]]], f[]], \dots\}$.
- $\mathrm{NF}(\mathcal{T}(\Omega)) = \emptyset$. This is the case for all unsolvables.

Taylor Expansion vs Böhm Trees

Advantages:

1. Approximants are closed under application.
2. Enjoy Strong Normalization & Confluence & Linearity.
3. Generalizable to the mainstream languages.

Disadvantage:

1. lots of indices arise from the linearization.

$$\begin{tikzcd}
\Lambda \arrow[r, "\mathcal{T}"] \arrow[d, "\mathrm{BT}"'] & \Lambda^r \arrow[d, "\mathrm{NF}"] \\
\mathcal{B} & \mathrm{NF}(\Lambda^r)
\end{tikzcd}$$

The Dynamics of Taylor Expansion

Computing the normal form:

$$\mathrm{NF}(\mathcal{T}(M)) = \bigcup \{\mathrm{nf}(t) \mid t \in \mathcal{T}(M)\}$$

Examples

- $\mathcal{T}(\lambda x.x)$, $\mathcal{T}(\lambda x.xx)$, $\mathcal{T}(\lambda x.f(xx))$ are already in normal form.
- $\mathrm{NF}(\mathcal{T}(\mathrm{Y})) = \{\lambda f.f[], \lambda f.f[f[]], \lambda f.f[f[], f[]], \lambda f.f[f[f[]], f[f[f[]]], f[]], \dots\}$.
- $\mathrm{NF}(\mathcal{T}(\Omega)) = \emptyset$. This is the case for all unsolvables.

Taylor Expansion vs Böhm Trees

Advantages:

1. Approximants are closed under application.
2. Enjoy Strong Normalization & Confluence & Linearity.
3. Generalizable to the mainstream languages.

Disadvantage:

1. lots of indices arise from the linearization.

$$\begin{tikzcd}
\Lambda \arrow[r, "\mathcal{T}"] \arrow[d, "\mathrm{BT}"'] & \Lambda^r \arrow[d, "\mathrm{NF}"] \\
\mathcal{B} & \mathrm{NF}(\Lambda^r)
\end{tikzcd}$$

The Dynamics of Taylor Expansion

Computing the normal form:

$$\mathrm{NF}(\mathcal{T}(M)) = \bigcup \{\mathrm{nf}(t) \mid t \in \mathcal{T}(M)\}$$

Examples

- $\mathcal{T}(\lambda x.x)$, $\mathcal{T}(\lambda x.xx)$, $\mathcal{T}(\lambda x.f(xx))$ are already in normal form.
- $\mathrm{NF}(\mathcal{T}(\mathrm{Y})) = \{\lambda f.f[], \lambda f.f[f[]], \lambda f.f[f[], f[]]], \lambda f.f[f[f[]], f[f[f[]]], f[]], \dots\}$.
- $\mathrm{NF}(\mathcal{T}(\Omega)) = \emptyset$. This is the case for all unsolvables.

Taylor Expansion vs Böhm Trees

Advantages:

1. Approximants are closed under application.
2. Enjoy Strong Normalization & Confluence & Linearity.
3. Generalizable to the mainstream languages.

Disadvantage:

1. lots of indices arise from the linearization.

Ehrhard & Regnier 2003

$$\begin{tikzcd} \Lambda \arrow[r, "\mathcal{T}"] \arrow[d, "\mathrm{BT}"'] & \Lambda^r \arrow[d, "\mathrm{NF}"] \\ \mathcal{B} \arrow[r, "\mathcal{T}"] & \mathrm{NF}(\Lambda^r) \end{tikzcd}$$

Approximation Theory

A common structure

1. Source language
2. Target language: resource calculus
   - confluence,
   - strong normalization.
3. Definition of Taylor Expansion
   - static analysis (coherence/cliques),
   - dynamic analysis (normalization).
4. Adequacy

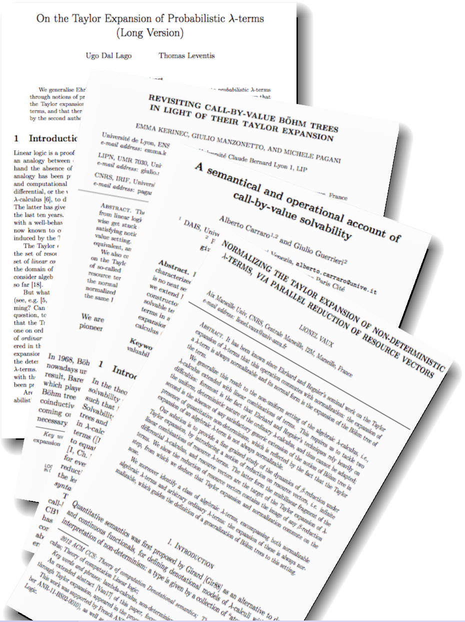
Commutation Theorem

$$\mathrm{NF}(\mathcal{T}(P)) = \mathcal{T}(\mathrm{BT}(P))$$

Corollary

Böhm trees and Taylor semantics coincide:

$$\mathrm{BT}(P) = \mathrm{BT}(P') \iff \mathrm{NF}(\mathcal{T}(P)) = \mathrm{NF}(\mathcal{T}(P'))$$



Approximation Theory

A common structure

1. Source language
2. Target language: resource calculus
   - confluence,
   - strong normalization.
3. Definition of Taylor Expansion
   - static analysis (coherence/cliques),
   - dynamic analysis (normalization).
4. Adequacy

Commutation Theorem

$$\mathrm{NF}(\mathcal{T}(P)) = \mathcal{T}(\mathrm{BT}(P))$$

Corollary

Böhm trees and Taylor semantics coincide:

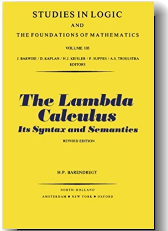
$$\mathrm{BT}(P) = \mathrm{BT}(P') \iff \mathrm{NF}(\mathcal{T}(P)) = \mathrm{NF}(\mathcal{T}(P'))$$

We explored the real power of the Taylor expansion!

Classic results

Scott's Continuity

Chapter 14



Berry's Stability

$\nexists$ parallel or

topological argument

Contextuality of BTs

Genericity Lemma

Perpendicular Lines Lemma

Khan & Plotkin's Sequentiality

Classic results with simpler inductive proofs

Scott's Continuity

Berry's Stability

Commutation Theorem
$\mathrm{NF}(\mathcal{T}(P)) = \mathcal{T}(\mathrm{BT}(P))$

$\nexists$ parallel or

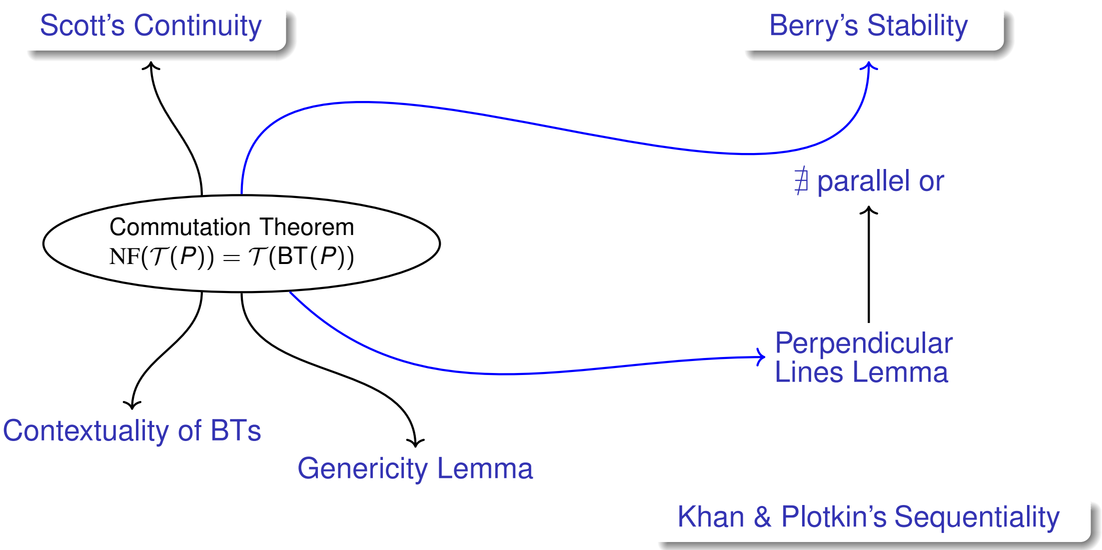
Perpendicular Lines Lemma

Contextuality of BTs

Genericity Lemma

Khan & Plotkin's Sequentiality

Classic results with simpler inductive proofs



A proof of context closure via Taylor Expansion

Contextuality of Böhm trees

$$\mathrm{BT}(M) = \mathrm{BT}(N) \quad \Rightarrow \quad \forall C[\,] \,.\, \mathrm{BT}(C[M]) = \mathrm{BT}(C[N])$$

Proof. By structural induction on $C[\,]$. Assuming $\mathrm{NF}(\mathcal{T}(M)) = \mathrm{NF}(\mathcal{T}(N))$, we have to prove:

$$\mathrm{NF}(\mathcal{T}(C[M])) = \mathrm{NF}(\mathcal{T}(C[N]))$$

The only difficult case is application: $C[\,] = (C_1[\,])\,(C_2[\,])$.

Take $t \in \mathrm{NF}(\mathcal{T}(C[M]))$, then $\exists t' \in \mathcal{T}((C_1[M])(C_2[M]))$ such that

$$t' = \qquad s_1[u_1, \dots, u_k] \longrightarrow\!\!\!\!\!\!\rightarrow t + \mathbb{T}$$

A proof of context closure via Taylor Expansion

Contextuality of Böhm trees

$$\mathrm{BT}(M) = \mathrm{BT}(N) \quad \Rightarrow \quad \forall C[] \,.\, \mathrm{BT}(C[M]) = \mathrm{BT}(C[N])$$

Proof. By structural induction on $C[]$. Assuming $\mathrm{NF}(\mathcal{T}(M)) = \mathrm{NF}(\mathcal{T}(N))$, we have to prove:

$$\mathrm{NF}(\mathcal{T}(C[M])) = \mathrm{NF}(\mathcal{T}(C[N]))$$

The only difficult case is application: $C[] = (C_1[])\,(C_2[])$.

Take $t \in \mathrm{NF}(\mathcal{T}(C[M]))$, then $\exists t' \in \mathcal{T}((C_1[M])(C_2[M]))$ such that

$$t' = \qquad s_1[u_1, \ldots, u_k] \longrightarrow\!\!\!\!\to t + \mathbb{T}$$

with $s_1 \in \mathcal{T}(C_1[M])$
and $u_1, \ldots, u_k \in \mathcal{T}(C_2[M])$.

A proof of context closure via Taylor Expansion

Contextuality of Böhm trees

$$\mathrm{BT}(M) = \mathrm{BT}(N) \quad \Rightarrow \quad \forall C[]\ .\ \mathrm{BT}(C[M]) = \mathrm{BT}(C[N])$$

Proof. By structural induction on $C[]$. Assuming $\mathrm{NF}(\mathcal{T}(M)) = \mathrm{NF}(\mathcal{T}(N))$, we have to prove:

$$\mathrm{NF}(\mathcal{T}(C[M])) = \mathrm{NF}(\mathcal{T}(C[N]))$$

The only difficult case is application: $C[] = (C_1[])(C_2[])$.

Take $t \in \mathrm{NF}(\mathcal{T}(C[M]))$, then $\exists t' \in \mathcal{T}((C_1[M])(C_2[M]))$ such that

$$t' = \begin{tikzcd} s_1[u_1, \ldots, u_k] \arrow[r, ->>] \arrow[d, ->>] & t + \mathbb{T} \\ \mathrm{nf}(s_1)[\mathrm{nf}(u_1), \ldots, \mathrm{nf}(u_k)] \arrow[ur, ->>] & \end{tikzcd}$$

with $\mathrm{nf}(s_1) \in \mathrm{NF}(\mathcal{T}(C_1[M]))$
and $\mathrm{nf}(u_1), \ldots, \mathrm{nf}(u_k) \in \mathrm{NF}(\mathcal{T}(C_2[M]))$

A proof of context closure via Taylor Expansion

Contextuality of Böhm trees

$$\mathsf{BT}(M) = \mathsf{BT}(N) \quad \Rightarrow \quad \forall C[\,]\,.\,\mathsf{BT}(C[M]) = \mathsf{BT}(C[N])$$

Proof. By structural induction on $C[\,]$. Assuming $\mathrm{NF}(\mathcal{T}(M)) = \mathrm{NF}(\mathcal{T}(N))$, we have to prove:

$$\mathrm{NF}(\mathcal{T}(C[M])) = \mathrm{NF}(\mathcal{T}(C[N]))$$

The only difficult case is application: $C[\,] = (C_1[\,])\,(C_2[\,])$.

Take $t \in \mathrm{NF}(\mathcal{T}(C[M]))$, then $\exists t' \in \mathcal{T}((C_1[M])(C_2[M]))$ such that

$$t' = \begin{tikzcd} s_1[u_1, \ldots, u_k] \arrow[r, ->>] \arrow[d, ->>] & t + \mathbb{T} \\ \mathrm{nf}(s_1)[\mathrm{nf}(u_1), \ldots, \mathrm{nf}(u_k)] \arrow[ur, ->>] & \end{tikzcd}$$

with $\mathrm{nf}(s_1) \in \mathrm{NF}(\mathcal{T}(C_1[M])) = \mathrm{NF}(\mathcal{T}(C_1[N]))$
and $\mathrm{nf}(u_1), \ldots, \mathrm{nf}(u_k) \in \mathrm{NF}(\mathcal{T}(C_2[M])) = \mathrm{NF}(\mathcal{T}(C_2[N]))$. We conclude that $t \in \mathrm{NF}(\mathcal{T}(C[N]))$. $\square$

The Genericity Lemma

Genericity Lemma

Let M unsolvable. If $C[M]$ has a β-nf, then $C[-]$ is constant (i.e., $\forall N \in \Lambda \,.\, C[N] =_\beta C[M]$).

Standard proof: Topological method.
Compactification points in the tree topology are precisely the unsolvables.

Several proofs in the literature:

Proving the Genericity Lemma by Leftmost Reduction is Simple

Jan Kuper

University of Twente, Department of Computer Science
P.O.Box 217, 7500 AE Enschede, The Netherlands
e-mail: jankuper@cs.utwente.nl

Abstract. The Genericity Lemma is one of the most important motivations to take in the untyped lambda calculus the notion of solvability as a formal representation of the informal notion of undefinedness. We generalise solvability towards typed lambda calculi, and we call this generalisation: *usability*. We then prove the Genericity Lemma for un-usable terms. The technique of the proof is based on *leftmost* reduction, which strongly simplifies the standard proof.

A Simple Proof of the Genericity Lemma

Masako Takahashi

Department of Information Science
Tokyo Institute of Technology
Ookayama, Meguro, Tokyo 152 Japan
masako@titisha.is.titech.ac.jp

Abstract. A short direct proof is given for the fundamental property of unsolvable λ-terms; if M is an unsolvable λ-term and $C[M]$ is solvable, then $C[N]$ is solvable for any λ-term N. (Here $C[\ ]$ stands for an arbitrary context.)

1. Preliminaries

A term in this note means a λ-term, which is either $x, \lambda x.M$ or MN, (where M, N are terms and x is a variable.) Unless otherwise stated, capital letters $M, N, P, \ldots$ stand for arbitrary terms, $\mathbf{M}, \mathbf{N}, \ldots$ for (possibly null) sequences of terms, $x, y, \ldots$ for variables, and $\mathbf{x}, \mathbf{y}, \ldots$ for (possibly null) sequences of variables. We refer to [1] as the standard text in the field.

A term of the form $\lambda \mathbf{x}.y\mathbf{M}$ (more precisely, $\lambda x_1.(\lambda x_2.(\ldots(\lambda x_n.((\ldots((yM_1)M_2)\ldots)M_m))\ldots))$ for some $n, m \geq 0$) is said to be in *head normal form* (*hnf*, for short). If a term M has a hnf (that is, $M =_\beta M'$ for a term M' in hnf), then M is called *solvable*. The following are well-known facts of solvable terms (cf.[1]

The Genericity Lemma

Genericity Lemma

Let M unsolvable. If $C[M]$ has a β-nf, then $C[-]$ is constant (i.e., $\forall N \in \Lambda \,.\, C[N] =_\beta C[M]$).

Proof. If $C[M]$ normalizable then there is a linearized $t \in \mathrm{NF}(\mathcal{T}(C[M]))$ such that

$$t \cong \mathrm{nf}(C[M])$$

So, there exist $t' \in \mathcal{T}(C[M])$ such that:

$$t' = \qquad c\langle s_1, \ldots, s_k \rangle \longrightarrow\!\!\!\!\!\rightarrow t + \mathbb{T}$$

for some $c \in \mathcal{T}(C[-])$ and $s_1, \ldots, s_k \in \mathcal{T}(M)$.

The Genericity Lemma

Genericity Lemma

Let M unsolvable. If $C[M]$ has a β-nf, then $C[-]$ is constant (i.e., $\forall N \in \Lambda \,.\, C[N] =_\beta C[M]$).

Proof. If $C[M]$ normalizable then there is a linearized $t \in \mathrm{NF}(\mathcal{T}(C[M]))$ such that

$$t \cong \mathrm{nf}(C[M])$$

So, there exist $t' \in \mathcal{T}(C[M])$ such that:

$$t' = \quad \begin{tikzcd} c\llparenthesis s_1, \ldots, s_k \rrparenthesis \arrow[r, twoheadrightarrow] \arrow[d] & t + \mathbb{T} \\ c\llparenthesis \mathrm{nf}(s_1), \ldots, \mathrm{nf}(s_k) \rrparenthesis \arrow[ur, twoheadrightarrow] & \end{tikzcd}$$

for some $c \in \mathcal{T}(C[-])$ and $s_1, \ldots, s_k \in \mathcal{T}(M)$. (By Confluence and Strong Normalization.)

The Genericity Lemma

Genericity Lemma

Let M unsolvable. If $C[M]$ has a β-nf, then $C[-]$ is constant (i.e., $\forall N \in \Lambda \,.\, C[N] =_\beta C[M]$).

Proof. If $C[M]$ normalizable then there is a linearized $t \in \mathrm{NF}(\mathcal{T}(C[M]))$ such that

$$t \cong \mathrm{nf}(C[M])$$

So, there exist $t' \in \mathcal{T}(C[M])$ such that:

$$t' = \begin{tikzcd} c\llparenthesis s_1, \dots, s_k \rrparenthesis \arrow[r, twoheadrightarrow] \arrow[d] & t + \mathbb{T} \\ c\llparenthesis 0, \dots, 0 \rrparenthesis \arrow[ur, twoheadrightarrow] & \end{tikzcd}$$

for some $c \in \mathcal{T}(C[-])$ and $s_1, \dots, s_k \in \mathcal{T}(M)$. Since M unsolvable entails $\mathrm{nf}(s_i) = 0$.

The Genericity Lemma

Genericity Lemma

Let M unsolvable. If $C[M]$ has a β-nf, then $C[-]$ is constant (i.e., $\forall N \in \Lambda\,.\, C[N] =_\beta C[M]$).

Proof. If $C[M]$ normalizable then there is a linearized $t \in \mathrm{NF}(\mathcal{T}(C[M]))$ such that

$$t \cong \mathrm{nf}(C[M])$$

So, there exist $t' \in \mathcal{T}(C[M])$ such that:

$$t' = \quad \begin{tikzcd} c\langle\!|\, s_1, \dots, s_k \,|\!\rangle \arrow[r, ->>] \arrow[d, ->>] & t + \mathbb{T} \\ c\langle\!|\, 0, \dots, 0 \,|\!\rangle \arrow[ur, ->>] & \end{tikzcd}$$

for some $c \in \mathcal{T}(C[-])$ and $s_1, \dots, s_k \in \mathcal{T}(M)$. Since M unsolvable entails $\mathrm{nf}(s_i) = 0$.
Therefore, no hole may actually occur in $c\langle\!|\, - \,|\!\rangle$ so we get:

$$c\langle\!|\, s_1, \dots, s_k \,|\!\rangle \in \mathcal{T}(C[N]) \quad \Rightarrow \quad t \in \mathrm{NF}(\mathcal{T}(C[N]))$$

and since t is linearized we obtain $\mathrm{nf}_\beta(C[N]) \cong t$. $\square$

For more details and proofs. . . look in the paper!

POPL 2020

1

Taylor Subsumes Scott, Berry, Kahn and Plotkin*

DAVIDE BARBAROSSA, Université Paris 13, Sorbonne Paris Cité, LIPN, CNRS UMR 7030, France
GIULIO MANZONETTO, Université Paris 13, Sorbonne Paris Cité, LIPN, CNRS UMR 7030, France

The speculative ambition of replacing the old theory of program approximation based on syntactic continuity with the theory of resource consumption based on Taylor expansion and originating from the differential λ-calculus is nowadays at hand. Using this resource sensitive theory, we provide simple proofs of important results in λ-calculus that are usually demonstrated by exploiting Scott's continuity, Berry's stability or Kahn and Plotkin's sequentiality theory. A paradigmatic example is given by the Perpendicular Lines Lemma for the Böhm tree semantics, which is proved here simply by induction, but relying on the main properties of resource approximants: strong normalization, confluence and linearity.

CCS Concepts: • **Theory of computation** → **Lambda calculus**; **Linear logic**.

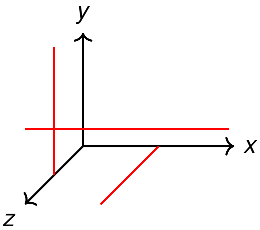
Additional Key Words and Phrases: Lambda calculus, Taylor expansion, Böhm trees, Linear Logic.

ACM Reference Format:
Davide Barbarossa and Giulio Manzonetto. 2020. Taylor Subsumes Scott, Berry, Kahn and Plotkin. *Proc. ACM Program. Lang.* 4, POPL, Article 1 (January 2020), 23 pages. https://doi.org/10.1145/3371069

Perpendicular Lines Lemma

PLL: *If a context* $C[-_1, \ldots, -_n] : \Lambda^n \to \Lambda$ *is constant on* n *perpendicular lines, then it must be constant everywhere.*

$\mathbb{R}^3$

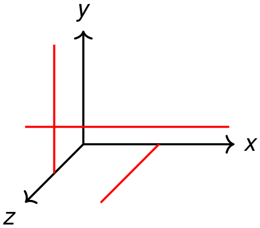


$$\ell_1 = \{(x, 1, 2) \mid x \in \mathbb{R}\},$$
$$\ell_2 = \{(0, y, 1) \mid y \in \mathbb{R}\},$$
$$\ell_3 = \{(1, 0, z) \mid z \in \mathbb{R}\}.$$

Perpendicular Lines Lemma

PLL: *If a context $C[-_1, \ldots, -_n] : \Lambda^n \to \Lambda$ is constant on n perpendicular lines, then it must be constant everywhere.*

Λ^3



$$\ell_1 = \{(X, \lambda x.x, \lambda xy.x) \mid X \in \Lambda\},$$
$$\ell_2 = \{(\lambda xy.y, Y, \lambda x.x) \mid Y \in \Lambda\},$$
$$\ell_3 = \{(\lambda x.x, \lambda xy.x, Z) \mid Z \in \Lambda\}.$$

Known results:

- $\mathcal{M}(\mathcal{B}) \models$ PLL, Barendregt's Book 1982,
  Proof technique: Sequentiality.
- $\mathcal{M}^o(\mathcal{B}) \models$ PLL?
- $\mathcal{M}^o(\beta) \not\models$ PLL, by Barendregt & Statman 1999.
  Proof: Counterexample via Plotkin's terms.
- $\mathcal{M}(\beta) \models$ PLL, by De Vrijer & Endrullis 2008.
  Proof: via Reduction under Substitution.

PLL	β	$\mathcal{B}$
open	✓	✓
closed	✗	?

Perpendicular Lines Lemma

PLL: *If a context $C[-_1, \ldots, -_n] : \Lambda^n \to \Lambda$ is constant on n perpendicular lines, then it must be constant everywhere.*

Perpendicular Lines Lemma

$$\forall Z \begin{cases} C[Z, M_{12}, \ldots\ldots, M_{1n}] & =_{\mathcal{B}} & N_1 \\ C[M_{21}, Z, \ldots\ldots, M_{2n}] & =_{\mathcal{B}} & N_2 \\ \ddots & \vdots & \vdots \\ C[M_{n1}, \ldots, M_{n(n-1)}, Z] & =_{\mathcal{B}} & N_n \end{cases}$$

$$\Downarrow$$

$$\forall \vec{Z} \,.\, C[Z_1, \ldots, Z_n] =_{\mathcal{B}} N_1 =_{\mathcal{B}} \cdots =_{\mathcal{B}} N_n$$

Perpendicular Lines Lemma

PLL: *If a context $C[-_1, \ldots, -_n] : \Lambda^n \to \Lambda$ is constant on n perpendicular lines, then it must be constant everywhere.*

Perpendicular Lines Lemma

$$\forall Z \left\{ \begin{array}{lcl} C[Z, M_{12}, \ldots\ldots, M_{1n}] & =_{\mathcal{B}} & N_1 \\ C[M_{21}, Z, \ldots\ldots, M_{2n}] & =_{\mathcal{B}} & N_2 \\ \ddots & \vdots & \vdots \\ C[M_{n1}, \ldots, M_{n(n-1)}, Z] & =_{\mathcal{B}} & N_n \end{array} \right.$$

$$\Downarrow$$

$$\forall \vec{Z} \,.\, C[Z_1, \ldots, Z_n] =_{\mathcal{B}} N_1 =_{\mathcal{B}} \cdots =_{\mathcal{B}} N_n$$

In $\mathcal{B}$ a context $C[-]$ can be constant for several reasons:

1. $C[-]$ does not contain the hole in the first place (the trivial case);
2. the hole is erased during its reduction;
3. the hole is “hidden” behind an unsolvable;
4. the hole is never erased but “pushed into infinity”.

Perpendicular Lines Lemma

PLL: *If a context $C[-_1, \dots, -_n] : \Lambda^n \to \Lambda$ is constant on n perpendicular lines, then it must be constant everywhere.*

Perpendicular Lines Lemma

$$\forall Z \left\{ \begin{array}{lcl} C[Z, M_{12}, \dots\dots, M_{1n}] & =_{\mathcal{B}} & N_1 \\ C[M_{21}, Z, \dots\dots, M_{2n}] & =_{\mathcal{B}} & N_2 \\ \ddots & \vdots & \vdots \\ C[M_{n1}, \dots, M_{n(n-1)}, Z] & =_{\mathcal{B}} & N_n \end{array} \right.$$

$$\Downarrow$$

$$\forall \vec{Z} \,.\, C[Z_1, \dots, Z_n] =_{\mathcal{B}} N_1 =_{\mathcal{B}} \cdots =_{\mathcal{B}} N_n$$

An approximant $c \in \mathcal{T}(C[-])$ such that $\mathrm{nf}(c) \neq 0$ can be constant for only one reason:

1. c does not contain the hole in the first place (the trivial case);
2. the hole is erased during its reduction ;
3. the hole is “hidden” behind an unsolvable;
4. the hole is never erased but “pushed into infinity”.

Perpendicular Lines Lemma

PLL: *If a context $C[-_1, \dots, -_n] : \Lambda^n \to \Lambda$ is constant on n perpendicular lines, then it must be constant everywhere.*

Perpendicular Lines Lemma

$$\forall Z \left\{ \begin{array}{lcl} C[Z, M_{12}, \dots\dots, M_{1n}] & =_{\mathcal{B}} & N_1 \\ C[M_{21}, Z, \dots\dots, M_{2n}] & =_{\mathcal{B}} & N_2 \\ \ddots & \vdots & \vdots \\ C[M_{n1}, \dots, M_{n(n-1)}, Z] & =_{\mathcal{B}} & N_n \end{array} \right.$$

$$\Downarrow$$

$$\forall \vec{Z} \,.\, C[Z_1, \dots, Z_n] =_{\mathcal{B}} N_1 =_{\mathcal{B}} \cdots =_{\mathcal{B}} N_n$$

An approximant $c \in \mathcal{T}(C[-])$ such that $\mathrm{nf}(c) \neq 0$ can be constant for only one reason:

1. c does not contain the hole in the first place (the trivial case);
2. ~~the hole is erased during its reduction~~ (linearity);
3. the hole is “hidden” behind an unsolvable;
4. the hole is never erased but “pushed into infinity”.

Perpendicular Lines Lemma

PLL: *If a context $C[-_1, \ldots, -_n] : \Lambda^n \to \Lambda$ is constant on n perpendicular lines, then it must be constant everywhere.*

Perpendicular Lines Lemma

$$\forall Z \left\{ \begin{array}{lcl} C[Z, M_{12}, \ldots\ldots, M_{1n}] & =_{\mathcal{B}} & N_1 \\ C[M_{21}, Z, \ldots\ldots, M_{2n}] & =_{\mathcal{B}} & N_2 \\ \ddots & \vdots & \vdots \\ C[M_{n1}, \ldots, M_{n(n-1)}, Z] & =_{\mathcal{B}} & N_n \end{array} \right.$$

$$\Downarrow$$

$$\forall \vec{Z} \,.\, C[Z_1, \ldots, Z_n] =_{\mathcal{B}} N_1 =_{\mathcal{B}} \cdots =_{\mathcal{B}} N_n$$

An approximant $c \in \mathcal{T}(C[-])$ such that $\mathrm{nf}(c) \neq 0$ can be constant for only one reason:

1. c does not contain the hole in the first place (the trivial case);
2. ~~the hole is erased during its reduction~~ (linearity);
3. ~~the hole is “hidden” behind an unsolvable~~ (strong normalization);
4. the hole is never erased but “pushed into infinity”.

Perpendicular Lines Lemma

PLL: *If a context $C[-_1, \dots, -_n] : \Lambda^n \to \Lambda$ is constant on n perpendicular lines, then it must be constant everywhere.*

Perpendicular Lines Lemma

$$\forall Z \left\{ \begin{array}{lcl} C[Z, M_{12}, \dots\dots, M_{1n}] & =_{\mathcal{B}} & N_1 \\ C[M_{21}, Z, \dots\dots, M_{2n}] & =_{\mathcal{B}} & N_2 \\ \ddots & \vdots & \vdots \\ C[M_{n1}, \dots, M_{n(n-1)}, Z] & =_{\mathcal{B}} & N_n \end{array} \right.$$

$$\Downarrow$$

$$\forall \vec{Z} \,.\, C[Z_1, \dots, Z_n] =_{\mathcal{B}} N_1 =_{\mathcal{B}} \cdots =_{\mathcal{B}} N_n$$

An approximant $c \in \mathcal{T}(C[-])$ such that $\mathrm{nf}(c) \neq 0$ can be constant for only one reason:

1. c does not contain the hole in the first place (the trivial case);
2. ~~the hole is erased during its reduction~~ (linearity);
3. ~~the hole is “hidden” behind an unsolvable~~ (strong normalization);
4. ~~the hole is never erased but “pushed into infinity”~~ (finiteness).

Perpendicular Lines Lemma

PLL: *If a context $C[-_1, \ldots, -_n] : \Lambda^n \to \Lambda$ is constant on n perpendicular lines, then it must be constant everywhere.*

Perpendicular Lines Lemma

$$\forall Z \begin{cases} C[Z, M_{12}, \ldots\ldots, M_{1n}] & =_{\mathcal{B}} & N_1 \\ C[M_{21}, Z, \ldots\ldots, M_{2n}] & =_{\mathcal{B}} & N_2 \\ \ddots & \vdots & \vdots \\ C[M_{n1}, \ldots, M_{n(n-1)}, Z] & =_{\mathcal{B}} & N_n \end{cases}$$

$$\Downarrow$$

$$\forall \vec{Z} \,.\, C[Z_1, \ldots, Z_n] =_{\mathcal{B}} N_1 =_{\mathcal{B}} \cdots =_{\mathcal{B}} N_n$$

Claim.

$\forall c \in \mathcal{T}(C[-_1, \ldots, -_n]), \mathrm{nf}(c) \neq 0 \quad \Rightarrow \quad c$ cannot contain any hole.

By induction on the size of c, using all the properties mentioned before.

PLL	β	$\mathcal{B}$
open	✓	✓
closed	✗	?

Perpendicular Lines Lemma

PLL: *If a context $C[-_1, \ldots, -_n] : \Lambda^n \to \Lambda$ is constant on n perpendicular lines, then it must be constant everywhere.*

Perpendicular Lines Lemma

$$\forall Z \begin{cases} C[Z, M_{12}, \ldots\ldots, M_{1n}] & =_{\mathcal{B}} & N_1 \\ C[M_{21}, Z, \ldots\ldots, M_{2n}] & =_{\mathcal{B}} & N_2 \\ \ddots & \vdots & \vdots \\ C[M_{n1}, \ldots, M_{n(n-1)}, Z] & =_{\mathcal{B}} & N_n \end{cases}$$

$$\Downarrow$$

$$\forall \vec{Z} \, . \, C[Z_1, \ldots, Z_n] =_{\mathcal{B}} N_1 =_{\mathcal{B}} \cdots =_{\mathcal{B}} N_n$$

Our proof does not need open terms!

$\mathcal{M}^o(\mathcal{B}) \models$ PLL ✓

PLL	β	$\mathcal{B}$
open	✓	✓
closed	✗	✓

These techniques extends easily to...

Other paradigms

- Revisiting Call-by-value Böhm trees in light of their Taylor expansion.
  Axel Kerinec, Giulio Manzonetto, Michele Pagani. Log. Methods Comput. Sci. 16(3) (2020)
- Taylor expansion for Call-By-Push-Value.
  Jules Chouquet, Christine Tasson. CSL 2020: 16:1-16:16

Other primitives (e.g., call-cc)

- Towards a resource based approximation theory of programs.
  Davide Barbarossa. PhD thesis. 2021.

Other kinds of "effects"

- Normalizing the Taylor expansion of non-deterministic λ-terms, via parallel reduction of resource vectors.
  Lionel Vaux. Log. Methods Comput. Sci. 15(3) (2019)
- On the Taylor Expansion of Probabilistic lambda-terms.
  Ugo Dal Lago, Thomas Leventis, FSCD 2019: 13:1-13:16.
- ⋮

Thanks for your attention!