Coinductive Intersection Types are Completely Unsound

Pierre VIAL
IRIF, Paris 7

Rencontres GeoCal – LAC

December 8, 2016

INVARIANTS OF EXECUTION

- In the course of its execution, a program passes through different states.

Invariants of Execution

- In the course of its execution, a program passes through different states.
- The state of a program at the beginning of the execution and at its end may be very different.

Invariants of Execution

- In the course of its execution, a program passes through different states.
- The state of a program at the beginning of the execution and at its end may be very different.
- Finding a **denotation** to a program = assigning to it an **invariant of execution** (*i.e.* an object that must the same for all its states).

INVARIANTS OF EXECUTION

- In the course of its execution, a program passes through different states.
- The state of a program at the beginning of the execution and at its end may be very different.
- Finding a **denotation** to a program = assigning to it an **invariant of execution** (*i.e.* an object that must the same for all its states).
- The denotation of a program gives us some informations about its behaviour. Usually, **dynamical information** (related to its execution).

INVARIANTS OF EXECUTION

- In the course of its execution, a program passes through different states.
- The state of a program at the beginning of the execution and at its end may be very different.
- Finding a **denotation** to a program = assigning to it an **invariant of execution** (*i.e.* an object that must the same for all its states).
- The denotation of a program gives us some informations about its behaviour. Usually, **dynamical information** (related to its execution).
- Usually, the information by a denotation implies that the concerned program is **terminating**.

INVARIANTS OF EXECUTION

- In the course of its execution, a program passes through different states.
- The state of a program at the beginning of the execution and at its end may be very different.
- Finding a **denotation** to a program = assigning to it an **invariant of execution** (*i.e.* an object that must the same for all its states).
- The denotation of a program gives us some informations about its behaviour. Usually, **dynamical information** (related to its execution).
- Usually, the information by a denotation implies that the concerned program is **terminating**.
- Another use of denotations: **equating** or **separating programs** *i.e.* two states that have different denotations cannot be instances of the same program.

Types as Invariants of Execution

- λ-terms: programs, β-reduction step: execution step.

TYPES AS INVARIANTS OF EXECUTION

- λ-terms: programs, β-reduction step: execution step.
- **Normalizability:** termination.

  Many variants: head-n, weak-n, strong-n,...

TYPES AS INVARIANTS OF EXECUTION

- λ-terms: programs, β-reduction step: execution step.
- **Normalizability:** termination.
  Many variants: head-n, weak-n, strong-n,...
- **Types:** check *statically* (without reducing) that a term is normalizable (**soundness** of a type system).

TYPES AS INVARIANTS OF EXECUTION

- λ-terms: programs, β-reduction step: execution step.
- **Normalizability:** termination.
  Many variants: head-n, weak-n, strong-n,...
- **Types:** check *statically* (without reducing) that a term is normalizable (**soundness** of a type system).
- Typing: assigning formulas (called *types*) to variables.
  The type of a λ-term can be computed, if some *typing rules* are respected.

Types as Invariants of Execution

- λ-terms: programs, β-reduction step: execution step.
- **Normalizability:** termination.
  Many variants: head-n, weak-n, strong-n,...
- **Types:** check *statically* (without reducing) that a term is normalizable (**soundness** of a type system).
- Typing: assigning formulas (called *types*) to variables.
  The type of a λ-term can be computed, if some *typing rules* are respected.
- When a type system enjoys **subject reduction** and **expansion**, types are execution invariants (and they usually provide us with models of λ-calculus).

NON-TERMINATING PROGRAMS

- Often given an "empty" denotation (a model that equates all the non-terminating terms is said to be **sensible**). However:

NON-TERMINATING PROGRAMS

- Often given an "empty" denotation (a model that equates all the non-terminating terms is said to be **sensible**). However:
- Not all non-terminating progams are *meaningless*.
  (For instance, streams, a program keeping on printing the list of prime numbers, fixpoint combinators...)

NON-TERMINATING PROGRAMS

- Often given an "empty" denotation (a model that equates all the non-terminating terms is said to be **sensible**). However:
- Not all non-terminating progams are *meaningless*.
  (For instance, streams, a program keeping on printing the list of prime numbers, fixpoint combinators…)
- Some programs are non terminating but **productive**.

NON-TERMINATING PROGRAMS

- Often given an "empty" denotation (a model that equates all the non-terminating terms is said to be **sensible**). However:
- Not all non-terminating progams are *meaningless*.
  (For instance, streams, a program keeping on printing the list of prime numbers, fixpoint combinators. . .)
- Some programs are non terminating but **productive**.
- Many possible definitions or variants of sound non termination
  Klop and alii[95], Endrullis,Polonsky and alii[15]

CONTRIBUTIONS

Using type theory, we build a **completely unsound** type system and a **non-sensible model** of pure λ-calculus in which:

- Every term has a non-empty denotation (including the **mute terms**).
- Terms are discriminated according to their **order** (the maximal number of abs that prefixes a reduct).

CONTRIBUTIONS

Using type theory, we build a **completely unsound** type system and a **non-sensible model** of pure λ-calculus in which:

- Every term has a non-empty denotation (including the **mute terms**).
- Terms are discriminated according to their **order** (the maximal number of abs that prefixes a reduct).

Related works

- Jacopino[75]: **easy** terms (t is easy if it can be consistently equated to any other term)
- Berarducci[96]: **mute** terms (”The most undefined terms”).
- Bucciarelli,Carraro,Favro,Salibra[15]: *Graph easy Sets of mute lambda terms*, TCS.

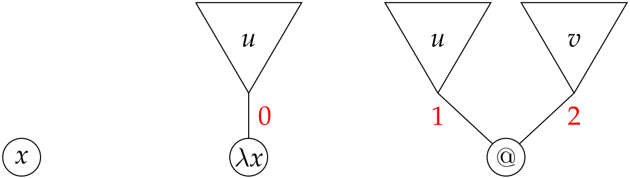
Plan

Types and Relevance

Coinductive Types

System S (sequential intersection)

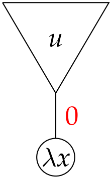
λ-TERMS AS LABELLED TREES



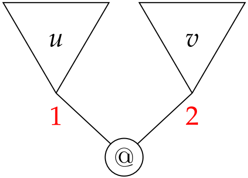
Variable x **Abstraction** $\lambda x.u$ **Application** $u\,v$

λ-terms as labelled trees

Variable x

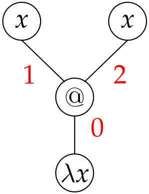


Abstraction $\lambda x.u$



Application $u\,v$

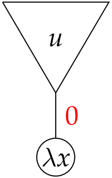
$$\Delta = \lambda x.xx$$



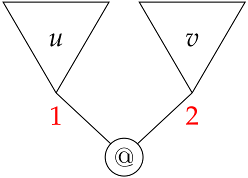
$\lambda x.xx$

λ-TERMS AS LABELLED TREES

Variable x

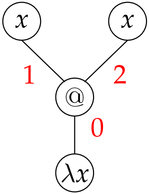


Abstraction $\lambda x.u$



Application $u\,v$

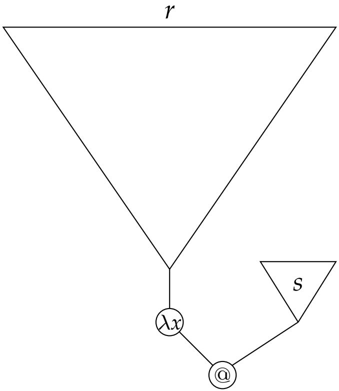
$\Delta = \lambda x.xx$



$\lambda x.xx$

Position: finite sequence in $\{0,1,2\}^*$, *e.g.* $0 \cdot 0 \cdot 2 \cdot 1 \cdot 2$.

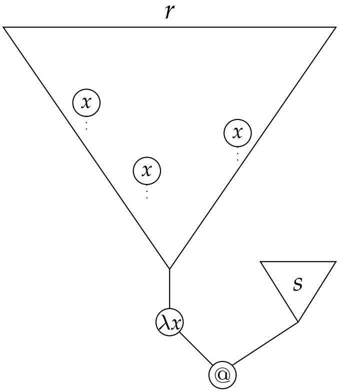
β-REDUCTION



Redex:

$(\lambda x.r)s$

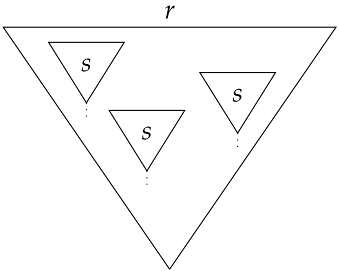
β-REDUCTION



Redex:

$(\lambda x.r)s$

β-Reduction

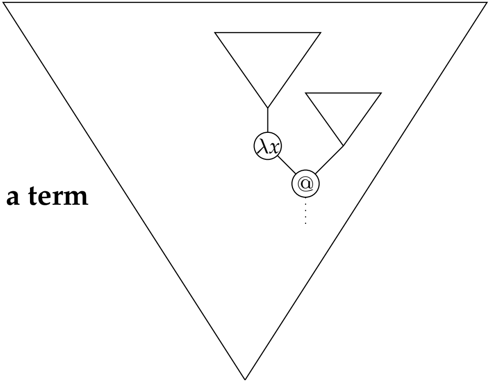


Reduct:

$r[s/x]$

β-REDUCTION

Redex
nested in a term



(UN)SOUNDNESS

- If a Type System is able to type a non (weakly) head-normalizing term (*e.g.* $\Omega = \Delta\,\Delta$), it is said here to be **unsound**.

(UN)SOUNDNESS

- If a Type System is able to type a non (weakly) head-normalizing term (*e.g.* $\Omega = \Delta\,\Delta$), it is said here to be **unsound**.
- If a Type System is able to type every term, it is said to be **completely unsound**.

(UN)SOUNDNESS

- If a Type System is able to type a non (weakly) head-normalizing term (*e.g.* $\Omega = \Delta\,\Delta$), it is said here to be **unsound**.
- If a Type System is able to type every term, it is said to be **completely unsound**.
- With SR and SE, a completely unsound type system should yield a model for pure λ-calculus.

TYPING RULES OF $\mathscr{R}_0$ (GARDNER/DE CARVALHO)

Types (τ, σ_i): $\tau, \sigma_i := \ o \in \mathscr{O} \mid [\sigma_i]_{i \in I} \to \tau.$

Context (Γ, Δ): assign *intersection* types to variables.

$$\frac{}{x : [\tau] \vdash x : \tau}\ \mathtt{ax} \qquad \frac{\Gamma, x : [\sigma_i]_{i\in I} \vdash t : \tau}{\Gamma \vdash \lambda x.t : [\sigma_i]_{i\in I} \to \tau}\ \mathtt{abs}$$

$$\frac{\Gamma \vdash t : [\sigma_i]_{i\in I} \to \tau \qquad (\Delta_i \vdash u : \sigma_i)_{i\in I}}{\Gamma +_{i\in I} \Delta_i \vdash t\,u : \tau}\ \mathtt{app}$$

Examples:

$$\cfrac{\cfrac{}{x : [\tau] \vdash x : \tau}\ \mathtt{ax}}{\vdash \lambda x.x : [\tau] \to \tau}\ \mathtt{abs} \qquad \cfrac{\cfrac{}{x : [\tau] \vdash x : \tau}\ \mathtt{ax}}{x : [\tau] \vdash \lambda y.x : [\,] \to \tau}\ \mathtt{abs}$$

Relevance vs Irrelevance

- **Observation:** In system $\mathscr{R}_0$, $\lambda x.x$ (resp. $\lambda y.x$) can only be typed with a type of the form $[\tau] \to \tau$ (resp. $[\,] \to \tau$).

Relevance vs Irrelevance

- **Observation:** In system $\mathscr{R}_0$, $\lambda x.x$ (resp. $\lambda y.x$) can only be typed with a type of the form $[\tau] \to \tau$ (resp. $[\,] \to \tau$).
- System $\mathscr{R}_0$ is said to be **relevant**: *weakening* is not allowed.

RELEVANCE VS IRRELEVANCE

- **Observation:** In system $\mathscr{R}_0$, $\lambda x.x$ (resp. $\lambda y.x$) can only be typed with a type of the form $[\tau] \to \tau$ (resp. $[\,] \to \tau$).
- System $\mathscr{R}_0$ is said to be **relevant**: *weakening* is not allowed. For instance, a type is used when it is assigned:

$$\frac{}{x : [\sigma] \vdash x : \sigma}\ \texttt{ax}$$

RELEVANCE VS IRRELEVANCE

- **Observation:** In system $\mathscr{R}_0$, $\lambda x.x$ (resp. $\lambda y.x$) can only be typed with a type of the form $[\tau] \to \tau$ (resp. $[\,] \to \tau$).
- System $\mathscr{R}_0$ is said to be **relevant**: *weakening* is not allowed. For instance, a type is used when it is assigned:

$$\frac{}{x : [\sigma] \vdash x : \sigma}\ \texttt{ax}$$

- If we replace `ax` by `axw`:

$$\frac{i_0 \in I}{\Gamma;\ x : [\sigma_i]_{i \in I} \vdash x : \sigma_{i_0}}\ \texttt{axw}$$

... we obtain an irrelevant system, called $\mathscr{R}_{0,\texttt{w}}$.

RELEVANCE VS IRRELEVANCE

- **Observation:** In system $\mathscr{R}_0$, $\lambda x.x$ (resp. $\lambda y.x$) can only be typed with a type of the form $[\tau] \to \tau$ (resp. $[\,] \to \tau$).
- System $\mathscr{R}_0$ is said to be **relevant**: *weakening* is not allowed. For instance, a type is used when it is assigned:

$$\frac{}{x : [\sigma] \vdash x : \sigma}\ \texttt{ax}$$

- If we replace `ax` by `axw`:

$$\frac{i_0 \in I}{\Gamma;\ x : [\sigma_i]_{i \in I} \vdash x : \sigma_{i_0}}\ \texttt{axw}$$

... we obtain an irrelevant system, called $\mathscr{R}_{0,\texttt{w}}$.

- In $\mathscr{R}_{0,\texttt{w}}$, we may derive:

$$\frac{\dfrac{}{x : [\tau, \tau_1, \tau_1] \vdash x : \tau}\ \texttt{axw}}{\vdash \lambda x.x : [\tau, \tau_1, \tau_2] \to \tau}\ \texttt{abs} \qquad \frac{\dfrac{}{x : [\tau], y : [\tau] \vdash x : \tau}\ \texttt{axw}}{x : [\tau] \vdash \lambda y.x : [\tau] \to \tau}\ \texttt{abs}$$

Subject Reduction Property for $\mathscr{R}_0$

If $\Pi \rhd \Gamma \vdash t : \tau$ and $t \to t'$, then $\exists \Pi' \rhd \Gamma \vdash t' : \tau$

SUBJECT REDUCTION PROPERTY FOR $\mathscr{R}_0$

If $\Pi \triangleright \Gamma \vdash t : \tau$ and $t \to t'$, then $\exists \Pi' \triangleright \Gamma \vdash t' : \tau$

$$(\lambda x.r)s \to r[s/x]$$

$$\dfrac{\dfrac{\begin{array}{c}\Pi_r \\ \vdots \\ \Gamma, x : [\sigma_i]_{i\in I} \vdash r : \tau\end{array}}{\Gamma \vdash \lambda x.r : [\sigma_i]_{i\in I} \to \tau}\,\texttt{abs} \qquad \left(\begin{array}{c}\Pi_i \\ \vdots \\ \Delta_i \vdash s : \sigma_i\end{array}\right)^{i\in I}}{\Gamma + \sum_{i\in I} \Delta_i \vdash (\lambda x.r)s : \tau}\,\texttt{app}$$

SUBJECT REDUCTION PROPERTY FOR $\mathscr{R}_0$

If $\Pi \rhd \Gamma \vdash t : \tau$ and $t \to t'$, then $\exists \Pi' \rhd \Gamma \vdash t' : \tau$

$$(\lambda x.r)s \to r[s/x]$$

Axiom leaves typing x inside Π_r

$$\dfrac{\dfrac{\begin{array}{c}\Pi_r \quad \left(\dfrac{}{x : [\sigma_i] \vdash x : \sigma_i}\,\texttt{ax}\right)_{i\in I}\\ \vdots \\ \Gamma, x : [\sigma_i]_{i\in I} \vdash r : \tau\end{array}}{\Gamma \vdash \lambda x.r : [\sigma_i]_{i\in I} \to \tau}\,\texttt{abs} \qquad \left(\begin{array}{c}\Pi_i\\ \vdots \\ \Delta_i \vdash s : \sigma_i\end{array}\right)^{i\in I}}{\Gamma + \sum_{i\in I}\Delta_i \vdash (\lambda x.r)s : \tau}\,\texttt{app}$$

SUBJECT REDUCTION PROPERTY FOR $\mathscr{R}_0$

If $\Pi \rhd \Gamma \vdash t : \tau$ and $t \to t'$, then $\exists \Pi' \rhd \Gamma \vdash t' : \tau$

$$(\lambda x.r)s \to r[s/x]$$

$$\dfrac{\dfrac{\begin{array}{c}\Pi_r \\ \left(\dfrac{}{x : [\sigma_i] \vdash x : \boxed{\sigma_i}}\,\texttt{ax}\right)_{i \in I} \\ \vdots \\ \Gamma, x : [\sigma_i]_{i \in I} \vdash r : \tau\end{array}}{\Gamma \vdash \lambda x.r : [\sigma_i]_{i \in I} \to \tau}\,\texttt{abs} \qquad \left(\begin{array}{c}\Pi_i \\ \vdots \\ \Delta_i \vdash s : \boxed{\sigma_i}\end{array}\right)^{i \in I}}{\Gamma + \sum_{i \in I} \Delta_i \vdash (\lambda x.r)s : \tau}\,\texttt{app}$$

Subject Reduction Property for $\mathscr{R}_0$

If $\Pi \rhd \Gamma \vdash t : \tau$ and $t \to t'$, then $\exists \Pi' \rhd \Gamma \vdash t' : \tau$

$$(\lambda x.r)s \to r[s/x]$$

$$\dfrac{\dfrac{\begin{array}{c}\Pi_r\\ \left(\dfrac{}{x : [\sigma_i] \vdash x : \boxed{\sigma_i}}\,\texttt{ax}\right)_{i\in I}\\ \vdots\\ \Gamma, x : [\sigma_i]_{i\in I} \vdash r \quad : \tau\end{array}}{\Gamma \vdash \lambda x.r : [\sigma_i]_{i\in I} \to \tau}\,\texttt{abs} \qquad \left(\begin{array}{c}\Pi_i\\ \vdots\\ \Delta_i \vdash s : \boxed{\sigma_i}\end{array}\right)^{i\in I}}{\Gamma + \sum_{i\in I}\Delta_i \vdash (\lambda x.r)s : \tau}\,\texttt{app}$$

"association"

SUBJECT REDUCTION PROPERTY FOR $\mathscr{R}_0$

If $\Pi \triangleright \Gamma \vdash t : \tau$ and $t \to t'$, then $\exists \Pi' \triangleright \Gamma \vdash t' : \tau$

$$(\lambda x.r)s \to r[s/x]$$

“association”

$$\dfrac{\dfrac{\begin{array}{c}\Pi_r\\ \left(\dfrac{}{\cancel{x : [\sigma_i] \vdash x} : \boxed{\sigma_i}}\,\texttt{ax}\right)_{i\in I}\\ \vdots\\ \Gamma, x : [\sigma_i]_{i\in I} \vdash r \quad : \tau\end{array}}{\cancel{\Gamma \vdash \lambda x.r : [\sigma_i]_{i\in I} \to \tau}}\,\texttt{abs} \qquad \left(\begin{array}{c}\Pi_i\\ \vdots\\ \Delta_i \vdash s : \boxed{\sigma_i}\end{array}\right)^{i\in I}}{\cancel{\Gamma + \sum_{i\in I}\Delta_i \vdash (\lambda x.r)s : \tau}}\,\texttt{app}$$

SUBJECT REDUCTION PROPERTY FOR $\mathscr{R}_0$

If $\Pi \triangleright \Gamma \vdash t : \tau$ and $t \to t'$, then $\exists \Pi' \triangleright \Gamma \vdash t' : \tau$

$$(\lambda x.r)s \to r[s/x]$$

$$\dfrac{\Pi_r \qquad \left(\dfrac{\Pi_i}{\Delta_i \vdash s : \sigma_i}\right)^{i \in I}}{\Gamma + \sum_{i \in I} \Delta_i \vdash r\,[s/x] : \tau}$$

Plan

Types and Relevance

Coinductive Types

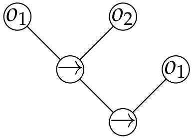
System S (sequential intersection)

INDUCTIVE VS COINDUCTIVE TYPES

Examples with Simple Types

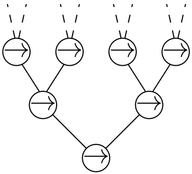
Inductive type:

$o_1 \to o_2 \to o_1$



Coinductive type:

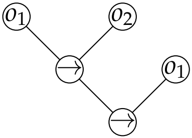
$A_{\texttt{ref}} = A_{\texttt{ref}} \to A_{\texttt{ref}}$



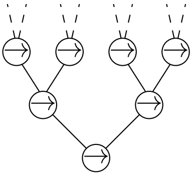
INDUCTIVE VS COINDUCTIVE TYPES

Examples with Simple Types

Inductive type:
$o_1 \to o_2 \to o_1$



Coinductive type:
$A_{\mathtt{ref}} = A_{\mathtt{ref}} \to A_{\mathtt{ref}}$



$A_{\mathtt{ref}}$ is a **reflexive type**.

COINDUCTIVE TYPE SYSTEMS

- We consider two new type systems $\mathscr{R}$ and $\mathscr{R}_{\mathtt{w}}$, having the same rules as $\mathscr{R}_0$ and $\mathscr{R}_{0,\mathtt{w}}$, but using coinductive types.

COINDUCTIVE TYPE SYSTEMS

- We consider two new type systems $\mathscr{R}$ and $\mathscr{R}_{\mathtt{w}}$, having the same rules as $\mathscr{R}_0$ and $\mathscr{R}_{0,\mathtt{w}}$, but using coinductive types.
- We define (coinductively) ρ by $\rho = [\rho]_\omega \to \rho$.

Coinductive Type Systems

- We consider two new type systems $\mathscr{R}$ and $\mathscr{R}_{\mathtt{w}}$, having the same rules as $\mathscr{R}_0$ and $\mathscr{R}_{0,\mathtt{w}}$, but using coinductive types.
- We define (coinductively) ρ by $\rho = [\rho]_\omega \to \rho$.
- Due to irrelevancy, every term is typable in $\mathscr{R}_{\mathtt{w}}$ (**complete unsoundness of $\mathscr{R}_{\mathtt{w}}$**).

COINDUCTIVE TYPE SYSTEMS

- We consider two new type systems $\mathscr{R}$ and $\mathscr{R}_{\mathtt{w}}$, having the same rules as $\mathscr{R}_0$ and $\mathscr{R}_{0,\mathtt{w}}$, but using coinductive types.
- We define (coinductively) ρ by $\rho = [\rho]_\omega \to \rho$.
- Due to irrelevancy, every term is typable in $\mathscr{R}_{\mathtt{w}}$ (**complete unsoundness of $\mathscr{R}_{\mathtt{w}}$**).
- **Claim:** Let t be a term. If $\Gamma(x) = [\rho]_\omega$ for all free variable x of t, then $\Gamma \vdash t : \rho$ is derivable in $\mathscr{R}_{\mathtt{w}}$.

COINDUCTIVE TYPE SYSTEMS

- We consider two new type systems $\mathscr{R}$ and $\mathscr{R}_{\mathtt{w}}$, having the same rules as $\mathscr{R}_0$ and $\mathscr{R}_{0,\mathtt{w}}$, but using coinductive types.
- We define (coinductively) ρ by $\rho = [\rho]_\omega \to \rho$.
- Due to irrelevancy, every term is typable in $\mathscr{R}_{\mathtt{w}}$ (**complete unsoundness of** $\mathscr{R}_{\mathtt{w}}$**)**.
- **Claim:** Let t be a term. If $\Gamma(x) = [\rho]_\omega$ for all free variable x of t, then $\Gamma \vdash t : \rho$ is derivable in $\mathscr{R}_{\mathtt{w}}$.

  Proof.

$$\frac{\Gamma; x : [\rho]_\omega \vdash t : \rho}{\Gamma \vdash \lambda x.t : [\rho]_\omega \to \rho \quad {\color{red}(= \rho)}}\,\mathtt{abs}$$

$$\frac{\Gamma \vdash t : \rho \quad {\color{red}(= [\rho]_\omega \to \rho)} \qquad (\Gamma \vdash u : \rho)_\omega}{\Gamma \vdash t\,u : \rho}\,\mathtt{app}$$

Relevant coinductive types

- In $\mathscr{R}$ (relevant), $\lambda y.x$ can still be typed only with types of the form $[\,] \to \tau$.

RELEVANT COINDUCTIVE TYPES

- In $\mathscr{R}$ (relevant), $\lambda y.x$ can still be typed only with types of the form $[\,] \to \tau$.
- More generally, if x not free in t and $\triangleright\Gamma \vdash t : \tau$, then $\tau = [\,] \to \tau_0$ for some τ_0.

RELEVANT COINDUCTIVE TYPES

- In $\mathscr{R}$ (relevant), $\lambda y.x$ can still be typed only with types of the form $[\,] \to \tau$.
- More generally, if x not free in t and $\rhd \Gamma \vdash t : \tau$, then $\tau = [\,] \to \tau_0$ for some τ_0.
- In $\mathscr{R}$, the typing rules constrain $[\,]$ to appear.
  Failure of the previous argument.

RELEVANT COINDUCTIVE TYPES

- In $\mathscr{R}$ (relevant), $\lambda y.x$ can still be typed only with types of the form $[\,] \rightarrow \tau$.
- More generally, if x not free in t and $\rhd \Gamma \vdash t : \tau$, then $\tau = [\,] \rightarrow \tau_0$ for some τ_0.
- In $\mathscr{R}$, the typing rules constrain $[\,]$ to appear. Failure of the previous argument.
- **Question:** what is the set of typable terms in $\mathscr{R}$?

Question: what is the set of typable terms in $\mathscr{R}$?

Question: what is the set of typable terms in $\mathscr{R}$?

- *In the finite case:* type Normal Forms and proceed by expansion.

Question: what is the set of typable terms in $\mathscr{R}$?

- *In the finite case:* type Normal Forms and proceed by expansion.
- *Problem for coinductive Types:* no form of normalization is granted (*e.g.* Ω typable in $\mathscr{R}$).

Question: what is the set of typable terms in $\mathscr{R}$?

- *In the finite case:* type Normal Forms and proceed by expansion.
- *Problem for coinductive Types:* no form of normalization is granted (*e.g.* Ω typable in $\mathscr{R}$).

We study then **typability** as a first order theory. For that, we resort to another type system $\mathtt{S}$, in which features *pointers*.

Question: what is the set of typable terms in $\mathscr{R}$?

- *In the finite case:* type Normal Forms and proceed by expansion.
- *Problem for coinductive Types:* no form of normalization is granted (*e.g.* Ω typable in $\mathscr{R}$).

We study then **typability** as a first order theory. For that, we resort to another type system $\mathtt{S}$, in which features *pointers*.
System $\mathtt{S}$ collapses on $\mathscr{R}$. Thus, if every term is typable in $\mathtt{S}$, then every term is typable in $\mathscr{R}$.

Plan

Types and Relevance

Coinductive Types

System S (sequential intersection)

SEQUENTIAL INTERSECTION

- **Types:**

$$S_k, T \ ::= \ o \in \mathscr{O} \mid (S_k)_{k \in K} \to T$$

SEQUENTIAL INTERSECTION

- **Types:**

$$S_k, T \ ::= \ o \in \mathscr{O} \mid (S_k)_{k \in K} \to T$$

- **Sequence Type**:
  - Intersection type replacing multiset types.

SEQUENTIAL INTERSECTION

- **Types:**

$$S_k, T \ ::= \ o \in \mathscr{O} \mid (S_k)_{k \in K} \to T$$

- **Sequence Type**:
  - Intersection type replacing multiset types.
  - $F = (T_k)_{k \in K}$ where T_k types and $K \subset \mathbb{N} - \{0, 1\}$.

SEQUENTIAL INTERSECTION

- **Types:**

$$S_k, T \ ::= \ o \in \mathscr{O} \mid (S_k)_{k \in K} \to T$$

- **Sequence Type**:
  - Intersection type replacing multiset types.
  - $F = (T_k)_{k \in K}$ where T_k types and $K \subset \mathbb{N} - \{0, 1\}$.
  - The integer indexes k are called **tracks**.

Sequential intersection

- **Types:**

$$S_k, T \ ::= \ o \in \mathscr{O} \mid (S_k)_{k \in K} \to T$$

- **Sequence Type**:
  - Intersection type replacing multiset types.
  - $F = (T_k)_{k \in K}$ where T_k types and $K \subset \mathbb{N} - \{0, 1\}$.
  - The integer indexes k are called **tracks**.
  - We also write $(S_k)_{k \in K} = (k \cdot S_k)_{k \in K}$.

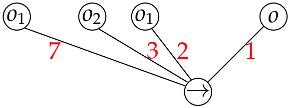
SEQUENTIAL INTERSECTION

- **Types:**

$$S_k, T \ ::= \ o \in \mathscr{O} \mid (S_k)_{k\in K} \to T$$

- **Sequence Type**:
  - Intersection type replacing multiset types.
  - $F = (T_k)_{k\in K}$ where T_k types and $K \subset \mathbb{N} - \{0,1\}$.
  - The integer indexes k are called **tracks**.
  - We also write $(S_k)_{k\in K} = (k \cdot S_k)_{k\in K}$.

- *Example:* $(7 \cdot o_1, 3 \cdot o_2, 2 \cdot o_1) \to o$



DERIVATIONS OF S

The set `Deriv` of rigid derivations is *coinductively* generated by:

$$\frac{}{x : (k \cdot T) \vdash x : T}\ \texttt{ax} \qquad \frac{C;\, x : (S_k)_{k \in K} \vdash t : T}{(S_k)_{k\in K} \vdash \lambda x.t : C(x) \to T}\ \texttt{abs}$$

$$\frac{C \vdash t : (S_k)_{k \in K} \to T \qquad (D_k \vdash u : S_k)_{k \in K}}{C \uplus_{k \in K} D_k \vdash t\,u : T}\ \text{app}$$

DERIVATIONS OF S

The set `Deriv` of rigid derivations is *coinductively* generated by:

$$\frac{}{x : (k \cdot T) \vdash x : T}\ \texttt{ax} \qquad \frac{C;\, x : (S_k)_{k \in K} \vdash t : T}{(S_k)_{k \in K} \vdash \lambda x.t : C(x) \to T}\ \texttt{abs}$$

$$\frac{C \vdash t : (S_k)_{k \in K} \to T \qquad (D_k \vdash u : S_k)_{k \in K}}{C \uplus_{k \in K} D_k \vdash t\, u : T}\ \text{app}$$

- If $\mathrm{Rt}(C)$ and the $\mathrm{Rt}(D_k)$ are not pairwise disjoint, contexts are incompatible.

DERIVATIONS OF S

The set `Deriv` of rigid derivations is *coinductively* generated by:

$$\frac{}{x : (k \cdot T) \vdash x : T}\ \texttt{ax} \qquad \frac{C; x : (S_k)_{k \in K} \vdash t : T}{(S_k)_{k \in K} \vdash \lambda x.t : C(x) \to T}\ \texttt{abs}$$

$$\frac{C \vdash t : (S_k)_{k \in K} \to T \qquad (D_k \vdash u : S_k)_{k \in K}}{C \uplus_{k \in K} D_k \vdash t\, u : T}\ \text{app}$$

- If $\mathrm{Rt}(C)$ and the $\mathrm{Rt}(D_k)$ are not pairwise disjoint, contexts are incompatible.
- Forget about the indexes: S collapses onto $\mathscr{R}$.

DERIVATIONS OF S

The set `Deriv` of rigid derivations is *coinductively* generated by:

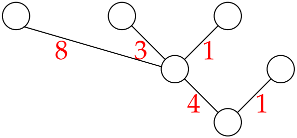
$$\frac{}{x : (k \cdot T) \vdash x : T}\ \texttt{ax} \qquad \frac{C; x : (S_k)_{k \in K} \vdash t : T}{(S_k)_{k \in K} \vdash \lambda x.t : C(x) \to T}\ \texttt{abs}$$

$$\frac{C \vdash t : (S_k)_{k \in K} \to T \qquad (D_k \vdash u : S_k)_{k \in K}}{C \uplus_{k \in K} D_k \vdash t\,u : T}\ \text{app}$$

- If $\mathrm{Rt}(C)$ and the $\mathrm{Rt}(D_k)$ are not pairwise disjoint, contexts are incompatible.
- Forget about the indexes: S collapses onto $\mathscr{R}$.
- S features **pointers** called **bipositions**.

CANDIDATE SUPPORTS

What is a correct type ?

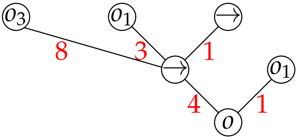


Support:

$\{\varepsilon, 1, 4, 4 \cdot 1, 4 \cdot 3, 4 \cdot 8\}$

CANDIDATE SUPPORTS

What is a correct type ?

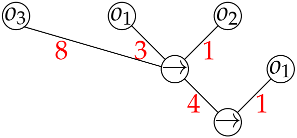


Wrong Labels

Support:
$\{\varepsilon, 1, 4, 4 \cdot 1, 4 \cdot 3, 4 \cdot 8\}$

CANDIDATE SUPPORTS

What is a correct type ?

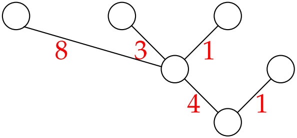


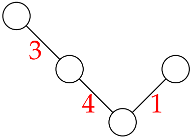
Correct Labels

Support:
$\{\varepsilon, 1, 4, 4 \cdot 1, 4 \cdot 3, 4 \cdot 8\}$

CANDIDATE SUPPORTS

What is a correct type ?



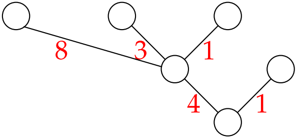


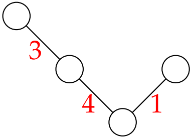
Support:
$\{\varepsilon, 1, 4, 4\cdot 1, 4\cdot 3, 4\cdot 8\}$

Support:
$\{\varepsilon, 1, 4, 4\cdot 3\}$

Candidate Supports

What is a correct type ?





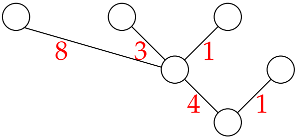
Wrong Support

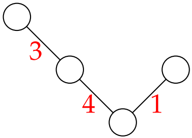
Support:
$\{\varepsilon, 1, 4, 4\cdot 1, 4\cdot 3, 4\cdot 8\}$

Support:
$\{\varepsilon, 1, 4, 4\cdot 3\}$

CANDIDATE SUPPORTS

What is a correct type ?





Support:
$\{\varepsilon, 1, 4, 4\cdot 1, 4\cdot 3, 4\cdot 8\}$

Support:
$\{\varepsilon, 1, 4, 4\cdot 3\}$

Candidate Support: a set of positions that is the support of a type

- $c \rightarrow_{\mathtt{t1}} c \cdot k$ (a candidate supp is a tree)
- $c \cdot 1 \rightarrow_{\mathtt{t2}} c \cdot k$ (if a node does not have a 1-son, it is a leaf)

CANDIDATE BISUPPORTS

- We want to show that every term t is typable in $\mathtt{S}$.

CANDIDATE BISUPPORTS

- We want to show that every term t is typable in $\mathtt{S}$.
- *Idea:* we try to capture the notion of **candidate bisupport**: a set of pointers that is the bisupport of a $\mathtt{S}$-derivation typing t.

CANDIDATE BISUPPORTS

- We want to show that every term t is typable in $\mathtt{S}$.
- *Idea:* we try to capture the notion of **candidate bisupport**: a set of pointers that is the bisupport of a $\mathtt{S}$-derivation typing t.
- We must find suitable stability conditions.

CANDIDATE BISUPPORTS

- We want to show that every term t is typable in $\mathtt{S}$.
- *Idea:* we try to capture the notion of **candidate bisupport**: a set of pointers that is the bisupport of a $\mathtt{S}$-derivation typing t.
- We must find suitable stability conditions.
- Then, we show that there is a *non-empty* set that satisfies them.

CANDIDATE BISUPPORTS

- $(a,\, c) \to_{\mathtt{asc}} (a \cdot 1, 1 \cdot c)$ if $t(a) = @$.
- $(a,\, 1 \cdot c) \to (a \cdot 0,\, c)$ if $t(a) = \lambda x$.
- $(a, k \cdot c) \to_{\mathtt{pi}} (pos(k), c)$ if $t(a) = \lambda x$ and $k \in \mathtt{Tr}_1(a)$.
- $(a, k \cdot c) \to_{\mathtt{pi}} \mathtt{b}_\perp$ if $t(\overline{a}) = \lambda x$ and $k \notin \mathtt{Tr}_1(a)$, $k \geqslant 2$.
- $(a \cdot 1, k \cdot c) \xrightarrow{a} (a \cdot k, c)$ if $t(a) = @$.
- $(a, c) \to_{\mathtt{t1}} (a, c \cdot k)$.
- $(a, c \cdot 1) \to_{\mathtt{t2}} (a, c \cdot k)$ for any $k \geqslant 2$.
- $(a, 1) \to_{\mathtt{rt}} (a, \varepsilon)$ if $t(a) = \lambda x$.
- $(a, \varepsilon) \to_{\mathtt{up}} \mathtt{b}_\perp$.
- $(a, \varepsilon) \to_{\mathtt{up}} (a', c)$ if $a \leqslant a'$

CANDIDATE BISUPPORTS

- $(a, c) \rightarrow_{\texttt{asc}} (a \cdot 1, 1 \cdot c)$ if $t(a) = @$.
- $(a, 1 \cdot c) \rightarrow (a \cdot 0, c)$ if $t(a) = \lambda x$.
- $(a, k \cdot c) \rightarrow_{\texttt{pi}} (pos(k), c)$ if $t(a) = \lambda x$ and $k \in \texttt{Tr}_1(a)$.
- $(a, k \cdot c) \rightarrow_{\texttt{pi}} \texttt{b}_\perp$ if $t(\bar{a}) = \lambda x$ and $k \notin \texttt{Tr}_1(a)$, $k \geqslant 2$.
- $(a \cdot 1, k \cdot c) \xrightarrow{a} (a \cdot k, c)$ if $t(a) = @$.
- $(a, c) \rightarrow_{\texttt{t1}} (a, c \cdot k)$.
- $(a, c \cdot 1) \rightarrow_{\texttt{t2}} (a, c \cdot k)$ for any $k \geqslant 2$.
- $(a, 1) \rightarrow_{\texttt{rt}} (a, \varepsilon)$ if $t(a) = \lambda x$.
- $(a, \varepsilon) \rightarrow_{\texttt{up}} \texttt{b}_\perp$.
- $(a, \varepsilon) \rightarrow_{\texttt{up}} (a', c)$ if $a \leqslant a'$

GUIDELINES OF THE PROOF

Goal: checking that the former conditions cannot prove that the type of t must be empty.
In that case, we can build a derivation whose bisupport is minimal.

GUIDELINES OF THE PROOF

Goal: checking that the former conditions cannot prove that the type of t must be empty.
In that case, we can build a derivation whose bisupport is minimal.

- *Ad absurbum*, we consider $\mathscr{P}$, a proof showing that the type of t is empty.

GUIDELINES OF THE PROOF

Goal: checking that the former conditions cannot prove that the type of t must be empty.
In that case, we can build a derivation whose bisupport is minimal.

- *Ad absurbum*, we consider $\mathscr{P}$, a proof showing that the type of t is empty.
- The presence of redex is still problematic. A finite reduction strategy (the **collapsing strategy**) allows us to reduce $\mathscr{P}$ to a proof $\mathscr{P}'$, in which redexes are not a problem.

GUIDELINES OF THE PROOF

Goal: checking that the former conditions cannot prove that the type of t must be empty.
In that case, we can build a derivation whose bisupport is minimal.

- *Ad absurbum*, we consider $\mathscr{P}$, a proof showing that the type of t is empty.
- The presence of redex is still problematic. A finite reduction strategy (the **collapsing strategy**) allows us to reduce $\mathscr{P}$ to a proof $\mathscr{P}'$, in which redexes are not a problem.
- In $\mathscr{P}'$, commutations and nice interactions occur. Considering a minimal case, we show that $\mathscr{P}'$ *cannot* prove that t has an empty type. *Contradiction*.

GUIDELINES OF THE PROOF

Goal: checking that the former conditions cannot prove that the type of t must be empty.
In that case, we can build a derivation whose bisupport is minimal.

- *Ad absurbum*, we consider $\mathscr{P}$, a proof showing that the type of t is empty.
- The presence of redex is still problematic. A finite reduction strategy (the **collapsing strategy**) allows us to reduce $\mathscr{P}$ to a proof $\mathscr{P}'$, in which redexes are not a problem.
- In $\mathscr{P}'$, commutations and nice interactions occur. Considering a minimal case, we show that $\mathscr{P}'$ *cannot* prove that t has an empty type. *Contradiction*.

This works for the infinitary λ-calculus.

ORDER

Theorem (complete unsoundness): in $\mathscr{R}$, every term is typable.

ORDER

Theorem (complete unsoundness): in $\mathscr{R}$, every term is typable.

Definition: The **order** of a λ-term t is the maximal $n \in \mathbb{N} \cup \{\infty\}$ s.t. $t \to^* t' = \lambda x_1 \ldots \lambda x_n.t'_0$.
A **zero term** is a term of order 0.

ORDER

Theorem (complete unsoundness): in $\mathscr{R}$, every term is typable.

Definition: The **order** of a λ-term t is the maximal $n \in \mathbb{N} \cup \{\infty\}$ s.t. $t \to^* t' = \lambda x_1 \ldots \lambda x_n . t'_0$.
A **zero term** is a term of order 0.

Proposition: if t is a zero-term, then, t is typable with o.

ORDER

Theorem (complete unsoundness): in $\mathscr{R}$, every term is typable.

Definition: The **order** of a λ-term t is the maximal $n \in \mathbb{N} \cup \{\infty\}$ s.t.
$t \rightarrow^* t' = \lambda x_1 \dots \lambda x_n.t'_0$.
A **zero term** is a term of order 0.

Proposition: if t is a zero-term, then, t is typable with o.

Definition (relational model): For all closed λ-term t, we set

$$[\![t]\!] = \{\tau \mid \vdash t : \tau \text{ is derivable}\}$$

ORDER

Theorem (complete unsoundness): in $\mathscr{R}$, every term is typable.

Definition: The **order** of a λ-term t is the maximal $n \in \mathbb{N} \cup \{\infty\}$ s.t. $t \to^* t' = \lambda x_1 \ldots \lambda x_n . t'_0$.
A **zero term** is a term of order 0.

Proposition: if t is a zero-term, then, t is typable with o.

Definition (relational model): For all closed λ-term t, we set

$$[\![t]\!] = \{\tau \mid \vdash t : \tau \text{ is derivable}\}$$

Theorem: This yields a non-sensible model that discriminates terms according to their order.

Related and Future Work

- The collapse of Type System $\mathbb{S}$ on type System $\mathscr{R}$ (Gardner/de Carvalho) is surjective [V,2015].

Related and Future Work

- The collapse of Type System $\mathbb{S}$ on type System $\mathscr{R}$ (Gardner/de Carvalho) is surjective [V,2015].
- Equational theory of the Model.

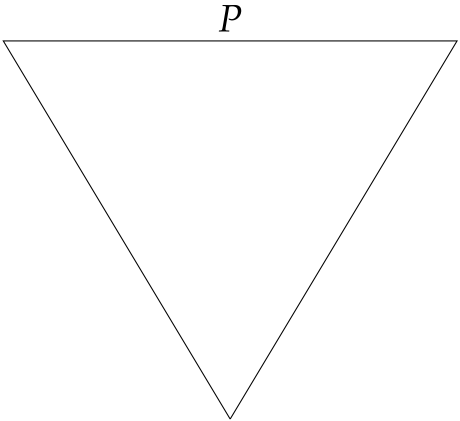
Related and Future Work

- The collapse of Type System $\mathbb{S}$ on type System $\mathscr{R}$ (Gardner/de Carvalho) is surjective [V,2015].
- Equational theory of the Model.
- Is the collapse of $\mathscr{R}$ onto $\mathscr{D}$ (idempotent intersection) surjective ?

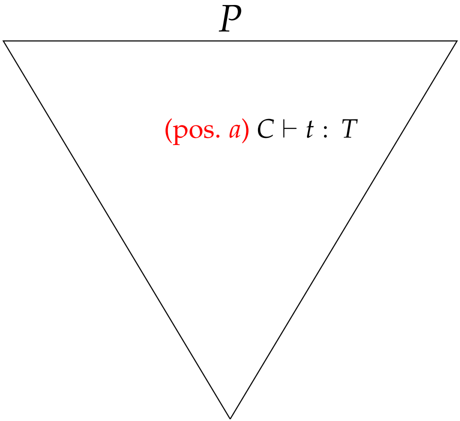
QUESTIONS

Thank you for your attention !

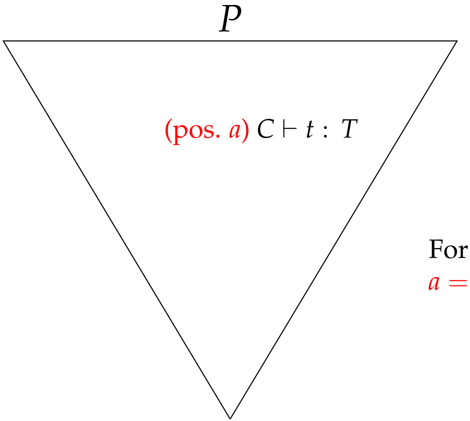
Pointers



Pointers

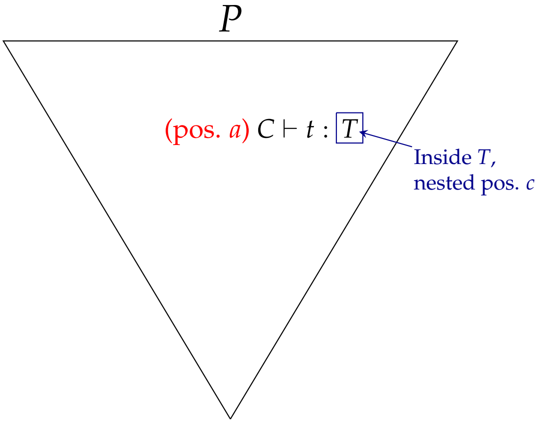


Pointers

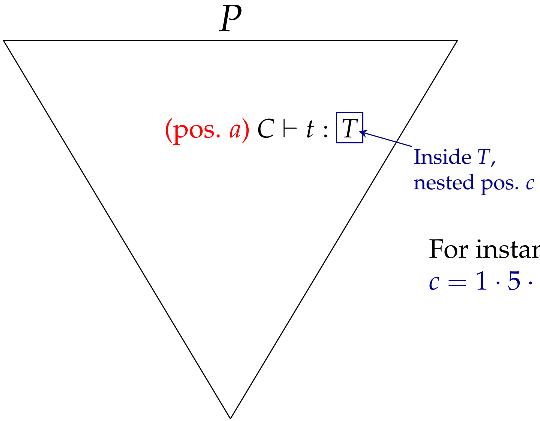


For instance
$a = 0 \cdot 1 \cdot 3 \cdot 0 \cdot 8 \cdot 1$

POINTERS

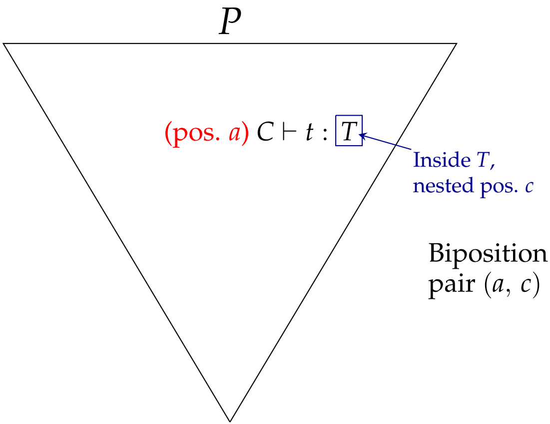


POINTERS



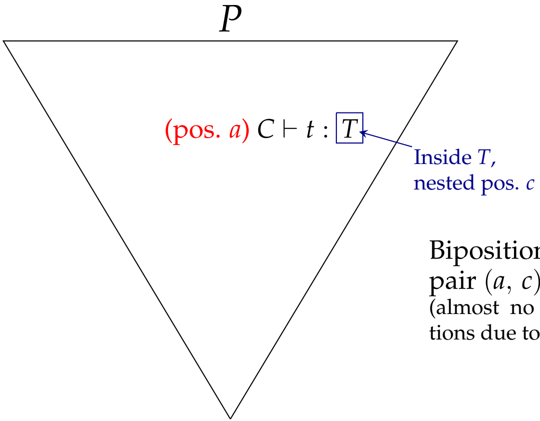
For instance
$c = 1 \cdot 5 \cdot 3 \cdot 1 \cdot 4$

POINTERS



Biposition (right h.s.):
pair (a, c)

POINTERS



Biposition (right h.s.):
pair (a, c)
(almost no need for left bipositions due to relevance)

Bisupport of P: the set of (right or left) bipositions

Ascendance

Some bipositions can be intuitively identified in a derivation.

Ascendance

Some bipositions can be intuitively identified in a derivation.

$$\dfrac{C \vdash t : (S_k)_{k \in K} \to T \quad \text{(pos. } a \cdot 1) \qquad (D_k \vdash u : S_k \ \text{(pos. } a \cdot k) \ )_{k \in K}}{C \cup_{k \in K} D_k \vdash tu : T \quad \text{(pos. } a)}$$

ASCENDANCE

Some bipositions can be intuitively identified in a derivation.

$$\frac{C \vdash t : (S_k)_{k \in K} \to T \quad (\text{pos. } a \cdot 1) \qquad (D_k \vdash u : S_k \ (\text{pos. } a \cdot k)\ )_{k \in K}}{C \cup_{k \in K} D_k \vdash tu : T \quad (\text{pos. } a)}$$

Ascendance

Some bipositions can be intuitively identified in a derivation.

$$\frac{C \vdash t : (S_k)_{k\in K} \to T \quad \text{(pos. } a \cdot 1\text{)} \qquad (D_k \vdash u :\ S_k \text{ (pos. } a \cdot k\text{)} \ )_{k\in K}}{C \cup_{k\in K} D_k \vdash tu : T \quad \text{(pos. } a\text{)}}$$

Two occurrences of the same type

ASCENDANCE

Some bipositions can be intuitively identified in a derivation.

$$\frac{C \vdash t : (S_k)_{k\in K} \to T \quad (\text{pos. } a \cdot 1) \qquad (D_k \vdash u :\ S_k\ (\text{pos. } a \cdot k)\ )_{k\in K}}{C \cup_{k\in K} D_k \vdash tu : T \quad (\text{pos. } a)}$$

Ascendance

Some bipositions can be intuitively identified in a derivation.

$$\frac{C \vdash t : (S_k)_{k\in K} \to T \quad \text{(pos. } a \cdot 1\text{)} \qquad (D_k \vdash u :\ S_k \ \text{(pos. } a \cdot k\text{)}\ )_{k\in K}}{C \cup_{k\in K} D_k \vdash tu : T \quad \text{(pos. } a\text{)}}$$

Nested position c here corresponds to...

ASCENDANCE

Some bipositions can be intuitively identified in a derivation.

nested position $1 \cdot c$ there.

$$\frac{C \vdash t : (S_k)_{k \in K} \to T \quad \text{(pos. } a \cdot 1\text{)} \qquad (D_k \vdash u : S_k \ \text{(pos. } a \cdot k\text{)})_{k \in K}}{C \cup_{k \in K} D_k \vdash tu : T \quad \text{(pos. } a\text{)}}$$

Nested position c here corresponds to…

ASCENDANCE

Some bipositions can be intuitively identified in a derivation.

nested position $1 \cdot c$ there.

$$\frac{C \vdash t : (S_k)_{k \in K} \to T \quad \text{(pos. } a \cdot 1\text{)} \qquad (D_k \vdash u : \ S_k \ \text{(pos. } a \cdot k\text{)} \ )_{k \in K}}{C \cup_{k \in K} D_k \vdash tu : T \quad \text{(pos. } a\text{)}}$$

Nested position c here corresponds to…

We then set:
$(a,\ c) \to_{\mathtt{asc}} (a \cdot 1, 1 \cdot c)$ when $t(a) = @$

Ascendance

Some bipositions can be intuitively identified in a derivation.

ASCENDANCE

Some bipositions can be intuitively identified in a derivation.

$$\frac{C;\ x : (S_k)_{k \in K} \vdash t : T \qquad \text{(pos. } a \cdot 0\text{)}}{C \vdash \lambda x.t : (S_k)_{k \in K} \to T \qquad \text{(pos. } a\text{)}}$$

Ascendance

Some bipositions can be intuitively identified in a derivation.

$$\frac{C;\, x : (S_k)_{k\in K} \vdash t : T \qquad \text{(pos. } a \cdot 0\text{)}}{C \vdash \lambda x.t : (S_k)_{k\in K} \to T \qquad \text{(pos. } a\text{)}}$$

We then set:
$(a,\ 1 \cdot c) \to_{\mathtt{asc}} (a \cdot 0, 1 \cdot c)$ when $t(a) = \lambda x$

POLAR INVERSION

Let us remind rules `ax` and `abs`:

$$\frac{}{x : (k \cdot T) \vdash x : T}\ \texttt{ax} \qquad\qquad \frac{C \vdash t : T}{C; (S_k)_{k \in K} \vdash \lambda x.t : C(x) \to T}\ \texttt{abs}$$

POLAR INVERSION

Let us remind rules `ax` and `abs`:

$$\frac{}{x : (k \cdot T) \vdash x : T}\ \texttt{ax} \qquad \frac{C \vdash t : T}{C; (S_k)_{k \in K} \vdash \lambda x.t : C(x) \to T}\ \texttt{abs}$$

In a derivation:

$$\frac{C; x : (S_k)_{k \in K} \vdash t : T \quad \text{(pos. } a \cdot 0\text{)}}{C \vdash \lambda x.t : (S_k)_{k \in K} \to T \quad \text{(pos. } a\text{)}}$$

Polar inversion

Let us remind rules `ax` and `abs`:

$$\frac{}{x : (k \cdot T) \vdash x : T}\ \texttt{ax} \qquad\qquad \frac{C \vdash t : T}{C; (S_k)_{k \in K} \vdash \lambda x.t : C(x) \to T}\ \texttt{abs}$$

Let $k \geqslant 2$. We have two cases :

$$\frac{C; x : (S_k)_{k \in K} \vdash t : T \qquad \text{(pos. } a \cdot 0\text{)}}{C \vdash \lambda x.t : (S_k)_{k \in K} \to T \qquad \text{(pos. } a\text{)}}$$

Look at S_7
inside this seq. type.

POLAR INVERSION

Let us remind rules `ax` and `abs`:

$$\frac{}{x : (k \cdot T) \vdash x : T}\ \texttt{ax} \qquad \frac{C \vdash t : T}{C; (S_k)_{k \in K} \vdash \lambda x.t : C(x) \to T}\ \texttt{abs}$$

Let $k \geqslant 2$. We have two cases :

- First case :

$$\frac{}{x : 7 \cdot S_7 \vdash x : S_7 \text{ (pos. } a')}$$

$$\frac{C; x : (S_k)_{k \in K} \vdash t : T \quad \text{(pos. } a \cdot 0)}{C \vdash \lambda x.t : (S_k)_{k \in K} \to T \quad \text{(pos. } a)}$$

Look at S_7
inside this seq. type.

POLAR INVERSION

Let us remind rules ax and abs:

$$\frac{}{x:\ (k\cdot T)\vdash x:\ T}\ \mathtt{ax} \qquad \frac{C\vdash t:\ T}{C;(S_k)_{k\in K}\vdash \lambda x.t:\ C(x)\to T}\ \mathtt{abs}$$

Let $k\geqslant 2$. We have two cases :

- First case :

$$\frac{}{x:\ 7\cdot S_7\vdash x:\ S_7\ \text{(pos. } a'\text{)}}$$

$$\frac{C;\ x:(S_k)_{k\in K}\vdash t:T \quad \text{(pos. } a\cdot 0\text{)}}{C\vdash \lambda x.t:(S_k)_{k\in K}\to T \quad \text{(pos. } a\text{)}}$$

Look at S_7
inside this seq. type.

We then set: $(a,7\cdot c)\to_{\mathtt{pi}}(a',c)$ when $t(a)=\lambda x$

POLAR INVERSION

Let us remind rules `ax` and `abs`:

$$\frac{}{x : (k \cdot T) \vdash x : T}\ \texttt{ax} \qquad \frac{C \vdash t : T}{C; (S_k)_{k \in K} \vdash \lambda x.t : C(x) \to T}\ \texttt{abs}$$

Let $k \geqslant 2$. We have two cases :

Second case :

$$\frac{C; x : (S_k)_{k \in K} \vdash t : T \quad \textcolor{red}{(\text{pos. } a \cdot 0)}}{C \vdash \lambda x.t : (S_k)_{k \in K} \to T \quad \textcolor{red}{(\text{pos. } a)}}$$

Look at S_7
inside this seq. type.

POLAR INVERSION

Let us remind rules `ax` and `abs`:

$$\frac{}{x : (k \cdot T) \vdash x : T}\ \texttt{ax} \qquad \frac{C \vdash t : T}{C; (S_k)_{k \in K} \vdash \lambda x.t : C(x) \to T}\ \texttt{abs}$$

Let $k \geqslant 2$. We have two cases :

Second case :

No `ax`-rule typing x with track 7.

$$\frac{C; x : (S_k)_{k \in K} \vdash t : T \qquad (\text{pos. } a \cdot 0)}{C \vdash \lambda x.t : (S_k)_{k \in K} \to T \qquad (\text{pos. } a)}$$

Look at S_7
inside this seq. type.

POLAR INVERSION

Let us remind rules ax and abs:

$$\frac{}{x : (k \cdot T) \vdash x : T}\ \texttt{ax} \qquad \frac{C \vdash t : T}{C; (S_k)_{k \in K} \vdash \lambda x.t : C(x) \to T}\ \texttt{abs}$$

Let $k \geqslant 2$. We have two cases :

No ax-rule typing x with track 7.

$$\frac{C; x : (S_k)_{k \in K} \vdash t : T \quad (\text{pos. } a \cdot 0)}{C \vdash \lambda x.t : (S_k)_{k \in K} \to T \quad (\text{pos. } a)}$$

Look at S_7
inside this seq. type.

We then set: $(a, 7 \cdot c) \to_{\texttt{pi}} \texttt{b}_\perp$ when $t(a) = \lambda x$

REFERENTS

- Let $\equiv$ be the reflexive, transitive, symmetric closure of $\rightarrow_{\mathtt{asc}} \cup \rightarrow_{\mathtt{pi}}$.

REFERENTS

- Let $\equiv$ be the reflexive, transitive, symmetric closure of $\to_{\mathtt{asc}} \cup \to_{\mathtt{pi}}$.
- Assume $\mathtt{b}_1 \equiv \mathtt{b}_2$.
  Then a derivation P typing t holds $\mathtt{b}_1$ iff it holds $\mathtt{b}_2$.

REFERENTS

- Let $\equiv$ be the reflexive, transitive, symmetric closure of $\rightarrow_{\texttt{asc}} \cup \rightarrow_{\texttt{pi}}$.
- Assume $\texttt{b}_1 \equiv \texttt{b}_2$.
  Then a derivation P typing t holds $\texttt{b}_1$ iff it holds $\texttt{b}_2$.
- Moreover, P cannot hold $\texttt{b}_\perp$.

Referents

- Let $\equiv$ be the reflexive, transitive, symmetric closure of $\rightarrow_{\mathtt{asc}} \cup \rightarrow_{\mathtt{pi}}$.
- Assume $\mathtt{b}_1 \equiv \mathtt{b}_2$.
  Then a derivation P typing t holds $\mathtt{b}_1$ iff it holds $\mathtt{b}_2$.
- Moreover, P cannot hold $\mathtt{b}_\perp$.
- An equivalence class of $\equiv$ is called a **referent**.
  Let $\mathtt{Ref}$ be the quotient set defined by $\equiv$.
  We write $\mathtt{b} : \mathtt{r}$ or $\mathtt{r} : \mathtt{b}$.

Consumption

CONSUMPTION

$$\frac{C \vdash t : (S_k)_{k \in K} \to T \quad (\text{pos. } a \cdot 1) \qquad (D_k \vdash u : S_k \ (\text{pos. } a \cdot k)\ )_{k \in K}}{C \cup_{k \in K} D_k \vdash tu : T \quad (\text{pos. } a)}$$

CONSUMPTION

$$\frac{C \vdash t : (S_k)_{k \in K} \to T \quad \text{(pos. } a \cdot 1) \qquad (D_k \vdash u : S_k \text{ (pos. } a \cdot k))_{k \in K}}{C \cup_{k \in K} D_k \vdash tu : T \quad \text{(pos. } a)}$$

We then set:
$(a \cdot 1,\ k \cdot c) \xrightarrow{a} (a \cdot k,\ c)$ when $t(a) = @$