Non-Idempotent Typing Operators, beyond the λ-Calculus

Soutenance de thèse

Pierre VIAL

IRIF (Univ. Paris Diderot and CNRS)

December 7, 2017

```
x = 1
while (x > 0):
    x = x + 1
transfer(1 000 000 000 $, calyon, my-account)
print("I'm rich now")
```

```
x = 1
while (x > 0):
    x = x + 1
transfer(1 000 000 000 $, calyon, my-account)
print("I'm rich now")
```

$$\boxed{x = 1}$$

```
x = 1
while (x > 0):
    x = x + 1
transfer(1 000 000 000 $, calyon, my-account)
print("I'm rich now")
```

$$\boxed{x = 2}$$

```
x = 1
while (x > 0):
    x = x + 1
transfer(1 000 000 000 $, calyon, my-account)
print("I'm rich now")
```

$$\boxed{x = 3}$$

```
x = 1
while (x > 0):
    x = x + 1
transfer(1 000 000 000 $, calyon, my-account)
print("I'm rich now")
```

$$\boxed{x = 4}$$

```
x = 1
while (x > 0):
    x = x + 1
transfer(1 000 000 000 $, calyon, my-account)
print("I'm rich now")
```

$$\boxed{x = \ \dots}$$

```
x = 1
while (x > 0):
    x = x + 1
transfer(1 000 000 000 $, calyon, my-account)
print("I'm rich now")
```

$$\boxed{x = 100}$$

Certification and logic in computer science

```
x = 1
while (x > 0):
    x = x + 1
transfer(1 000 000 000 $, calyon, my-account)
print("I'm rich now")
```

$$\boxed{x = \ \ldots}$$

```
x = 1
while (x > 0):
    x = x + 1
transfer(1 000 000 000 $, calyon, my-account)
print("I’m rich now")
```

$$\boxed{x = \quad}$$

```
x = 1
while (x > 0):
    x = x + 1
transfer(1 000 000 000 $, calyon, my-account)
print("I'm rich now")
```

$\boxed{x =\quad}$

The core of this thesis

- **Termination** or **productivity** (*via* source codes)
- **Paths** to terminal states.
- For that, using **types** (data descriptors).

```
x = 1
while (x > 0):
    x = x + 1
transfer(1 000 000 000 $, calyon, my-account)
print("I'm rich now")
```

$x =$

The core of this thesis

- **Termination** or **productivity** (*via* source codes)
- **Paths** to terminal states.
- For that, using **types** (data descriptors).

Productivity:

- O. S.
- 2, 3, 5, 7,. . . (primes)

```
x = 1
while (x > 0):
    x = x + 1
transfer(1 000 000 000 $, calyon, my-account)
print("I'm rich now")
```

$$\boxed{x = \quad}$$

The core of this thesis

- **Termination** or **productivity** (*via* source codes)
- **Paths** to terminal states.
- For that, using **types** (data descriptors).

Productivity:

- O. S.
- 2, 3, 5, 7,. . . (primes)

Backtracking:

$\simeq$ Classical logic.

Formal logic (valar morghulis)

All men are mortal. Socrates is a man. Therefore, Socrates is mortal.

Formal logic (valar morghulis)

All men are mortal. Socrates is a man. Therefore, Socrates is mortal.

All ringtus are delgo. Vinkri is a ringtu. Therefore, Vinkri is delgo.

All men are mortal. Socrates is a man. Therefore, Socrates is mortal.

All ringtus are delgo. Vinkri is a ringtu. Therefore, Vinkri is delgo.

All men are mortal. Socrates is a man. Therefore, Socrates is mortal.

All ringtus are delgo. Vinkri is a ringtu. Therefore, Vinkri is delgo.

$$\dfrac{\dfrac{\overline{\forall x,\ \mathscr{H}(x) \Rightarrow \mathscr{M}(x)}}{\mathscr{H}(\mathtt{S}) \Rightarrow \mathscr{M}(\mathtt{S})} \qquad \overline{\mathscr{H}(\mathtt{S})}}{\mathscr{M}(\mathtt{S})}$$

Formal logic (valar morghulis)

All men are mortal. Socrates is a man. Therefore, Socrates is mortal.

All ringtus are delgo. Vinkri is a ringtu. Therefore, Vinkri is delgo.

$$\dfrac{\dfrac{\overline{\forall x,\ \mathscr{H}(x) \Rightarrow \mathscr{M}(x)}}{\mathscr{H}(\mathtt{S}) \Rightarrow \mathscr{M}(\mathtt{S})} \qquad \overline{\mathscr{H}(\mathtt{S})}}{\mathscr{M}(\mathtt{S})}$$

All men are mortal. Socrates is a man. Therefore, Socrates is mortal.

All ringtus are delgo. Vinkri is a ringtu. Therefore, Vinkri is delgo.

$$\dfrac{\dfrac{\overline{\forall x,\ \mathscr{H}(x) \Rightarrow \mathscr{M}(x)}}{\mathscr{H}(\mathtt{S}) \Rightarrow \mathscr{M}(\mathtt{S})} \qquad \overline{\mathscr{H}(\mathtt{S})}}{\mathscr{M}(\mathtt{S})}$$

All men are mortal. Socrates is a man. Therefore, Socrates is mortal.

All ringtus are delgo. Vinkri is a ringtu. Therefore, Vinkri is delgo.

$$\dfrac{\dfrac{\overline{\forall x,\ \mathscr{H}(x) \Rightarrow \mathscr{M}(x)}}{\mathscr{H}(\mathtt{S}) \Rightarrow \mathscr{M}(\mathtt{S})} \qquad \overline{\mathscr{H}(\mathtt{S})}}{\mathscr{M}(\mathtt{S})}$$

Formal logic (valar morghulis)

All men are mortal. Socrates is a man. Therefore, Socrates is mortal.

All ringtus are delgo. Vinkri is a ringtu. Therefore, Vinkri is delgo.

$$\dfrac{\dfrac{\overline{\forall x,\ \mathscr{H}(x) \Rightarrow \mathscr{M}(x)}}{\mathscr{H}(\mathtt{S}) \Rightarrow \mathscr{M}(\mathtt{S})} \qquad \overline{\mathscr{H}(\mathtt{S})}}{\mathscr{M}(\mathtt{S})}$$

Formalization

Reduce semantic (= meaning) to mechanical/grammatical/syntactic rules.

Turing and computability

Entscheidung (1928): given a symbolic statement, is there an *algorithmic* procedure to *decide* whether it is *true* or not?

Entscheidung (1928): given a symbolic statement, is there an *algorithmic* procedure to *decide* whether it is *true* or not?

Gödel, 1931:
$\exists$ *unprovable* statements

Entscheidung (1928): given a symbolic statement, is there an *algorithmic* procedure to *decide* whether it is *true* or not?

Gödel, 1931: $\exists$ *unprovable* statements $\longrightarrow$ Provable $\neq$ True

Entscheidung (1928): given a symbolic statement, is there an *algorithmic* procedure to *decide* whether it is *true* or not?

Gödel, 1931:
$\exists$ *unprovable* statements

$\rightarrow$ Provable $\neq$ True

primitive recursive functions (poor)

Entscheidung (1928): given a symbolic statement, is there an *algorithmic* procedure to *decide* whether it is *true* or not?

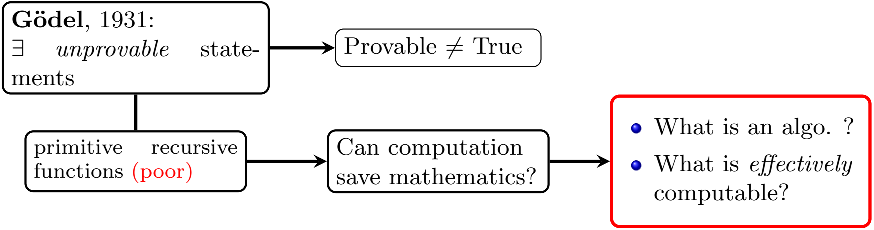
Gödel, 1931: $\exists$ *unprovable* statements

Provable $\neq$ True

primitive recursive functions (poor)

Can computation save mathematics?

Entscheidung (1928): given a symbolic statement, is there an *algorithmic* procedure to *decide* whether it is *true* or not?



Entscheidung (1928): given a symbolic statement, is there an *algorithmic* procedure to *decide* whether it is *true* or not?

Gödel, 1931: $\exists$ *unprovable* statements → Provable $\neq$ True

primitive recursive functions (poor) → Can computation save mathematics? →

- What is an algo. ?
- What is *effectively* computable?

Turing machines (1936)

TM are universal

f *effectively computable*

iff f *implementable in a TM*

$\rightsquigarrow$ A prog. language $\mathcal{L}$ is **Turing-complete**

if $\mathcal{L}$ has the same computational power as TMs.

Entscheidung (1928): given a symbolic statement, is there an *algorithmic* procedure to *decide* whether it is *true* or not?

Turing and computability

Entscheidung (1928): given a symbolic statement, is there an *algorithmic* procedure to *decide* whether it is *true* or not?

Theorem (Turing, 1936)

- *The Entscheidungsproblem has a negative answer*
- *The* ***halting problem*** *is undecidable: there does not exist a general method deciding whether any program terminates or not.*

The λ-calculus

- One primitive.
- Functional paradigm.
- Turing complete.

Allows to emulate many rewriting systems *e.g.*:

The λ-calculus

- One primitive.
- Functional paradigm.
- Turing complete.

Allows to emulate many rewriting systems *e.g.*:

Example (implementing natural numbers)

$\mathtt{0}$: zero $\mathtt{S}$: successor

Thus: $\mathtt{S\,0} \simeq 1$ $\mathtt{S\,S\,0} \simeq 2$ $\mathtt{S\,S\,S\,S\,S\,0} \simeq 5.$

The λ-calculus

- One primitive.
- Functional paradigm.
- Turing complete.

Allows to emulate many rewriting systems *e.g.*:

Example (implementing natural numbers)

$\mathtt{0}$: zero $\qquad$ $\mathtt{S}$: successor

Thus: $\mathtt{S}\,\mathtt{0} \simeq 1$ $\qquad$ $\mathtt{S}\,\mathtt{S}\,\mathtt{0} \simeq 2$ $\qquad$ $\mathtt{S}\,\mathtt{S}\,\mathtt{S}\,\mathtt{S}\,\mathtt{S}\,\mathtt{0} \simeq 5.$

Addition

$n + \mathtt{0} \to n$ (terminal case) $\qquad$ $n\ + \mathtt{S}\,m \to \mathtt{S}\,n + m$ (inductive case)

The λ-calculus

- One primitive.
- Functional paradigm.
- Turing complete.

Allows to emulate many rewriting systems *e.g.*:

Example (implementing natural numbers)

$\mathtt{0}$: zero $\qquad\qquad$ $\mathtt{S}$: successor

Thus: $\quad \mathtt{S\,0} \simeq 1 \qquad \mathtt{S\,S\,0} \simeq 2 \qquad \mathtt{S\,S\,S\,S\,S\,0} \simeq 5.$

Addition

$n + \mathtt{0} \to n$ (terminal case) $\qquad\qquad$ $n + \mathtt{S}\, m \to \mathtt{S}\, n + m$ (inductive case)

The λ-calculus

- One primitive.
- Functional paradigm.
- Turing complete.

Allows to emulate many rewriting systems *e.g.*:

Example (implementing natural numbers)

$\mathtt{0}$: zero $\qquad$ $\mathtt{S}$: successor

Thus: $\quad \mathtt{S}\,\mathtt{0} \simeq 1 \qquad \mathtt{S}\,\mathtt{S}\,\mathtt{0} \simeq 2 \qquad \mathtt{S}\,\mathtt{S}\,\mathtt{S}\,\mathtt{S}\,\mathtt{S}\,\mathtt{0} \simeq 5.$

Addition

$n + \mathtt{0} \to n$ (terminal case) $\qquad$ $n + \mathtt{S}\,m \to \mathtt{S}\,n + m$ (inductive case)

$$\begin{array}{ccccccc}
\mathtt{S\,S\,S\,0} + \mathtt{S\,S\,0} & \to & \mathtt{S\,S\,S\,S\,0} + \mathtt{S\,0} & \to & \mathtt{S\,S\,S\,S\,S\,0} + \mathtt{0} & \to & \mathtt{S\,S\,S\,S\,S\,0} \\
3 + 2 & \to & 4 + 1 & \to & 5 + 0 & \to & 5
\end{array}$$

Computation as rewriting

The λ-calculus

- One primitive.
- Functional paradigm.
- Turing complete.

Allows to emulate many rewriting systems *e.g.*:

Example (implementing natural numbers)

$\mathtt{0}$: zero $\qquad$ $\mathtt{S}$: successor

Thus: $\quad \mathtt{S\,0} \simeq 1 \qquad \mathtt{S\,S\,0} \simeq 2 \qquad \mathtt{S\,S\,S\,S\,S\,0} \simeq 5.$

Addition

$n + \mathtt{0} \to n$ (terminal case) $\qquad$ $n + \mathtt{S}\,m \to \mathtt{S}\,n + m$ (inductive case)

$$\begin{array}{ccccccc} \mathtt{S\,S\,S\,0+S\,S\,0} & \to & \mathtt{S\,S\,S\,S\,0+S\,0} & \to & \mathtt{S\,S\,S\,S\,S\,0+0} & \to & \mathtt{S\,S\,S\,S\,S\,0} \\ 3+2 & \to & 4+1 & \to & 5+0 & \to & 5 \end{array}$$

The λ-calculus

- One primitive.
- Functional paradigm.
- Turing complete.

Allows to emulate many rewriting systems *e.g.*:

Example (implementing natural numbers)

$\mathtt{0}$: zero $\qquad$ $\mathtt{S}$: successor

Thus: $\quad \mathtt{S\,0} \simeq 1 \quad \mathtt{S\,S\,0} \simeq 2 \quad \mathtt{S\,S\,S\,S\,S\,0} \simeq 5.$

Addition

$n + \mathtt{0} \to n$ (terminal case) $\qquad$ $n + \mathtt{S}\,m \to \mathtt{S}\,n + m$ (inductive case)

$$\begin{array}{ccccccc} \mathtt{S\,S\,S\,0} + \mathtt{S\,S\,0} & \to & \mathtt{S\,S\,S\,S\,0} + \mathtt{S\,0} & \to & \mathtt{S\,S\,S\,S\,S\,0} + \mathtt{0} & \to & \mathtt{S\,S\,S\,S\,S\,0} \\ 3+2 & \to & 4+1 & \to & 5+0 & \to & 5 \end{array}$$

The λ-calculus

- One primitive.
- Functional paradigm.
- Turing complete.

Allows to emulate many rewriting systems *e.g.*:

Example (implementing natural numbers)

$\mathtt{0}$: zero $\qquad$ $\mathtt{S}$: successor

Thus: $\quad \mathtt{S}\,\mathtt{0} \simeq 1 \qquad \mathtt{S}\,\mathtt{S}\,\mathtt{0} \simeq 2 \qquad \mathtt{S}\,\mathtt{S}\,\mathtt{S}\,\mathtt{S}\,\mathtt{S}\,\mathtt{0} \simeq 5.$

Addition

$n + \mathtt{0} \to n$ (terminal case) $\qquad$ $n + \mathtt{S}\,m \to \mathtt{S}\,n + m$ (inductive case)

$$\begin{array}{ccccccc} \mathtt{S\,S\,S\,0} + \mathtt{S\,S\,0} & \to & \mathtt{S\,S\,S\,S\,0} + \mathtt{S\,0} & \to & \mathtt{S\,S\,S\,S\,S\,0} + \mathtt{0} & \to & \mathtt{S\,S\,S\,S\,S\,0} \\ 3+2 & \to & 4+1 & \to & 5+0 & \to & 5 \end{array}$$

The λ-calculus

- One primitive.
- Functional paradigm.
- Turing complete.

Allows to emulate many rewriting systems *e.g.*:

Example (implementing natural numbers)

$\mathtt{0}$: zero $\qquad$ $\mathtt{S}$: successor

Thus: $\quad \mathtt{S\,0} \simeq 1 \qquad \mathtt{S\,S\,0} \simeq 2 \qquad \mathtt{S\,S\,S\,S\,S\,0} \simeq 5.$

Addition

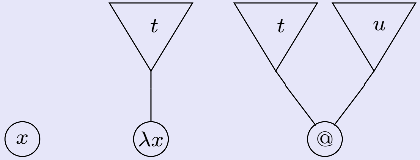
$n + \mathtt{0} \to n$ (terminal case) $\qquad$ $n + \mathtt{S}\,m \to \mathtt{S}\,n + m$ (inductive case)

$$\mathtt{S\,S\,S\,0} + \mathtt{S\,S\,0} \to \mathtt{S\,S\,S\,S\,0} + \mathtt{S\,0} \to \mathtt{S\,S\,S\,S\,S\,0} + \mathtt{0} \to \mathtt{S\,S\,S\,S\,S\,0}$$

- Most structures (tabs, strings, pair of integers) can be implemented in this fashion or in the λ-calculus.

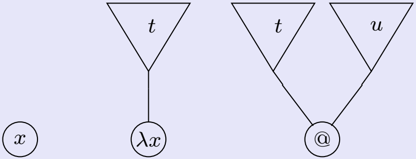
λ-calcul (Church, 1928)

Term construction (inductive grammar)

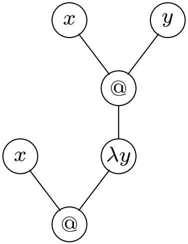


Variable	Abstraction	Application
x	$\lambda x.t$	$t\,u$

Term construction (inductive grammar)

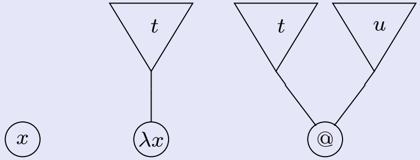


x	$\lambda x.t$	$t\,u$
Variable	**Abstraction**	**Application**



Example: $x(\lambda y.x\,y)$

Term construction (inductive grammar)

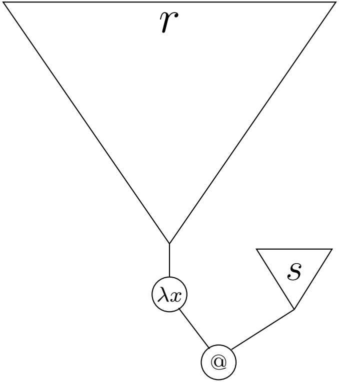


x	$\lambda x.t$	$t\,u$
Variable	**Abstraction**	**Application**

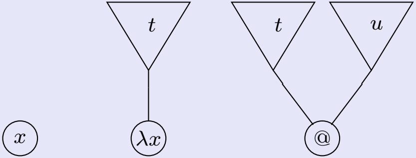
Redex (reducible expression):
$\leadsto$ computation *via* **substitution** producing a **reduct**

Redex:

$(\lambda x.r)s$



Term construction (inductive grammar)



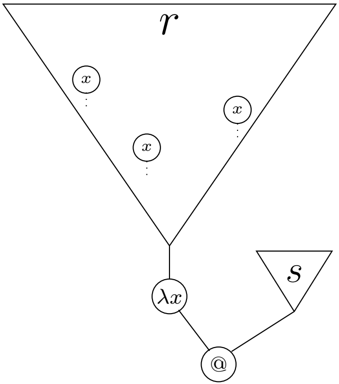
x	$\lambda x.t$	$t\,u$
Variable	**Abstraction**	**Application**

Redex (reducible expression):

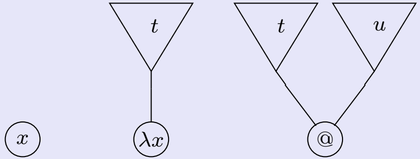
$\rightsquigarrow$ computation *via* **substitution** producing a **reduct**

Redex:

$$(\lambda x.r)s$$



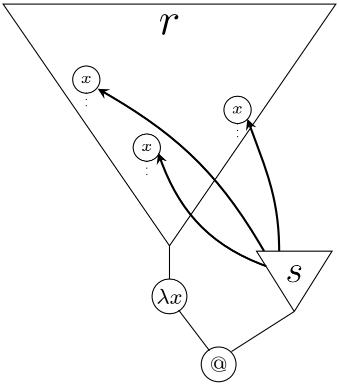
Term construction (inductive grammar)



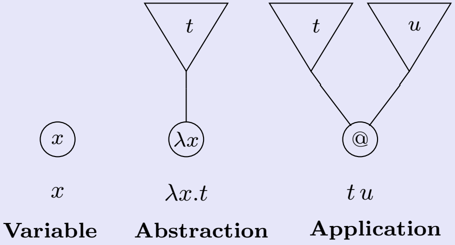
x	$\lambda x.t$	$t\,u$
Variable	**Abstraction**	**Application**

Redex (reducible expression):
$\rightsquigarrow$ computation *via* **substitution** producing a **reduct**

λ-calcul (Church, 1928)



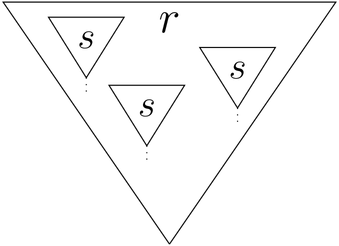
Term construction (inductive grammar)



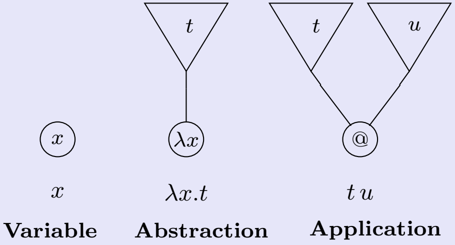
Redex (reducible expression):
$\rightsquigarrow$ computation *via* **substitution** producing a **reduct**

Reduct:

$r[s/x]$



Term construction (inductive grammar)



Redex (reducible expression):

$\rightsquigarrow$ computation *via* **substitution** producing a **reduct**

Higher-order functions and their (possible) dangers

- Let $\mathtt{app}_2(\mathtt{f}, \mathtt{x}) := \mathtt{f}(\mathtt{f}(\mathtt{x}))$.
  - $\mathtt{app}_2$ takes a function $\mathtt{f}$ as an argument.
  - $\mathtt{app}_2$ is a **higher-order function**.

Higher-order functions and their (possible) dangers

- Let $\mathtt{app}_2(\mathtt{f}, \mathtt{x}) := \mathtt{f}(\mathtt{f}(\mathtt{x}))$.
  - $\mathtt{app}_2$ takes a function $\mathtt{f}$ as an argument.
  - $\mathtt{app}_2$ is a **higher-order function**.

- Autoapplication is defined by:

$$\mathtt{autoapp}(\mathtt{f}) \to \mathtt{f}(\mathtt{f})$$

Higher-order functions and their (possible) dangers

- Let $\mathtt{app}_2(\mathtt{f}, \mathtt{x}) := \mathtt{f}(\mathtt{f}(\mathtt{x}))$.
  - $\mathtt{app}_2$ takes a function $\mathtt{f}$ as an argument.
  - $\mathtt{app}_2$ is a **higher-order function**.
- Autoapplication is defined by:

$$\mathtt{autoapp}(\mathtt{f}) \to \mathtt{f}(\mathtt{f})$$

Higher-order functions and their (possible) dangers

- Let $\mathtt{app}_2(\mathtt{f}, \mathtt{x}) := \mathtt{f}(\mathtt{f}(\mathtt{x}))$.
  - $\mathtt{app}_2$ takes a function $\mathtt{f}$ as an argument.
  - $\mathtt{app}_2$ is a **higher-order function**.
- Autoapplication is defined by:

$$\mathtt{autoapp}(\mathtt{f}) \to \mathtt{f}(\mathtt{f})$$

- Auto-autoapplication:

$$\mathtt{autoapp}(\mathtt{autoapp}) \to$$

Higher-order functions and their (possible) dangers

- Let $\texttt{app}_2(\texttt{f}, \texttt{x}) := \texttt{f}(\texttt{f}(\texttt{x}))$.
  - $\texttt{app}_2$ takes a function $\texttt{f}$ as an argument.
  - $\texttt{app}_2$ is a **higher-order function**.
- Autoapplication is defined by:

$$\texttt{autoapp}(\texttt{f}) \to \texttt{f}(\texttt{f})$$

- Auto-autoapplication:

$$\texttt{autoapp}(\texttt{autoapp}) \to \texttt{autoapp}(\texttt{autoapp})$$

Higher-order functions and their (possible) dangers

- Let $\mathtt{app}_2(\mathtt{f}, \mathtt{x}) := \mathtt{f}(\mathtt{f}(\mathtt{x}))$.
  - $\mathtt{app}_2$ takes a function $\mathtt{f}$ as an argument.
  - $\mathtt{app}_2$ is a **higher-order function**.
- Autoapplication is defined by:

$$\mathtt{autoapp}(\mathtt{f}) \to \mathtt{f}(\mathtt{f})$$

- Auto-autoapplication:

$$\mathtt{autoapp}(\mathtt{autoapp}) \to \mathtt{autoapp}(\mathtt{autoapp}) \to \mathtt{autoapp}(\mathtt{autoapp})$$

Higher-order functions and their (possible) dangers

- Let $\mathtt{app}_2(\mathtt{f}, \mathtt{x}) := \mathtt{f}(\mathtt{f}(\mathtt{x}))$.
  - $\mathtt{app}_2$ takes a function $\mathtt{f}$ as an argument.
  - $\mathtt{app}_2$ is a **higher-order function**.
- Autoapplication is defined by:

$$\mathtt{autoapp}(\mathtt{f}) \to \mathtt{f}(\mathtt{f})$$

- Auto-autoapplication:

$$\mathtt{autoapp}(\mathtt{autoapp}) \to \mathtt{autoapp}(\mathtt{autoapp}) \to \mathtt{autoapp}(\mathtt{autoapp})$$
$$\to \mathtt{autoapp}(\mathtt{autoapp})$$

- Let $\mathtt{app}_2(\mathtt{f},\mathtt{x}) := \mathtt{f}(\mathtt{f}(\mathtt{x}))$.
  - $\mathtt{app}_2$ takes a function $\mathtt{f}$ as an argument.
  - $\mathtt{app}_2$ is a **higher-order function**.
- Autoapplication is defined by:

$$\mathtt{autoapp}(\mathtt{f}) \to \mathtt{f}(\mathtt{f})$$

- Auto-autoapplication:

$$\mathtt{autoapp}(\mathtt{autoapp}) \to \mathtt{autoapp}(\mathtt{autoapp}) \to \mathtt{autoapp}(\mathtt{autoapp})$$
$$\to \mathtt{autoapp}(\mathtt{autoapp}) \to \ldots\ldots\ldots\ldots\ldots\ldots\ldots$$

Higher-order functions and their (possible) dangers

- Let $\texttt{app}_2(\texttt{f}, \texttt{x}) := \texttt{f}(\texttt{f}(\texttt{x}))$.
  - $\texttt{app}_2$ takes a function $\texttt{f}$ as an argument.
  - $\texttt{app}_2$ is a **higher-order function**.
- Autoapplication is defined by:

$$\texttt{autoapp}(\texttt{f}) \to \texttt{f}(\texttt{f})$$

- Auto-autoapplication:

$$\texttt{autoapp}(\texttt{autoapp}) \to \texttt{autoapp}(\texttt{autoapp}) \to \texttt{autoapp}(\texttt{autoapp}) \\ \to \texttt{autoapp}(\texttt{autoapp}) \to \ldots\ldots\ldots\ldots\ldots\ldots\ldots$$

Remember

- Some programs that do not terminate are still meaningful: the **streams**.
- Keep on **producing** terminated values.

Example: The program printing 2, 3, 5, 7, 11, 13. . . (the list of primes).

Higher-order functions and their (possible) dangers

- Let $\mathtt{app}_2(\mathtt{f}, \mathtt{x}) := \mathtt{f}(\mathtt{f}(\mathtt{x}))$.
  - $\mathtt{app}_2$ takes a function $\mathtt{f}$ as an argument.
  - $\mathtt{app}_2$ is a **higher-order function**.
- Autoapplication is defined by:

$$\mathtt{autoapp}(\mathtt{f}) \to \mathtt{f}(\mathtt{f})$$

- Auto-autoapplication:

$$\mathtt{autoapp}(\mathtt{autoapp}) \to \mathtt{autoapp}(\mathtt{autoapp}) \to \mathtt{autoapp}(\mathtt{autoapp})$$
$$\to \mathtt{autoapp}(\mathtt{autoapp}) \to \dots\dots\dots\dots\dots\dots$$

Remember

- Some programs that do not terminate are still meaningful: the **streams**.
- Keep on **producing** terminated values.

Example: The program printing 2, 3, 5, 7, 11, 13. . . (the list of primes).

Contribution:
characterizing productive streams.

Terminal states and execution/reduction strategies

$$\underbrace{2+3\times 5}_{} \quad \longrightarrow \quad \underbrace{2+15}_{} \quad \longrightarrow \quad 17$$

Reducible (non-terminal) states

Terminal state

Terminal states and execution/reduction strategies

$$\underbrace{2+3\times 5}_{} \longrightarrow \underbrace{2+15}_{} \longrightarrow 17$$

Reducible (non-terminal) states

Terminal state

- Let $f(x) = x \times x \times x$. What is the value of $f(3+4)$?

$$\underbrace{2 + 3 \times 5} \quad \longrightarrow \quad \underbrace{2 + 15} \quad \longrightarrow \quad 17$$

Reducible (non-terminal) states

Terminal state

- Let $f(x) = x \times x \times x$. What is the value of $f(3 + 4)$?

Kim (smart)

$$\begin{aligned} f(3+4) &\rightarrow f(7) \\ &\rightarrow 7 \times 7 \times 7 \\ &\rightarrow 49 \times 7 \\ &\rightarrow 343 \end{aligned}$$

Lee (not so)

$$\begin{aligned} f(3+4) &\rightarrow (3+4) \times (3+4) \times (3+4) \\ &\rightarrow 7 \times (3+4) \times (3+4) \\ &\rightarrow 7 \times 7 \times (3+4) \\ &\rightarrow 7 \times 7 \times 7 \\ &\rightarrow 49 \times 7 \\ &\rightarrow 343 \end{aligned}$$

Thurston (don't be Thurston)

$$\begin{aligned} f(3+4) &\rightarrow (3+4) \times (3+4) \times (3+4) \\ &\rightarrow 3 \times (3+4) \times (3+4) + 4 \times (3+4) \times (3+4) \\ &\rightarrow \text{dozens of computation steps} \\ &\dots \quad \dots\dots\dots\dots\dots\dots \\ &\rightarrow 343 \end{aligned}$$

Terminal states and execution/reduction strategies

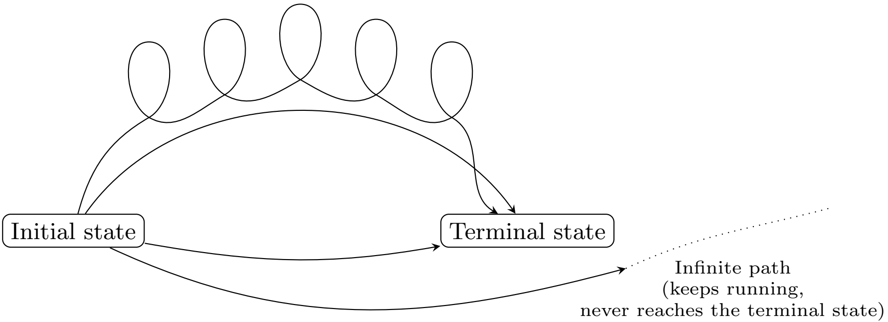
$$\underbrace{2 + 3 \times 5}_{} \quad \longrightarrow \quad \underbrace{2 + 15}_{} \quad \longrightarrow \quad 17$$

Reducible (non-terminal) states

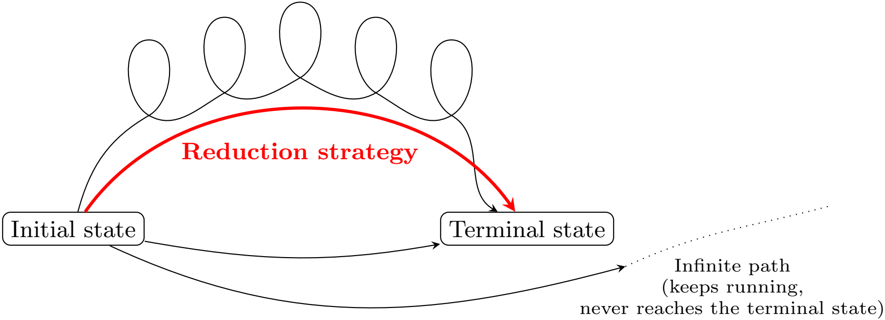
Terminal state

Terminal states and execution/reduction strategies

Terminal states and execution/reduction strategies



Terminal states and execution/reduction strategies



Reduction strategy

- **Choice** of a reduction path.
- Can be **complete**
- Must be **certified**.

Principle

- Types = data **descriptors**, following a **grammar**.
- Types provide certifications of **correction**.

Types

Principle

- Types = data **descriptors**, following a **grammar**.
- Types provide certifications of **correction**.

Primitive types:

5: `int` (integer)

"Leopard": `String` (string of characters)

Principle

- Types = data **descriptors**, following a **grammar**.
- Types provide certifications of **correction**.

Primitive types:

5: `int` (integer)

"Leopard": `String` (string of characters)

Compound types:

`length` : `String` $\rightarrow$ `int` (function)

Types

Principle

- Types = data **descriptors**, following a **grammar**.
- Types provide certifications of **correction**.

Types

Principle

- Types = data **descriptors**, following a **grammar**.
- Types provide certifications of **correction**.

Example

Let `toLetters` : `int` $\rightarrow$ `String` be the program:

$$\texttt{toLetters}(2) = \text{"two"} \qquad \texttt{toLetters}(10) = \text{"ten"}$$

Types

Principle

- Types = data **descriptors**, following a **grammar**.
- Types provide certifications of **correction**.

Example

Let $\texttt{toLetters} : \texttt{int} \to \texttt{String}$ be the program:

$\texttt{toLetters}(2) = \text{"two"}$ $\qquad$ $\texttt{toLetters}(10) = \text{"ten"}$

$\texttt{toLetters}(5)$ $\qquad$ $\texttt{toLetters}(\text{"Leopard"})$

Types

Principle

- Types = data **descriptors**, following a **grammar**.
- Types provide certifications of **correction**.

Example

Let $\texttt{toLetters} : \texttt{int} \rightarrow \texttt{String}$ be the program:

$$\texttt{toLetters}(2) = \text{"two"} \qquad \texttt{toLetters}(10) = \text{"ten"}$$

$\texttt{toLetters}(5)$

Correct!
Returns "five"

$\texttt{toLetters}(\text{"Leopard"})$

Incorrect!
The arg. "Leopard" is not an int.

Types

`toLetters`(5)

Correct!
Returns "five"

`toLetters`("Leopard")

Incorrect!
The arg. "Leopard" is not an int.

$$\frac{\texttt{toLetters} : \texttt{int} \to \texttt{String} \qquad 5 : \texttt{int}}{\texttt{toLetters}(5) : \texttt{String}}$$

Typing certificate

$\texttt{toLetters}(5)$

Correct!
Returns "five"

$\texttt{toLetters}(\text{"Leopard"})$

Incorrect!
The arg. "Leopard" is not an int.

Types

$$\frac{\texttt{toLetters} : \texttt{int} \to \texttt{String} \qquad 5 : \texttt{int}}{\texttt{toLetters}(5) : \texttt{String}}$$

Typing certificate

$$\frac{A \to B \qquad A}{B}$$

Proof

$\texttt{toLetters}(5)$

Correct!
Returns "five"

$\texttt{toLetters}(\text{"Leopard"})$

Incorrect!
The arg. "Leopard" is not an int.

Types

$$\frac{\texttt{int} \to \texttt{String} \qquad \texttt{int}}{\texttt{String}}$$

Typing certificate

$$\frac{A \to B \qquad A}{B}$$

Proof

`toLetters`(5)

`toLetters`("Leopard")

Correct!
Returns "five"

Incorrect!
The arg. "Leopard" is not an int.

$$\frac{\texttt{toLetters} : \texttt{int} \to \texttt{String} \qquad 5 : \texttt{int}}{\texttt{toLetters}(5) : \texttt{String}} \qquad\qquad \frac{A \to B \qquad A}{B}$$

Typing certificate **Proof**

$\boxed{\textbf{This analogy goes further!}}$

$$\frac{\texttt{int} \to \texttt{String} \qquad \texttt{int}}{\texttt{String}} \qquad\qquad \frac{A \to B \qquad A}{B}$$

Typing certificate **Proof**

This analogy goes further!

Curry-Howard correspondence!

Curry-Howard (50s)

Programming languages	**Logic**
Type	Formula
Simply Typed Program	Proof
Reduction Step	Cut-Elimination Step
Termination	Termination

$$\frac{\texttt{toLetters} : \texttt{int} \to \texttt{String} \qquad 5 : \texttt{int}}{\texttt{toLetters}(5) : \texttt{String}} \qquad\qquad \frac{A \to B \qquad A}{B}$$

Curry-Howard (50s)

Programming languages	**Logic**
Type	Formula
Simply Typed Program	Proof
Reduction Step	Cut-Elimination Step
Termination	Termination

Simple types

Curry-Howard (50s)

Programming languages	**Logic**
Type	Formula
Simply Typed Program	Proof
Reduction Step	Cut-Elimination Step
Termination	Termination

Simple types

- Harness higher-order comput. in a limited way.
- Many progs. in terminal state not typable.

Curry-Howard (50s)

Programming languages	**Logic**
Type	Formula
Simply Typed Program	Proof
Reduction Step	Cut-Elimination Step
Termination	Termination

Simple types

- Harness higher-order comput. in a limited way.
- Many progs. in terminal state not typable.

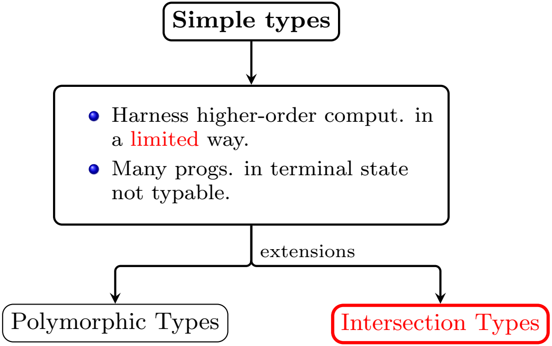
extensions

Polymorphic Types

Intersection Types

Curry-Howard (50s)

Programming languages	**Logic**
Type	Formula
Simply Typed Program	Proof
Reduction Step	Cut-Elimination Step
Termination	Termination



Curry-Howard (50s)

Programming languages	**Logic**
Type	Formula
Simply Typed Program	Proof
Reduction Step	Cut-Elimination Step
Termination	Termination

Simple types

λ-calculus

- Harness higher-order comput. in a limited way.
- Many progs. in terminal state not typable.

extensions

Polymorphic Types

Intersection Types

Curry-Howard (50s)

Programming languages	**Logic**
Type	Formula
Simply Typed Program	Proof
Reduction Step	Cut-Elimination Step
Termination	Termination

Simple types

- Harness higher-order comput. in a limited way.
- Many progs. in terminal state not typable.

extensions

Polymorphic Types

Intersection Types

λ-calculus

Does not capture classical logic

Programming languages	**Logic**
Type	Formula
Simply Typed Program	Proof
Reduction Step	Cut-Elimination Step
Termination	Termination

Simple types

- Harness higher-order comput. in a limited way.
- Many progs. in terminal state not typable.

extensions

Polymorphic Types

Intersection Types

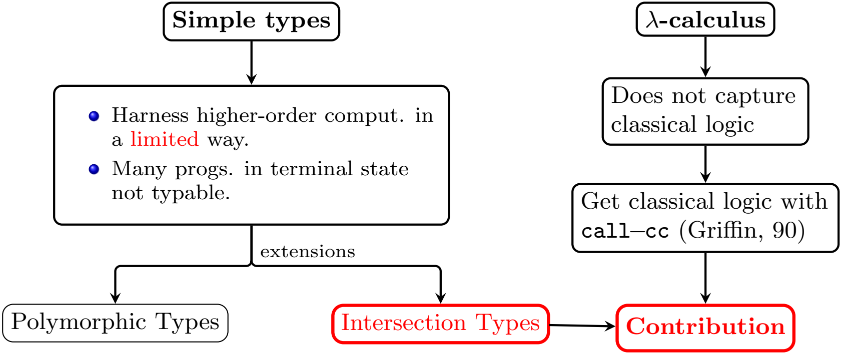
λ-calculus

Does not capture classical logic

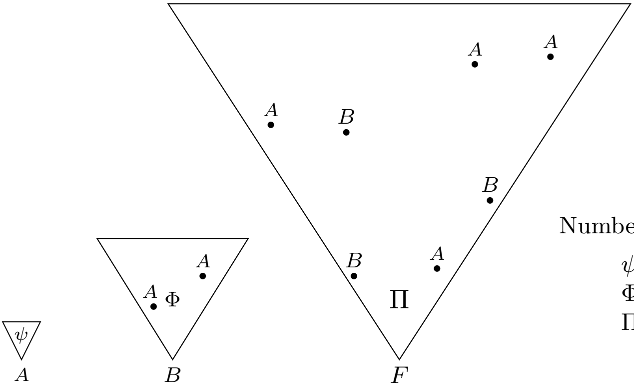
Get classical logic with `call–cc` (Griffin, 90)

CURRY-HOWARD (50S)

Programming languages	**Logic**
Type	Formula
Simply Typed Program	Proof
Reduction Step	Cut-Elimination Step
Termination	Termination



Cut-Elimination (animation)

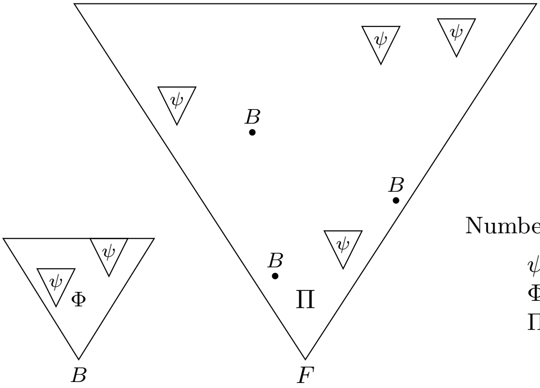


Number of occurrences:

ψ	:	1
Φ	:	1
Π	:	1

Initial proof of F (using two lemmas)

Goal: having a one-block proof



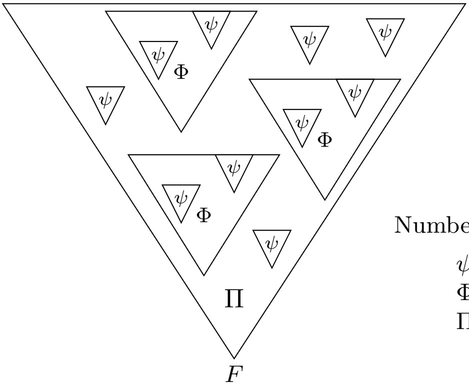
Number of occurrences:

ψ	:	6
Φ	:	1
Π	:	1

After one cut-elim. step (one lemma)

$\boxed{\text{Goal: having a one-block proof}}$

Cut-Elimination (animation)

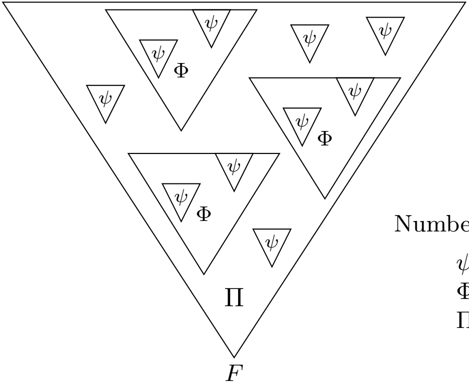


Number of occurrences:

ψ	:	10
Φ	:	3
Π	:	1

Goal: having a one-block proof

Cut-Elimination (animation)

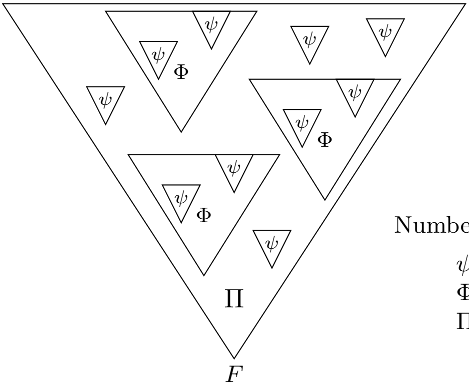


Number of occurrences:

ψ	:	10
Φ	:	3
Π	:	1

After two cut-elim. steps

Cut-Elimination (animation)



Number of occurrences:

ψ	:	10
Φ	:	3
Π	:	1

Theorem (Gentzen, 1936, Prawitz, 1965)

The cut-elimination procedure terminates (and tells us a lot of things).

Intersections types (Coppo, Dezani, 1980)

Goal

Equivalences of the form

> *"the program t is typable iff it can reach a terminal state"*

Idea: **several** certificates to a same subprogram.

Intersections types (Coppo, Dezani, 1980)

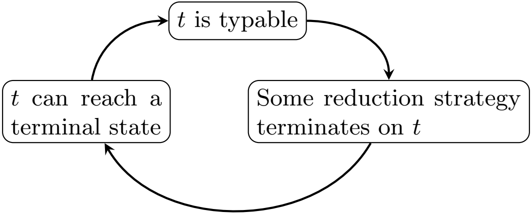
Goal

Equivalences of the form

"the program t is typable iff it can reach a terminal state"

Idea: **several** certificates to a same subprogram.

Proof: by the "circular" implications:



Intersections types (Coppo, Dezani, 1980)

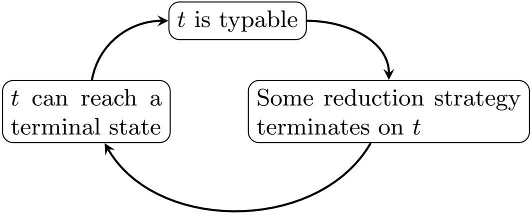
Goal

Equivalences of the form

> *"the program t is typable iff it can reach a terminal state"*

Idea: **several** certificates to a same subprogram.

Proof: by the "circular" implications:



Intersection types

- Perhaps too expressive. . .
- . . . but certify reduction strategies!

Non-idempotency

Computation causes duplication.

Non-idempotency

Computation causes duplication.

Non-idempotent intersection types

Disallow duplication for typing certificates.

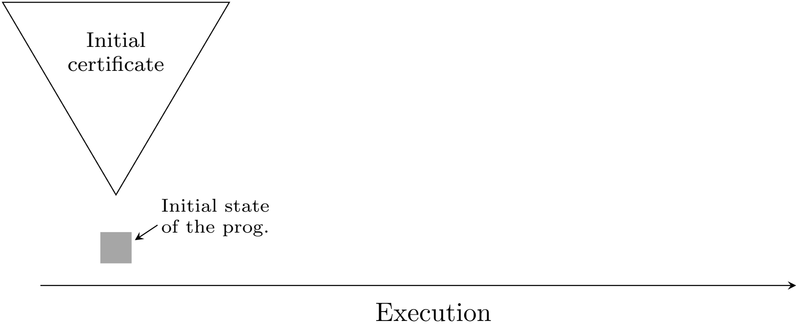
- $\rightsquigarrow$ Possibly many certificates for a subprogram.
- $\rightsquigarrow$ Size of certificates decreases.

Computation causes duplication.

Non-idempotent intersection types

Disallow duplication for typing certificates.

- $\rightsquigarrow$ Possibly many certificates for a subprogram.
- $\rightsquigarrow$ Size of certificates decreases.



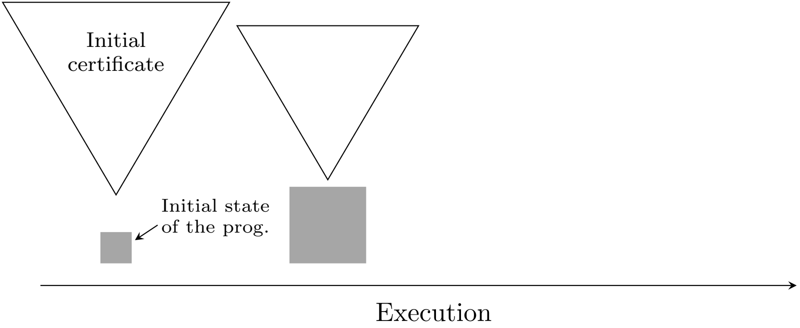
Non-idempotency

Computation causes duplication.

Non-idempotent intersection types

Disallow duplication for typing certificates.

- ⇝ Possibly many certificates for a subprogram.
- ⇝ Size of certificates decreases.



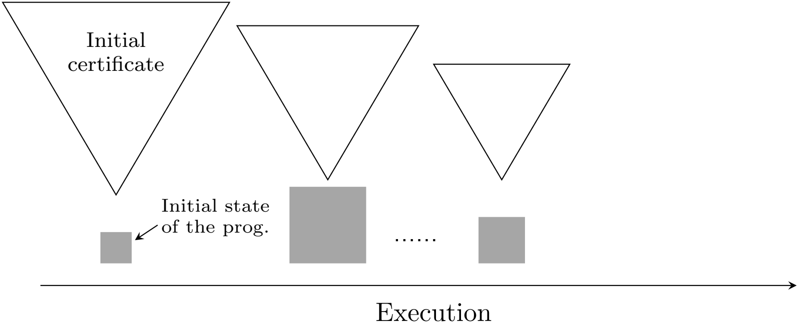
Non-idempotency

Computation causes duplication.

Non-idempotent intersection types

Disallow duplication for typing certificates.

- ⇝ Possibly many certificates for a subprogram.
- ⇝ Size of certificates decreases.



Non-idempotency

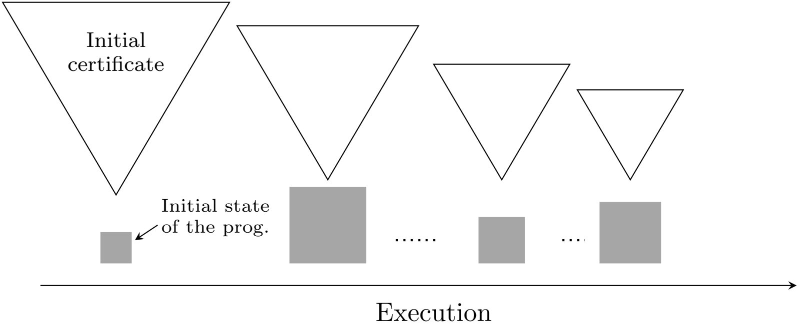
Computation causes duplication.

Non-idempotent intersection types

Disallow duplication for typing certificates.

⇝ Possibly many certificates for a subprogram.

⇝ Size of certificates decreases.



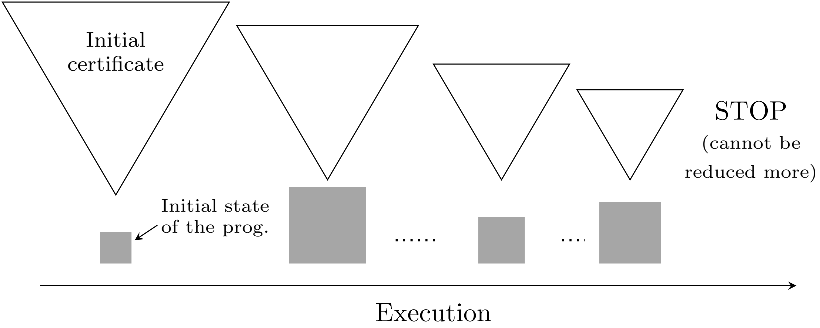
Non-idempotency

Computation causes duplication.

Non-idempotent intersection types

Disallow duplication for typing certificates.

- ⇝ Possibly many certificates for a subprogram.
- ⇝ Size of certificates decreases.



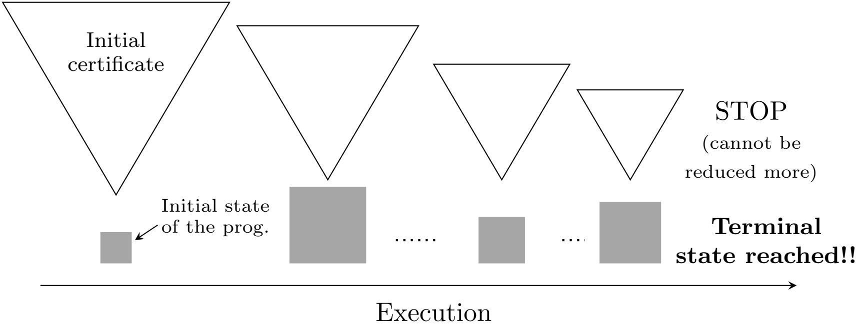
Non-idempotency

Computation causes duplication.

Non-idempotent intersection types

Disallow duplication for typing certificates.

- $\rightsquigarrow$ Possibly many certificates for a subprogram.
- $\rightsquigarrow$ Size of certificates decreases.



Computation causes duplication.

Non-idempotent intersection types

Disallow duplication for typing certificates.

$\rightsquigarrow$ Possibly many certificates for a subprogram.

$\rightsquigarrow$ Size of certificates decreases.

Computation causes duplication.

Non-idempotent intersection types

Disallow duplication for typing certificates.

$\rightsquigarrow$ Possibly many certificates for a subprogram.

$\rightsquigarrow$ Size of certificates decreases.

Comparative (dis)advantages

- Insanely difficult to type a particular program.
- Whole type system **easier** to study!
  - Easier proofs of **termination**!
  - Easier proofs of **characterization**!
  - Easier to certify a **reduction strategy**!

Contents

- Gardner/de Caravalho's non-idempotent type system.

Contribution 1:

- Quantitative types for the $\lambda\mu$-calculus (a *classical* calculus)
- Certificates of reduction strategies.

Contribution 2:

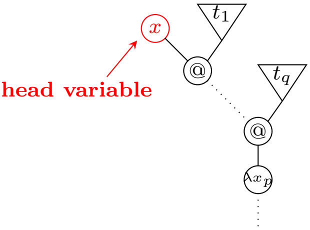
- Positive answer to **Klop's Problem**.
- Certification of an *infinitary* reduction strategy. Introduction of a new type system: system $\mathtt{S}$ (standing for **sequences**).

Contribution 3:

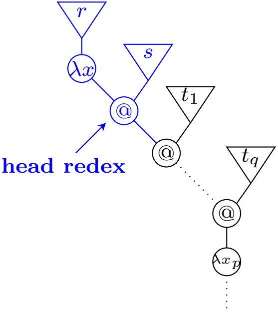
- Around the expressive power of unconstrained infinitary intersection types.

Plan

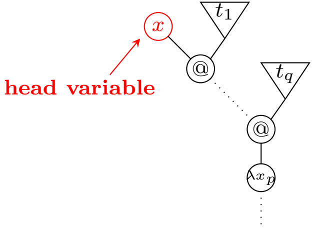
Head Normalization (λ)



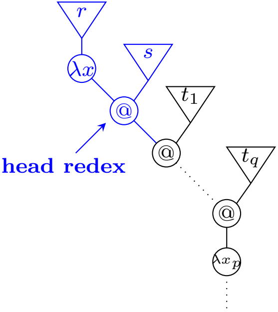
Head Normal Form



Head Reducible Term



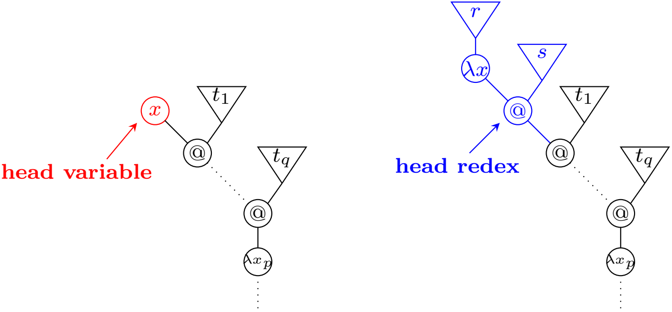
Head Normal Form



Head Reducible Term

- t is **head normalizing (HN)** if $\exists$ reduction path from t to a HNF.

Head Normalization (λ)



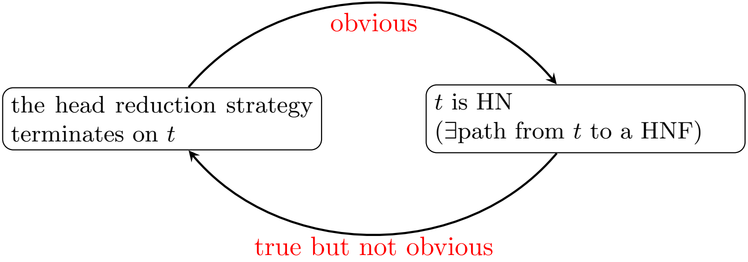
Head Normal Form **Head Reducible Term**

- t is **head normalizing (HN)** if $\exists$ reduction path from t to a HNF.
- The **head reduction strategy**: reducing head redexes while it is possible.

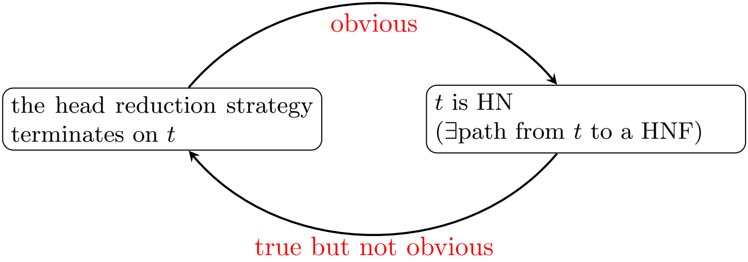
Head Normalization (λ)

- t is **head normalizing (HN)** if $\exists$ reduction path from t to a HNF.
- The **head reduction strategy**: reducing head redexes while it is possible.

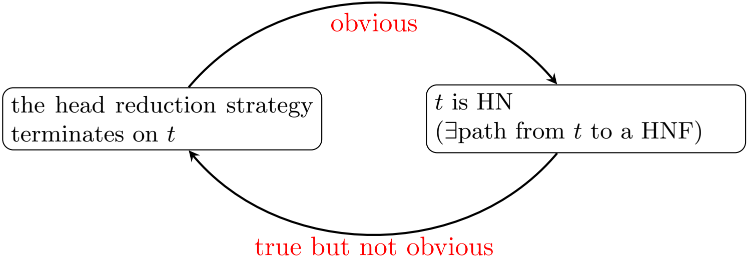
Head Normalization (λ)



- t is **head normalizing (HN)** if $\exists$ reduction path from t to a HNF.
- The **head reduction strategy**: reducing head redexes while it is possible.



- The **head reduction strategy**: reducing head redexes while it is possible.



Intersection types come to help!

- The **head reduction strategy**: reducing head redexes while it is possible.

Subject Reduction and Subject Expansion

A good intersection type system should enjoy:

Subject Reduction (SR):
Typing is stable under reduction.

Subject Expansion (SE):
Typing is stable under anti-reduction.

SE is usually not verified by simple or polymorphic type systems

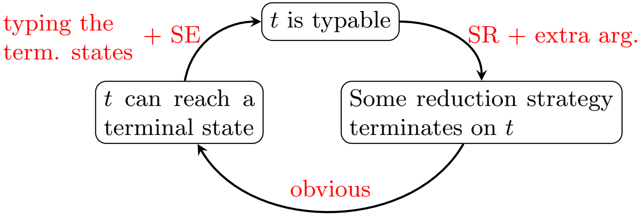
Subject Reduction and Subject Expansion

A good intersection type system should enjoy:

Subject Reduction (SR): Typing is stable under reduction.

Subject Expansion (SE): Typing is stable under anti-reduction.

SE is usually not verified by simple or polymorphic type systems



From Intersection Types to Quantitative Types

Types are built by means of base types, arrow ($\to$) and intersection ($\wedge$).

$$\text{ACI Axioms} = \begin{cases} \mathit{Associativity} & (A \wedge D) \wedge C & \sim & A \wedge (D \wedge C) \\ \mathit{Commutativity} & A \wedge D & \sim & D \wedge A \\ \mathit{Idempotence} & A \wedge A & \sim & A \end{cases}$$

From Intersection Types to Quantitative Types

Types are built by means of base types, arrow ($\rightarrow$) and intersection ($\wedge$).

$$\text{ACI Axioms} = \begin{cases} \textit{Associativity} & (A \wedge D) \wedge C & \sim & A \wedge (D \wedge C) \\ \textit{Commutativity} & A \wedge D & \sim & D \wedge A \\ \textit{Idempotence} & A \wedge A & \sim & A \end{cases}$$

Traditional Intersection Types **Coppo & Dezani 80**	Quantitative Types **Gardner 94 - Kfoury 96**
ACI (Idempotent)	AC (Non-idempotent)
Types are sets: $A \wedge A \wedge C$ is $\{A, C\}$	Types are multisets: $A \wedge A \wedge C$ is $[A, A, C]$
Qualitative properties	Quantitative properties

Remark (non-idem. case):

- $[A, A, C] \neq [A, C]$ *i.e.* $A \wedge A \wedge C \nsim A \wedge C$.
- $[A, B] + [A] = [A, A, B]$ *i.e.* $\wedge$ is multiset sum.

$$
\begin{array}{lcll}
\textbf{(Strict Types)} & \tau, \sigma & := & o \in \mathscr{O} \mid \mathcal{I} \to \tau \\
\textbf{(Intersection Types)} & \mathcal{I} & := & [\sigma_i]_{i \in I}
\end{array}
$$

Strict types $\rightsquigarrow$ **syntax directed rules**:

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

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

System $\mathscr{R}_0$

Remark

- **Relevant** system (no weakening)
- In `app`-rule, pointwise multiset sum *e.g.*,

$$
(x : [\sigma]; y : [\tau]) + (x : [\sigma, \tau]) = x : [\sigma, \sigma, \tau]; y : [\tau]
$$

Properties ($\mathscr{R}_0$)

- **Weighted Subject Reduction**
  - Reduction preserves types and environments, and...
  - ... *head* reduction strictly decreases the nodes of the deriv. tree.
- **Subject Expansion**
  - Anti-reduction preserves types and environments.

Theorem (de Carvalho)

Let t be a λ-term. Then equivalence between:

1. *t is typable (in $\mathscr{R}_0$)*
2. *t is HN*
3. *the head reduction strategy terminates on t ($\rightsquigarrow$ certification!)*

Bonus (quantitative information)

If Π types t, then $\texttt{size}\Pi$ bounds the number of **steps** of the head. red. strategy on t.

Head vs Weak and Strong Normalization

Let t be a λ-term.

- **Head normalization (HN):** there is a path from t to a head normal form.

Head vs Weak and Strong Normalization

Let t be a λ-term.

- **Head normalization (HN):** there is a path from t to a head normal form.

- **Weak normalization (WN):** there is *at least one path* from t to **normal form (NF)**.

Let t be a λ-term.

- **Head normalization (HN):** there is a path from t to a head normal form.
- **Weak normalization (WN):** there is *at least one path* from t to **normal form (NF)**.
- **Strong normalization (SN):** there is *no infinite path* starting at t.

Head vs Weak and Strong Normalization

Let t be a λ-term.

- **Head normalization (HN):** there is a path from t to a head normal form.
- **Weak normalization (WN):** there is *at least one path* from t to **normal form (NF)**.
- **Strong normalization (SN):** there is *no infinite path* starting at t.

Normalization

SN $\Rightarrow$ WN $\Rightarrow$ HN.

Nota Bene: $y\,\Omega$ HNF but not WN $(\lambda x.y)\Omega$ WN but not SN

Characterizing Weak and Strong Normalization

HN	System $\mathscr{R}_0$ *any* arg. can be left *untyped*	$\mathtt{sz}(\Pi)$ bounds the number of *head* reduction steps
WN	System $\mathscr{R}_0$ + **unforgetfulness criterion** *non-erasable* args must be typed	$\mathtt{sz}(\Pi)$ bounds the number of leftmost-outermost red. steps (and more)
SN	Modify system $\mathscr{R}_0$ with **choice operator** *all* args must be typed	$\mathtt{sz}(\Pi)$ bounds the length of *any* reduction path

Subject reduction and expansion in $\mathscr{R}_0$

From a typing of $(\lambda x.r)s$ … to a typing of $r[s/x]$

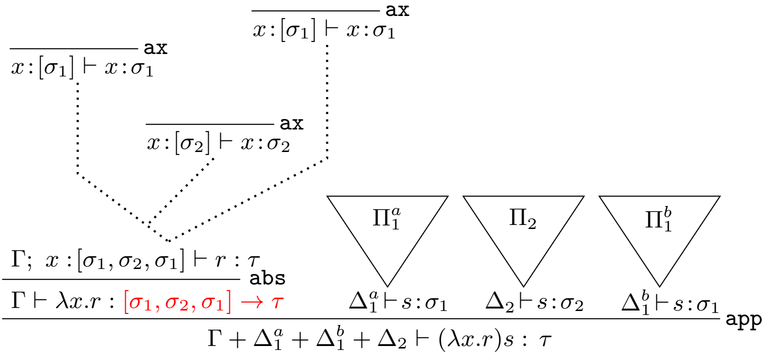
$$\dfrac{}{x:[\sigma_1]\vdash x:\sigma_1}\,\texttt{ax}$$

$$\dfrac{}{x:[\sigma_1]\vdash x:\sigma_1}\,\texttt{ax}$$

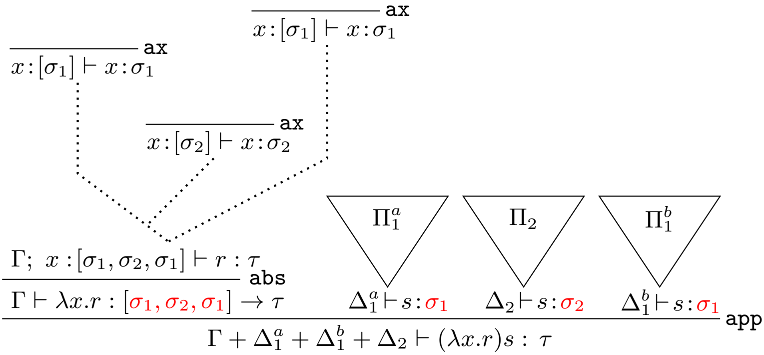
$$\dfrac{}{x:[\sigma_2]\vdash x:\sigma_2}\,\texttt{ax}$$

$$\dfrac{\dfrac{\Gamma;\ x:[\sigma_1,\sigma_2,\sigma_1]\vdash r:\tau}{\Gamma\vdash\lambda x.r:[\sigma_1,\sigma_2,\sigma_1]\to\tau}\,\texttt{abs}\quad \begin{array}{c}\Pi_1^a\\ \Delta_1^a\vdash s:\sigma_1\end{array}\quad \begin{array}{c}\Pi_2\\ \Delta_2\vdash s:\sigma_2\end{array}\quad \begin{array}{c}\Pi_1^b\\ \Delta_1^b\vdash s:\sigma_1\end{array}}{\Gamma+\Delta_1^a+\Delta_1^b+\Delta_2\vdash(\lambda x.r)s:\ \tau}\,\texttt{app}$$

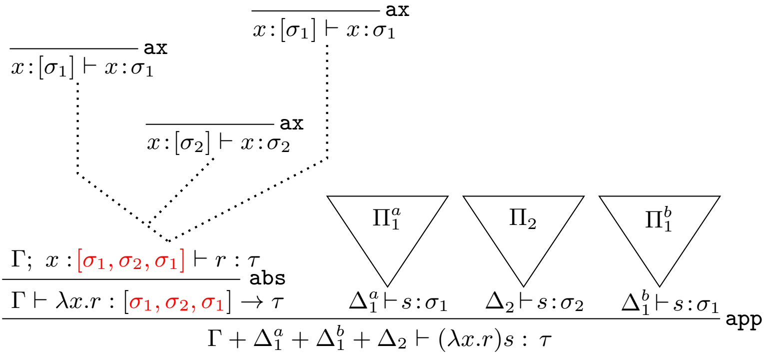
From a typing of $(\lambda x.r)s$ ... to a typing of $r[s/x]$



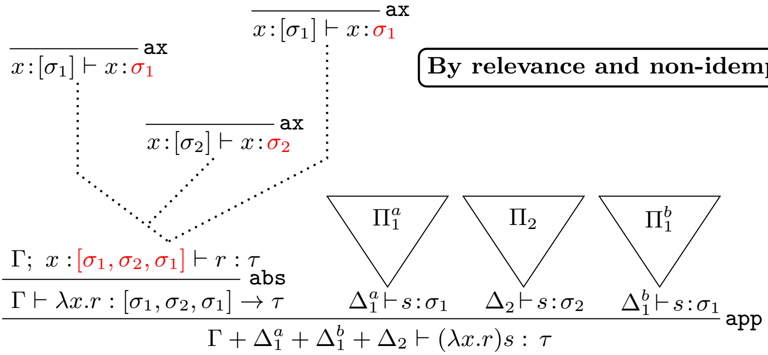
From a typing of $(\lambda x.r)s$ … to a typing of $r[s/x]$



From a typing of $(\lambda x.r)s$ ... to a typing of $r[s/x]$

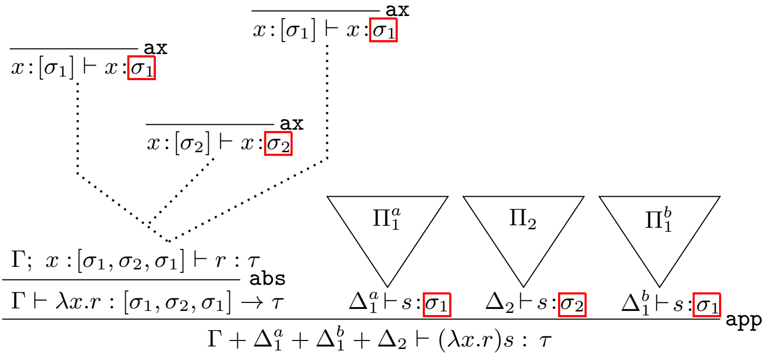


From a typing of $(\lambda x.r)s$ ... to a typing of $r[s/x]$

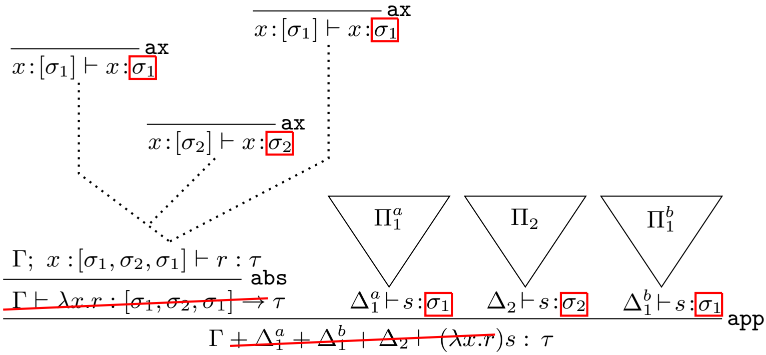


By relevance and non-idempotence !

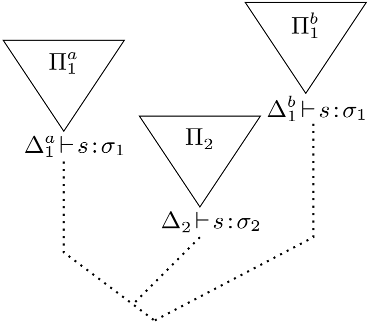
From a typing of $(\lambda x.r)s$ ... to a typing of $r[s/x]$



From a typing of $(\lambda x.r)s$ ... to a typing of $r[s/x]$

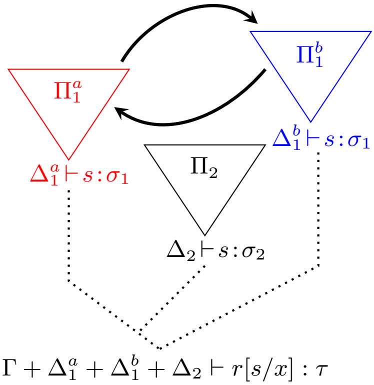


From a typing of $(\lambda x.r)s$ ... to a typing of $r[s/x]$



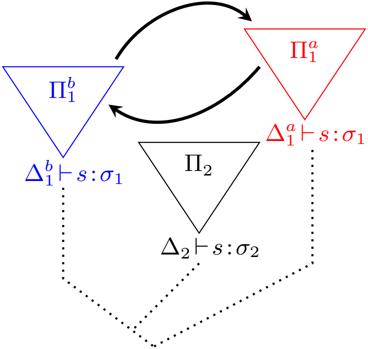
$\Gamma + \Delta_1^a + \Delta_1^b + \Delta_2 \vdash r[s/x] : \tau$

From a typing of $(\lambda x.r)s$ ... to a typing of $r[s/x]$



Non-determinism of SR

From a typing of $(\lambda x.r)s$ ... to a typing of $r[s/x]$



Non-determinism of SR

$\Gamma + \Delta_1^a + \Delta_1^b + \Delta_2 \vdash r[s/x] : \tau$

Plan

The Lambda-Mu Calculus

- Intuit. logic + Peirce's Law $((A \to B) \to A) \to A$
  gives classical logic.

- Intuit. logic + Peirce's Law $((A \to B) \to A) \to A$
  gives classical logic.

- **Griffin 90**: `call–cc` and Felleisen's $\mathcal{C}$-operator typable with Peirce's Law $((A \to B) \to A) \to A$
  $\rightsquigarrow$ the **Curry-Howard** iso extends to classical logic

The Lambda-Mu Calculus

- Intuit. logic + Peirce's Law $((A \to B) \to A) \to A$
  gives classical logic.

- **Griffin 90**: `call–cc` and Felleisen's $\mathcal{C}$-operator typable with Peirce's Law $((A \to B) \to A) \to A$
  $\leadsto$ the **Curry-Howard** iso extends to classical logic

- **Parigot 92**: $\lambda\mu$-calculus = computational interpretation of classical *natural deduction* (*e.g.*, *vs.* $\bar{\lambda}\mu\tilde{\mu}$).

The Lambda-Mu Calculus

- Intuit. logic + Peirce's Law $((A \to B) \to A) \to A$
  gives classical logic.

- **Griffin 90**: `call–cc` and Felleisen's $\mathcal{C}$-operator typable with Peirce's Law $((A \to B) \to A) \to A$
  $\rightsquigarrow$ the **Curry-Howard** iso extends to classical logic

- **Parigot 92**: $\lambda\mu$-calculus = computational interpretation of classical *natural deduction* (*e.g.*, *vs.* $\bar{\lambda}\mu\tilde{\mu}$).

- Captures **continuations**

Syntax: Variables x and **names** α

$$\begin{array}{llll} \text{(Objects)} & o & ::= & t \mid c \\ \text{(Terms)} & t, u & ::= & x \mid \lambda x.t \mid tu \mid \mu\alpha.c \\ \textbf{(Commands)} & c & ::= & [\alpha]t \end{array}$$

Basic Meta-Operations:

- $t[u/x]$ (subst.)
- $c\{u /\!\!/ \alpha\}$ **replaces** every occurrence of $[\alpha]v$ inside t by $[\alpha]v\,u$.

Syntax: Variables x and **names** α

$$
\begin{array}{llll}
\text{(Objects)} & o & ::= & t \mid c \\
\text{(Terms)} & t, u & ::= & x \mid \lambda x.t \mid tu \mid \mu\alpha.c \\
\textbf{(Commands)} & c & ::= & [\alpha]t
\end{array}
$$

Basic Meta-Operations:

- $t[u/x]$ (subst.)
- $c\{u /\!\!/ \alpha\}$ **replaces** every occurrence of $[\alpha]v$ inside t by $[\alpha]v\,u$.

Example:

- $[\alpha](x\,(\mu\gamma.[\alpha]x))\{u /\!\!/ \alpha\} =$

Syntax: Variables x and **names** α

$$\begin{array}{llll} \text{(Objects)} & o & ::= & t \mid c \\ \text{(Terms)} & t, u & ::= & x \mid \lambda x.t \mid tu \mid \mu\alpha.c \\ \textbf{(Commands)} & c & ::= & [\alpha]t \end{array}$$

Basic Meta-Operations:

- $t[u/x]$ (subst.)
- $c\{u /\!\!/ \alpha\}$ **replaces** every occurrence of $[\alpha]v$ inside t by $[\alpha]v\,u$.

Example:

- $[\alpha](x\,(\mu\gamma.[\alpha]x))\{u /\!\!/ \alpha\} =$

Syntax: Variables x and **names** α

$$\begin{array}{llll} \text{(Objects)} & o & ::= & t \mid c \\ \text{(Terms)} & t, u & ::= & x \mid \lambda x.t \mid tu \mid \mu\alpha.c \\ \textbf{(Commands)} & c & ::= & [\alpha]t \end{array}$$

Basic Meta-Operations:

- $t[u/x]$ (subst.)
- $c\{u /\!\!/ \alpha\}$ **replaces** every occurrence of $[\alpha]v$ inside t by $[\alpha]v\,u$.

Example:

- $[\alpha](x\,(\mu\gamma.[\alpha]x))\{u /\!\!/ \alpha\} =$

The $\lambda\mu$-calculus

Syntax: Variables x and **names** α

$$
\begin{array}{llll}
\text{(Objects)} & o & ::= & t \mid c \\
\text{(Terms)} & t, u & ::= & x \mid \lambda x.t \mid tu \mid \mu\alpha.c \\
\textbf{(Commands)} & c & ::= & [\alpha]t
\end{array}
$$

Basic Meta-Operations:

- $t[u/x]$ (subst.)
- $c\{u /\!\!/ \alpha\}$ **replaces** every occurrence of $[\alpha]v$ inside t by $[\alpha]v\,u$.

Example:

- $[\alpha](x\,(\mu\gamma.[\alpha]x))\{u /\!\!/ \alpha\} = [\alpha](x\,(\mu\gamma.[\alpha]x\,u))u$

Syntax: Variables x and **names** α

$$\begin{array}{llll} \text{(Objects)} & o & ::= & t \mid c \\ \text{(Terms)} & t, u & ::= & x \mid \lambda x.t \mid tu \mid \mu\alpha.c \\ \textbf{(Commands)} & c & ::= & [\alpha]t \end{array}$$

Basic Meta-Operations:

- $t[u/x]$ (subst.)
- $c\{u /\!\!/ \alpha\}$ **replaces** every occurrence of $[\alpha]v$ inside t by $[\alpha]v\,u$.

Example:

- $[\alpha](x\,(\mu\gamma.[\alpha]x))\{u /\!\!/ \alpha\} = [\alpha](x\,(\mu\gamma.[\alpha]x\,u))u$
- $\texttt{call-cc} := \lambda y.\mu\alpha.[\alpha]y(\lambda x.\mu\beta.[\alpha]x)$

Syntax: Variables x and **names** α

$$\begin{array}{llll} \text{(Objects)} & o & ::= & t \mid c \\ \text{(Terms)} & t, u & ::= & x \mid \lambda x.t \mid tu \mid \mu\alpha.c \\ \textbf{(Commands)} & c & ::= & [\alpha]t \end{array}$$

Basic Meta-Operations:

- $t[u/x]$ (subst.)
- $c\{u /\!\!/ \alpha\}$ **replaces** every occurrence of $[\alpha]v$ inside t by $[\alpha]v\,u$.

Example:

- $[\alpha](x\,(\mu\gamma.[\alpha]x))\{u /\!\!/ \alpha\} = [\alpha](x\,(\mu\gamma.[\alpha]x\,u))u$
- $\texttt{call-cc} := \lambda y.\mu\alpha.[\alpha]y(\lambda x.\mu\beta.[\alpha]x) : ((A \to B) \to A) \to A$ (simple typing)

The $\lambda\mu$-calculus

Syntax: Variables x and **names** α

$$
\begin{array}{llll}
\text{(Objects)} & o & ::= & t \mid c \\
\text{(Terms)} & t, u & ::= & x \mid \lambda x.t \mid tu \mid \mu\alpha.c \\
\textbf{(Commands)} & c & ::= & [\alpha]t
\end{array}
$$

Basic Meta-Operations:

- $t[u/x]$ (subst.)
- $c\{u/\!\!/\alpha\}$ **replaces** every occurrence of $[\alpha]v$ inside t by $[\alpha]v\,u$.

Example:

- $[\alpha](x\,(\mu\gamma.[\alpha]x))\{u/\!\!/\alpha\} = [\alpha](x\,(\mu\gamma.[\alpha]x\,u))u$
- `call–cc` $:= \lambda y.\mu\alpha.[\alpha]y(\lambda x.\mu\beta.[\alpha]x) : ((A \to B) \to A) \to A$ (simple typing)

Operational Semantics:

$$
\begin{array}{llll}
(\lambda x.t)u & \to_\beta & t[u/x] & \text{substitution} \\
(\mu\alpha.c)u & \to_\mu & \mu\alpha.c\{u/\!\!/\alpha\} & \text{replacement}
\end{array}
$$

Principles

Extend non-idempotent types to **classical logic**.

Problem 1:
finding *quantitative* descriptors suitable to classical logic

Problem 2:
guaranteeing a *decrease* in measure (weighted s.r.)

The Typing System

Principles

Extend non-idempotent types to **classical logic**.

Problem 1:
finding *quantitative* descriptors suitable to classical logic

$\rightsquigarrow$ resort to **non-idempotent union types** (below right)

Problem 2:
guaranteeing a *decrease* in measure (weighted s.r.)

Not obvious! The number of nodes does not work (see later).

Principles

Extend non-idempotent types to **classical logic**.

Problem 1:
finding *quantitative* descriptors suitable to classical logic

$\rightsquigarrow$ resort to **non-idempotent union types** (below right)

Problem 2:
guaranteeing a *decrease* in measure (weighted s.r.)

Not obvious! The number of nodes does not work (see later).

Intersection: $\mathcal{I}, \mathcal{J} := [\mathcal{U}_k]_{k \in K}$

$\mathcal{U}, \mathcal{V} =: \langle \sigma_k \rangle_{k \in K}$: **Union**

The Typing System

Principles

Extend non-idempotent types to **classical logic**.

Problem 1:
finding *quantitative* descriptors suitable to classical logic

$\rightsquigarrow$ resort to **non-idempotent union types** (below right)

Problem 2:
guaranteeing a *decrease* in measure (weighted s.r.)

Not obvious! The number of nodes does not work (see later).

Intersection: $\mathcal{I}, \mathcal{J} := [\mathcal{U}_k]_{k \in K}$ $\qquad$ $\mathcal{U}, \mathcal{V} =: \langle \sigma_k \rangle_{k \in K}$: **Union**

$$x : [\mathcal{U}_1, \mathcal{U}_2];\ y : [\mathcal{V}] \vdash t : \mathcal{U} \mid \alpha : \langle \sigma_1, \sigma_2 \rangle, \beta : \langle \tau_1, \tau_2, \tau_3 \rangle$$

The Typing System

Principles

Extend non-idempotent types to **classical logic**.

Problem 1:
finding *quantitative* descriptors suitable to classical logic

$\rightsquigarrow$ resort to **non-idempotent union types** (below right)

Problem 2:
guaranteeing a *decrease* in measure (weighted s.r.)

Not obvious! The number of nodes does not work (see later).

Intersection: $\mathcal{I}, \mathcal{J} := [\mathcal{U}_k]_{k \in K}$ $\qquad$ $\mathcal{U}, \mathcal{V} =: \langle \sigma_k \rangle_{k \in K}$: **Union**

$$x : [\mathcal{U}_1, \mathcal{U}_2];\ y : [\mathcal{V}] \vdash t : \mathcal{U} \mid \alpha : \langle \sigma_1, \sigma_2 \rangle, \beta : \langle \tau_1, \tau_2, \tau_3 \rangle$$

A, **C** and **non-I** *e.g.*, $\quad \langle \sigma_1, \sigma_2 \rangle \vee \langle \sigma_1 \rangle = \langle \sigma_1, \sigma_2, \sigma_1 \rangle$

Some Typing Rules (System $\mathcal{H}_{\lambda\mu}$)

Features

Syntax-direction, relevance, multiplicative rules **accumulation of typing information**.

- `app`-rule based upon the *admissible* rule of ND:

$$\frac{A_1 \to B_1 \vee \ldots \vee A_k \to B_k \qquad A_1 \wedge \ldots \wedge A_k}{B_1 \vee \ldots \vee B_k} \qquad \left(vs. \frac{A \to B \quad A}{B} \right)$$

Some Typing Rules (System $\mathcal{H}_{\lambda\mu}$)

Features

Syntax-direction, relevance, multiplicative rules **accumulation of typing information**.

- **app**-rule based upon the *admissible* rule of ND:

$$\frac{A_1 \to B_1 \vee \ldots \vee A_k \to B_k \qquad A_1 \wedge \ldots \wedge A_k}{B_1 \vee \ldots \vee B_k} \qquad \left(vs. \frac{A \to B \quad A}{B} \right)$$

- Two new rules (manipulation on the right-h.s.):

$$\frac{\Gamma \vdash t : \mathcal{U} \mid \Delta}{\Gamma \vdash [\alpha]t : \# \mid \Delta \vee \{\alpha : \mathcal{U}\}}\ \texttt{save} \qquad \frac{\Gamma \vdash c : \# \mid \Delta}{\Gamma \vdash \mu\alpha.c : \Delta(\alpha)^* \mid \Delta \setminus\!\!\setminus \alpha}\ \texttt{restore}$$

Some Typing Rules (System $\mathcal{H}_{\lambda\mu}$)

Features

Syntax-direction, relevance, multiplicative rules **accumulation of typing information**.

- `app`-rule based upon the *admissible* rule of ND:

$$\frac{A_1 \to B_1 \vee \ldots \vee A_k \to B_k \qquad A_1 \wedge \ldots \wedge A_k}{B_1 \vee \ldots \vee B_k} \qquad \left(vs. \frac{A \to B \quad A}{B} \right)$$

- Two new rules (manipulation on the right-h.s.):

$$\frac{\Gamma \vdash t : \mathcal{U} \mid \Delta}{\Gamma \vdash [\alpha]t : \# \mid \Delta \vee \{\alpha : \mathcal{U}\}}\ \texttt{save} \qquad \frac{\Gamma \vdash c : \# \mid \Delta}{\Gamma \vdash \mu\alpha.c : \Delta(\alpha)^* \mid \Delta \setminus\!\!\setminus \alpha}\ \texttt{restore}$$

Some Typing Rules (System $\mathcal{H}_{\lambda\mu}$)

Features

Syntax-direction, relevance, multiplicative rules **accumulation of typing information**.

- **app**-rule based upon the *admissible* rule of ND:

$$\frac{A_1 \to B_1 \vee \ldots \vee A_k \to B_k \qquad A_1 \wedge \ldots \wedge A_k}{B_1 \vee \ldots \vee B_k} \qquad \left(vs. \frac{A \to B \quad A}{B} \right)$$

- Two new rules (manipulation on the right-h.s.):

$$\frac{\Gamma \vdash t : \mathcal{U} \mid \Delta}{\Gamma \vdash [\alpha]t : \# \mid \Delta \vee \{\alpha : \mathcal{U}\}}\ \texttt{save} \qquad \frac{\Gamma \vdash c : \# \mid \Delta}{\Gamma \vdash \mu\alpha.c : \Delta(\alpha)^* \mid \Delta \setminus\!\!\setminus \alpha}\ \texttt{restore}$$

Some Typing Rules (System $\mathcal{H}_{\lambda\mu}$)

Features

Syntax-direction, relevance, multiplicative rules **accumulation of typing information**.

- `app`-rule based upon the *admissible* rule of ND:

$$\frac{A_1 \to B_1 \vee \ldots \vee A_k \to B_k \qquad A_1 \wedge \ldots \wedge A_k}{B_1 \vee \ldots \vee B_k} \qquad \left(vs. \frac{A \to B \quad A}{B} \right)$$

- Two new rules (manipulation on the right-h.s.):

$$\frac{\Gamma \vdash t : \mathcal{U} \mid \Delta}{\Gamma \vdash [\alpha]t : \# \mid \Delta \vee \{\alpha : \mathcal{U}\}}\ \texttt{save} \qquad \frac{\Gamma \vdash c : \# \mid \Delta}{\Gamma \vdash \mu\alpha.c : \Delta(\alpha)^* \mid \Delta \setminus\!\!\setminus \alpha}\ \texttt{restore}$$

Some Typing Rules (System $\mathcal{H}_{\lambda\mu}$)

Features

Syntax-direction, relevance, multiplicative rules **accumulation of typing information**.

- `app`-rule based upon the *admissible* rule of ND:

$$\frac{A_1 \to B_1 \vee \ldots \vee A_k \to B_k \qquad A_1 \wedge \ldots \wedge A_k}{B_1 \vee \ldots \vee B_k} \qquad \left(vs. \frac{A \to B \quad A}{B} \right)$$

- Two new rules (manipulation on the right-h.s.):

$$\frac{\Gamma \vdash t : \mathcal{U} \mid \Delta}{\Gamma \vdash [\alpha]t : \# \mid \Delta \vee \{\alpha : \mathcal{U}\}} \texttt{save} \qquad \frac{\Gamma \vdash c : \# \mid \Delta}{\Gamma \vdash \mu\alpha.c : \Delta(\alpha)^* \mid \Delta \setminus\!\setminus \alpha} \texttt{restore}$$

where $\_^* =$ **choice operator**.

Some Typing Rules (System $\mathcal{H}_{\lambda\mu}$)

Features

Syntax-direction, relevance, multiplicative rules **accumulation of typing information**.

- `app`-rule based upon the *admissible* rule of ND:

$$\frac{A_1 \to B_1 \vee \ldots \vee A_k \to B_k \qquad A_1 \wedge \ldots \wedge A_k}{B_1 \vee \ldots \vee B_k} \qquad \left(vs. \frac{A \to B \quad A}{B} \right)$$

- Two new rules (manipulation on the right-h.s.):

$$\frac{\Gamma \vdash t : \mathcal{U} \mid \Delta}{\Gamma \vdash [\alpha]t : \# \mid \Delta \vee \{\alpha : \mathcal{U}\}} \, \texttt{save} \qquad \frac{\Gamma \vdash c : \# \mid \Delta}{\Gamma \vdash \mu\alpha.c : \Delta(\alpha)^* \mid \Delta \setminus\!\!\setminus \alpha} \, \texttt{restore}$$

where $\_^*$ = **choice operator**.

$$\boxed{\texttt{call-cc} : [[[A]{\to}B]{\to}A] \to \langle A, A\rangle \qquad \text{vs.} \qquad ((A \to B) \to A) \to A}$$

Properties and contributions (1)

- **Weighted Subject Reduction**

  with $\texttt{size}(\Pi) = \begin{cases} \text{number of nodes of } \Pi\ + \\ \text{size of the } \textbf{type arities} \text{ of all the names of commands } + \\ \textbf{multiplicities} \text{ of arguments in all the } \textbf{app. nodes} \text{ of } \Pi. \end{cases}$

- **Subject Expansion**

Theorem (Kesner, Vial, FSCD17)

Let t be a $\lambda\mu$-term. Then equivalence between:

1. *t is typable (in $\mathcal{H}_{\lambda\mu}$)*
2. *t is HN*
3. *the head reduction strategy terminates on t (thus, h.r.strat. certified!).*

Bonus (quantitative information)

$\texttt{size}(\Pi)$ **bounds** the number of **steps** of the head. red. strategy on t.

Contributions (2)

Theorem (Kesner,Vial, FSCD17)

- *System $\mathcal{S}_{\lambda\mu}$ characterizing SN for the $\lambda\mu$-calculus.*
- $\mathtt{sz}(\Pi)$ *bounds the length of any reduction sequence starting at t.*

Extension (small-step operational semantics for the $\lambda\mu$-calculus)

- Processing substitution and replacement *one occurrence at a time.*
  - In λ: $(x\,y\,x\,x)[s/x] \rightsquigarrow s\,y\,s\,s$ (1 big step)
  - In $\lambda_{\mathtt{ex}}$ $(x\,y\,x\,x)[s/x] \rightsquigarrow s\,y\,x\,x \rightsquigarrow s\,y\,x\,s \rightsquigarrow s\,y\,s\,s$ (3 small-steps)
- Characterization of SN (extension of $\mathcal{S}_{\lambda\mu}$).

Plan

Klop's Problem

- **HN**, **WN**, **SN**,... have been *statically* characterized by various **ITS**.
- **Klop's Problem:** can the set of ∞-WN terms be characterized by an ITS ?
  Def: t is ∞-WN iff its Böhm tree does not contain $\perp$

> - **Tatsuta [07]:** an inductive ITS cannot do it.
> - Can a coinductive ITS characterize the set of ∞-WN terms?

Klop's Problem

- **HN**, **WN**, **SN**,... have been *statically* characterized by various **ITS**.
- **Klop's Problem:** can the set of ∞-WN terms be characterized by an ITS ?
  Def: t is ∞-WN iff its Böhm tree does not contain $\perp$

  - **Tatsuta [07]:** an inductive ITS cannot do it.
  - Can a coinductive ITS characterize the set of ∞-WN terms?

- **YES**, with ITS = **sequential** + **validity criterion**.

Klop's Problem

- **HN**, **WN**, **SN**,... have been *statically* characterized by various **ITS**.

- **Klop's Problem:** can the set of ∞-WN terms be characterized by an ITS ?
  Def: t is ∞-WN iff its Böhm tree does not contain $\perp$

  - **Tatsuta [07]:** an inductive ITS cannot do it.
  - Can a coinductive ITS characterize the set of ∞-WN terms?

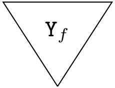
- **YES**, with ITS = **sequential** + **validity criterion**.
- But... what is infinitary normalization?

Productive vs. Unproductive Reduction

Productive vs. Unproductive Reduction

Productive reduction: $\Delta_f := \lambda x.f(xx)$ $\qquad \mathtt{Y}_f := \Delta_f \Delta_f$ "Curry f"

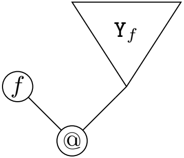
$$\mathtt{Y}_f \to f(\mathtt{Y}_f) \to f^2(\mathtt{Y}_f) \to f^3(\mathtt{Y}_f) \to f^4(\mathtt{Y}_f) \to \ldots \to f^n(\mathtt{Y}_f) \to \ldots \to^\infty f^\omega$$



Productive vs. Unproductive Reduction

Productive reduction: $\Delta_f := \lambda x.f(xx)$ $\qquad$ $\mathtt{Y}_f := \Delta_f \Delta_f$ "Curry f"

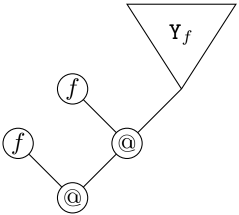
$$\mathtt{Y}_f \to f(\mathtt{Y}_f) \to f^2(\mathtt{Y}_f) \to f^3(\mathtt{Y}_f) \to f^4(\mathtt{Y}_f) \to \ldots \to f^n(\mathtt{Y}_f) \to \ldots \to^\infty f^\omega$$



Productive vs. Unproductive Reduction

Productive reduction: $\Delta_f := \lambda x.f(xx)$ $\quad$ $\mathtt{Y}_f := \Delta_f\Delta_f$ "Curry f"

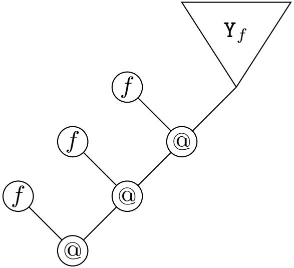
$$\mathtt{Y}_f \to f(\mathtt{Y}_f) \to f^2(\mathtt{Y}_f) \to f^3(\mathtt{Y}_f) \to f^4(\mathtt{Y}_f) \to \ldots \to f^n(\mathtt{Y}_f) \to \ldots \to^\infty f^\omega$$



Productive vs. Unproductive Reduction

Productive reduction: $\Delta_f := \lambda x.f(xx)$ $\quad \mathtt{Y}_f := \Delta_f\Delta_f$ "Curry f"

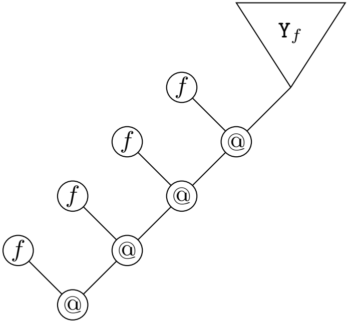
$\mathtt{Y}_f \to f(\mathtt{Y}_f) \to f^2(\mathtt{Y}_f) \to f^3(\mathtt{Y}_f) \to f^4(\mathtt{Y}_f) \to \ldots \to f^n(\mathtt{Y}_f) \to \ldots \to^\infty f^\omega$



Productive vs. Unproductive Reduction

Productive reduction: $\Delta_f := \lambda x.f(xx)$ $\mathtt{Y}_f := \Delta_f \Delta_f$ "Curry f"

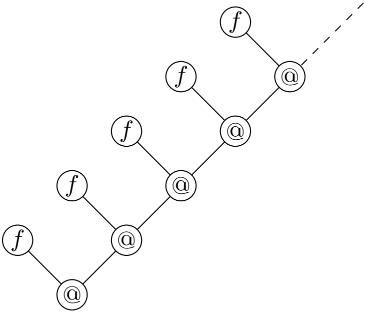
$$\mathtt{Y}_f \to f(\mathtt{Y}_f) \to f^2(\mathtt{Y}_f) \to f^3(\mathtt{Y}_f) \to f^4(\mathtt{Y}_f) \to \ldots \to f^n(\mathtt{Y}_f) \to \ldots \to^\infty f^\omega$$



Productive vs. Unproductive Reduction

Productive reduction: $\Delta_f := \lambda x.f(xx)$ $\quad$ $\mathtt{Y}_f := \Delta_f \Delta_f$ "Curry f"

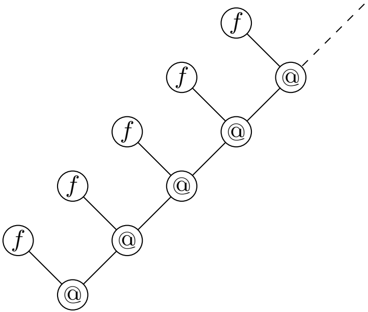
$$\mathtt{Y}_f \to f(\mathtt{Y}_f) \to f^2(\mathtt{Y}_f) \to f^3(\mathtt{Y}_f) \to f^4(\mathtt{Y}_f) \to \ldots \to f^n(\mathtt{Y}_f) \to \ldots \to^\infty f^\omega$$



Productive vs. Unproductive Reduction

Productive reduction: $\Delta_f := \lambda x.f(xx)$ $\quad$ $\mathtt{Y}_f := \Delta_f\Delta_f$ "Curry f"

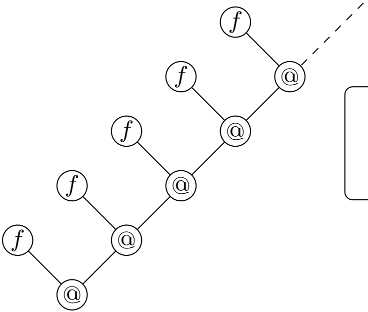
$$\mathtt{Y}_f \to f(\mathtt{Y}_f) \to f^2(\mathtt{Y}_f) \to f^3(\mathtt{Y}_f) \to f^4(\mathtt{Y}_f) \to \ldots \to f^n(\mathtt{Y}_f) \to \ldots \to^{\infty} f^{\omega}$$



Productive vs. Unproductive Reduction

Productive reduction: $\Delta_f := \lambda x.f(xx)$ $\qquad$ $\mathtt{Y}_f := \Delta_f \Delta_f$ "Curry f"

$$\mathtt{Y}_f \to f(\mathtt{Y}_f) \to f^2(\mathtt{Y}_f) \to f^3(\mathtt{Y}_f) \to f^4(\mathtt{Y}_f) \to \ldots \to f^n(\mathtt{Y}_f) \to \ldots \to^{\infty} f^{\omega}$$

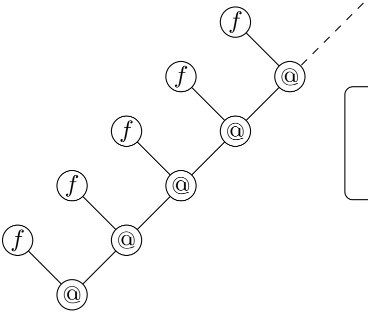


- $\mathtt{Y}_f$ not WN
- $\mathtt{Y}_f$ is ∞-WN
- ∞-NF: $f^\omega = f(f^\omega)$

Productive vs. Unproductive Reduction

Productive reduction: $\Delta_f := \lambda x.f(xx)$ $\quad$ $\mathtt{Y}_f := \Delta_f \Delta_f$ "Curry f"

$$\mathtt{Y}_f \to f(\mathtt{Y}_f) \to f^2(\mathtt{Y}_f) \to f^3(\mathtt{Y}_f) \to f^4(\mathtt{Y}_f) \to \ldots \to f^n(\mathtt{Y}_f) \to \ldots \to^{\infty} f^{\omega}$$



- $\mathtt{Y}_f$ not WN
- $\mathtt{Y}_f$ is ∞-WN
- ∞-NF: $f^\omega = f(f^\omega)$

Unproductive reduction: $\Delta = \lambda x.x\,x$, $\Omega = \Delta\,\Delta$ (*i.e.* `autoapp(autoapp)`)

$$\Omega \to \Omega \to \Omega \to \Omega \to \Omega \to \Omega \to \ldots$$

Infinite Terms

- Infinite λ-terms.
- Infinite NF *e.g.*, f^ω.
- *Productive* reduction sequence of infinite length (**strongly converging reduction sequence)**
$$\mathtt{Y}_f \to f(\mathtt{Y}_f)\ldots \text{ ok} \qquad \text{not } \Omega \to \Omega \ldots$$
- A term t is ∞**-WN** if $\exists$ a reduction path to an ∞-NF.
- **Hereditary head reduction strategy:**
from lower (root) to upper levers.

Towards Infinitary Typing

Idea

To characterize ∞-WN, let us unforgetfully type infinite normal forms
$\leadsto$ no part of an ∞-NF must be left untyped...

- Need to consider infinite derivations with a coinductive type grammar ($\mathscr{R}_0 \leadsto \mathscr{R}$).

Towards Infinitary Typing

Idea

To characterize ∞-WN, let us unforgetfully type infinite normal forms
$\rightsquigarrow$ no part of an ∞-NF must be left untyped...

- Need to consider infinite derivations with a coinductive type grammar ($\mathscr{R}_0 \rightsquigarrow \mathscr{R}$).

Problem 1: how do we perform infinite subject reduction/expansion?

Actually, this is difficult only for SE (extra-slide available)

Problem 2: the coinductive type grammar allows to define $\rho = [\rho]_\omega \to o$.

Using ρ, we may type Ω with o (*unsound* derivations)

Idea

To characterize ∞-WN, let us unforgetfully type infinite normal forms
$\rightsquigarrow$ no part of an ∞-NF must be left untyped...

- Need to consider infinite derivations with a coinductive type grammar ($\mathscr{R}_0 \rightsquigarrow \mathscr{R}$).

Problem 1: how do we perform infinite subject reduction/expansion?

Actually, this is difficult only for SE (extra-slide available)

Problem 2: the coinductive type grammar allows to define $\rho = [\rho]_\omega \to o$.

Using ρ, we may type Ω with o (*unsound* derivations)

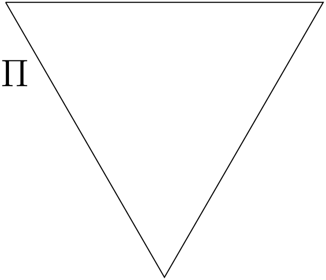
- **Solution (for both problems):** resort to a *validity criterion* called **approximability**.

- A derivation is a set of symbols, that satisfies some grammar.
- Some derivations are included in others

$$\dfrac{\dfrac{}{x : [[\ ] \to o] \vdash x : [\ ] \to o}}{x : [[o] \to o] \qquad \vdash x\, y : o}$$

- **Informal Definition [Vial, LICS17]:** a derivation Π is approximable if, for all *finite* selection of symbols B_0, there is a *finite* derivation Π_f included in Π and containing B_0.

Π

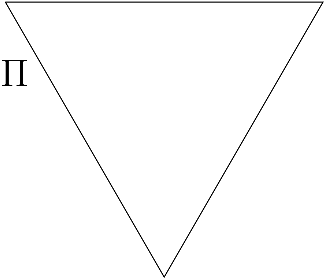


Approximability (intuitions)

- A derivation is a set of symbols, that satisfies some grammar.
- Some derivations are included in others

$$\dfrac{\dfrac{}{x : [[o] \to o] \vdash x : [o] \to o} \qquad \dfrac{}{y : [o] \vdash y : o}}{x : [[o] \to o]; y : [o] \vdash x\, y : o}$$

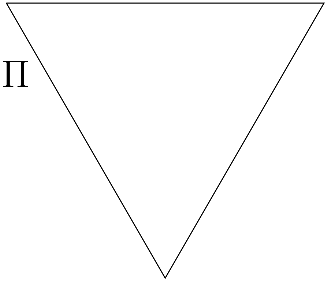
- **Informal Definition [Vial, LICS17]:** a derivation Π is approximable if, for all *finite* selection of symbols B_0, there is a *finite* derivation Π_f included in Π and containing B_0.



- A derivation is a set of symbols, that satisfies some grammar.
- Some derivations are included in others (black $\subseteq$ black+red)

$$\dfrac{\dfrac{}{x : [[o] \to o] \vdash x : [o] \to o} \qquad \dfrac{}{y : [o] \vdash y : o}}{x : [[o] \to o]; y : [o] \vdash x\, y : o}$$

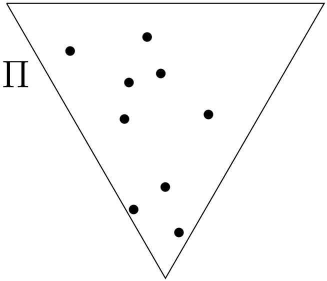
- **Informal Definition [Vial, LICS17]:** a derivation Π is approximable if, for all *finite* selection of symbols B_0, there is a *finite* derivation Π_f included in Π and containing B_0.



- A derivation is a set of symbols, that satisfies some grammar.
- Some derivations are included in others (black $\subseteq$ black+red)

$$\dfrac{\dfrac{}{x : [[o] \to o] \vdash x : [o] \to o} \qquad \dfrac{}{y : [o] \vdash y : o}}{x : [[o] \to o]; y : [o] \vdash x\, y : o}$$

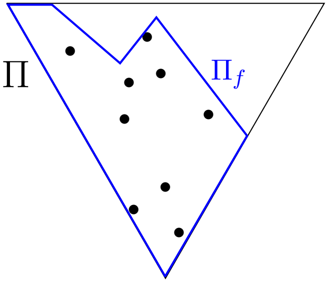
- **Informal Definition [Vial, LICS17]:** a derivation Π is approximable if, for all *finite* selection of symbols B_0, there is a *finite* derivation Π_f included in Π and containing B_0.



- A derivation is a set of symbols, that satisfies some grammar.
- Some derivations are included in others (black $\subseteq$ black+red)

$$\dfrac{\dfrac{}{x : [[o] \to o] \vdash x : [o] \to o} \qquad \dfrac{}{y : [o] \vdash y : o}}{x : [[o] \to o]; y : [o] \vdash x\, y : o}$$

- **Informal Definition [Vial, LICS17]:** a derivation Π is approximable if, for all *finite* selection of symbols B_0, there is a *finite* derivation Π_f included in Π and containing B_0.

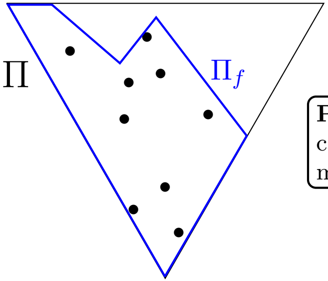


Approximability (intuitions)

- A derivation is a set of symbols, that satisfies some grammar.
- Some derivations are included in others (black $\subseteq$ black+red)

$$\dfrac{\dfrac{}{x : [[o] \to o] \vdash x : [o] \to o} \qquad \dfrac{}{y : [o] \vdash y : o}}{x : [[o] \to o]; y : [o] \vdash x\, y : o}$$

- **Informal Definition [Vial, LICS17]:** a derivation Π is approximable if, for all *finite* selection of symbols B_0, there is a *finite* derivation Π_f included in Π and containing B_0.



Problem 3: Approximability cannot be expressed with multisets.

(no tracking with multisets)

Solution

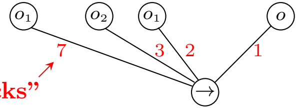
Resorting to sequential intersection ! ($\rightsquigarrow$ approximability becomes definable)

- **Strict Types:**

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

- **Sequence Types** $(k \cdot S_k)_{k \in K}$

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



7, 3, 2, 1 = **"tracks"**

- **Tracking:** $(3 \cdot \sigma, 5 \cdot \tau, 9 \cdot \sigma) = (3 \cdot \sigma, 5 \cdot \tau) \uplus (9 \cdot \tau)$

  vs. $[\sigma, \tau, \sigma] = [\sigma, \tau] + [\sigma]$

Sequential intersection

Solution

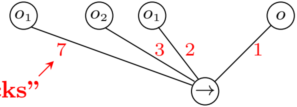
Resorting to sequential intersection ! ($\rightsquigarrow$ approximability becomes definable)

- **Strict Types:**

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

- **Sequence Types** $(k \cdot S_k)_{k \in K}$

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



7, 3, 2, 1 = **"tracks"**

- **Tracking:** $(3 \cdot \sigma, 5 \cdot \tau, 9 \cdot \sigma) = (3 \cdot \sigma, 5 \cdot \tau) \uplus (9 \cdot \tau)$

vs. $[\sigma, \tau, \sigma] = [\sigma, \tau] + [\sigma]$

Solution

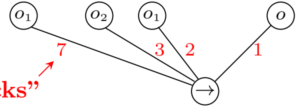
Resorting to sequential intersection ! ($\rightsquigarrow$ approximability becomes definable)

- **Strict Types:**

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

- **Sequence Types** $(k \cdot S_k)_{k \in K}$

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



7, 3, 2, 1 = **"tracks"**

- **Tracking:** $(3 \cdot \sigma, 5 \cdot \tau, 9 \cdot \sigma) = (3 \cdot \sigma, 5 \cdot \tau) \uplus (9 \cdot \tau)$

  vs. $[\sigma, \tau, \sigma] = [\sigma, \tau] + [\sigma]$

Sequential intersection

Solution

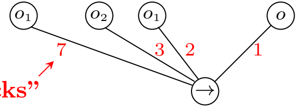
Resorting to sequential intersection ! ($\rightsquigarrow$ approximability becomes definable)

- **Strict Types:**

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

- **Sequence Types** $\quad (k \cdot S_k)_{k \in K}$

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



7, 3, 2, 1 = “tracks”

- **Tracking:** $(3 \cdot \sigma, 5 \cdot \tau, 9 \cdot \sigma) = (3 \cdot \sigma, 5 \cdot \tau) \uplus (9 \cdot \tau)$

$$vs. \ [\sigma, \tau, \sigma] = [\sigma, \tau] + [\sigma]$$

Solution

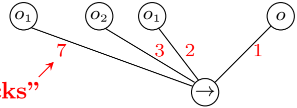
Resorting to sequential intersection ! ($\rightsquigarrow$ approximability becomes definable)

- **Strict Types:**

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

- **Sequence Types** $(k \cdot S_k)_{k \in K}$

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



7, 3, 2, 1 = **"tracks"**

- **Tracking:** $(3 \cdot \sigma, 5 \cdot \tau, 9 \cdot \sigma) = (3 \cdot \sigma, 5 \cdot \tau) \uplus (9 \cdot \tau)$

$$vs. \ [\sigma, \tau, \sigma] = [\underset{?}{\sigma}, \tau] + [\underset{?}{\sigma}]$$

Derivations of S

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

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

- System S features **pointers** (called **bipositions**).

Approximability is definable in S

Problem 3 solved!

- Every S-derivation collapses on a $\mathscr{R}$-derivation.

Theorem

Given t, the set of the S-derivations typing t is a complete partial order (c.p.o.).

Characterization of infinitary WN

Proposition (Vial, LICS17)

In System $\mathtt{S}$:

- *SR:* typing is stable by productive ∞-reduction.
- *SE: approximable* typing stable by productive ∞-expansion.

Theorem (Vial, LICS17)

- *A ∞-term t is ∞-WN iff t is unforgetfully typable by means of an approximable derivation* $\leadsto$ *Klop's Problem solved*
- *The hereditary head reduction strategy is complete for infinitary weak normalization.*

Characterization of infinitary WN

Proposition (Vial, LICS17)

In System $\mathtt{S}$:

- *SR:* typing is stable by productive ∞-reduction.
- *SE: approximable* typing stable by productive ∞-expansion.

Theorem (Vial, LICS17)

- *A ∞-term t is ∞-WN iff t is unforgetfully typable by means of an approximable derivation* $\rightsquigarrow$ *Klop's Problem solved*
- *The hereditary head reduction strategy is complete for infinitary weak normalization.*

Last bonus (positive answer to TLCA Problem #20)

System $\mathtt{S}$ also provides a type-theoretic characterization of the **hereditary permutations** (not possible in the inductive case, Tatsuta [LICS07]).

Plan

Two questions arising from Klop's problem

Question 1 (the set of typable terms)

What is the set of typable terms in system $\mathscr{R}$ and $\mathtt{S}$? (without approximability condition)

Question 2 (relation between $\mathtt{S}$ and $\mathscr{R}$)

Every $\mathtt{S}$-derivation collapses on a $\mathscr{R}$-derivation.
But is the converse true?

Two questions arising from Klop's problem

Question 1 (the set of typable terms)

What is the set of typable terms in system $\mathscr{R}$ and $\mathtt{S}$? (without approximability condition)

Theorem (Vial)

- *Every term is typable in systems $\mathscr{R}$ and* $\mathtt{S}$ *(non-trivial).*
- *One can extract from $\mathscr{R}$-typing the* ***order*** *(arity) of any λ-term.*
- *In the infinitary relational model, no term has an empty denotation.*

Question 2 (relation between $\mathtt{S}$ and $\mathscr{R}$)

Every $\mathtt{S}$-derivation collapses on a $\mathscr{R}$-derivation.
But is the converse true?

Two questions arising from Klop's problem

Question 1 (the set of typable terms)

What is the set of typable terms in system $\mathscr{R}$ and $\mathtt{S}$? (without approximability condition)

Theorem (Vial)

- *Every term is typable in systems $\mathscr{R}$ and $\mathtt{S}$ (non-trivial).*
- *One can extract from $\mathscr{R}$-typing the* ***order*** *(arity) of any λ-term.*
- *In the infinitary relational model, no term has an empty denotation.*

Question 2 (relation between $\mathtt{S}$ and $\mathscr{R}$)

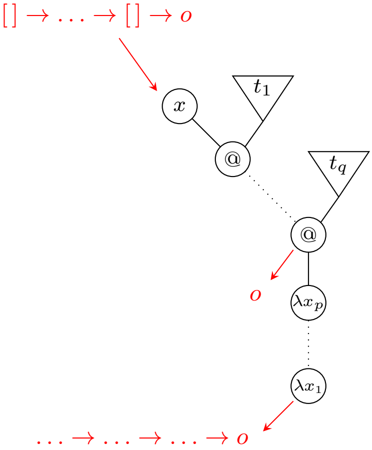
Every $\mathtt{S}$-derivation collapses on a $\mathscr{R}$-derivation.
But is the converse true?

Theorem (Vial)

- *Every $\mathscr{R}$-derivation is the collapse of a $\mathtt{S}$-derivation.*
- *One can encode any reduction choice in system $\mathscr{R}$ b.m.o. a $\mathtt{S}$-derivation.*

Difficulties

- In the *productive* cases (HN,WN,SN,∞-WN), in i.t.s., one types the normal forms and uses subject expansion.
  $\boxed{\text{normalizing terms} \subseteq \text{typable terms}}$
- Here, no form of productivity/stabilization.
- We develop a corpus of methods inspired by **first order model theory** (last part of the dissertation).



Plan

Intersection types *via* Grothendieck construction
[Mazza,Pellissier,Vial, POPL2018]

- Categorical generalization of ITS. *à la* Melliès-Zeilberger.
- Type systems = 2-operads (see below).

Type systems as 2-operads

- Level 1: $\Gamma \vdash t : B$ $\quad$ t = *multimorphism* from Γ to B.
- Level 2: if $\Gamma \vdash t : B \overset{\text{SR}}{\rightsquigarrow} \Gamma \vdash t' : B$,
  $t \rightsquigarrow t'$ = *2-morphism* from t to t'.

- Construction of an i.t.s. via a Grothendieck construction (pullbacks).
- **Modularity:** retrieving automatically
  e.g., *e.g.*, Coppo-Dezani, Gardner, $\mathscr{R}_0$, call-by-value + $\mathcal{H}_{\lambda\mu}$ (use *cyclic* 2-operads)

The $\lambda\mu$-calculus:

- Characterization of HN and SN with non-idempotent/quantitative methods (extension of $\mathscr{R}_0$).
- Certification of reduction strategies.
- **Upper** bounds on normalizing strategies.
- Small-step operational semantics and SN (extension).

What we did and what we shall do

The $\lambda\mu$-calculus:

- Characterization of HN and SN with non-idempotent/quantitative methods (extension of $\mathscr{R}_0$).
- Certification of reduction strategies.
- **Upper** bounds on normalizing strategies.
- Small-step operational semantics and SN (extension).

Perspectives

- **Exact** bounds on normalizing strategies (*à la* Bernadet-Lengrand).
- Quantitative types for other classical calculi (*e.g.*, Curien-Herbelin's $\bar{\lambda}\mu\tilde{\mu}$).
- Studying the model underlying $\mathcal{H}_{\lambda\mu}$.

What we did and what we shall do

What we did and what we shall do

Klop's Problem and Infinitary Normalization

- Characterizing *infinitary* weak normalization.
- Certifying an *infinitary* reduction strategy (HHN).
- Positive answer to TLCA Problem # 20.

- Introduction of system $\mathtt{S}$ (sequential intersection, non-idem. flavor).
- Introduction of a validity criterion (*approximability*).

Klop's Problem and Infinitary Normalization

- Characterizing *infinitary* weak normalization.
- Certifying an *infinitary* reduction strategy (HHN).
- Positive answer to TLCA Problem # 20.

- Introduction of system $\mathtt{S}$ (sequential intersection, non-idem. flavor).
- Introduction of a validity criterion (*approximability*).

Perspectives

- Other forms of ∞-normalization (other calculi, ∞-SN)
- Relations between system $\mathtt{S}$ and ludics, GoI, indexed LL...
- Relations with Grellois-Melliès infinitary model of LL.

Thank you

Thank you for your attention!

$$\cfrac{\cfrac{\cfrac{\cfrac{}{(A \to B) \to A \vdash (A \to B) \to A} \qquad \cfrac{\cfrac{}{A \vdash A, B}}{\vdash A \to B, A}}{(A \to B) \to A \vdash A, A}}{(A \to B) \to A \vdash A}}{\vdash ((A \to B) \to A) \to A}$$

Standard Style

Peirce's Law in Classical Natural Deduction

$$\dfrac{\dfrac{\dfrac{}{(A \to B) \to A \vdash (A \to B) \to A} \qquad \dfrac{\dfrac{}{A \vdash A, {\color{red}B}}}{\vdash A \to B, A}}{\dfrac{(A \to B) \to A \vdash A, A}{(A \to B) \to A \vdash A}}}{\vdash ((A \to B) \to A) \to A}$$

Standard Style

Peirce's Law in Classical Natural Deduction

$$
\dfrac{\dfrac{}{(A \to B) \to A \vdash (A \to B) \to A} \qquad \dfrac{\dfrac{}{A \vdash A, B}}{\vdash A \to B, A}}{\dfrac{(A \to B) \to A \vdash \color{red}{A}, \color{red}{A}}{\dfrac{(A \to B) \to A \vdash \color{red}{A}}{\vdash ((A \to B) \to A) \to A}}}
$$

Standard Style

Peirce's Law in Classical Natural Deduction

$$\dfrac{\dfrac{\dfrac{\dfrac{}{(A \to B) \to A \vdash (A \to B) \to A} \qquad \dfrac{\dfrac{}{A \vdash A, B}}{\vdash A \to B, A}}{(A \to B) \to A \vdash A, A}}{(A \to B) \to A \vdash A}}{\vdash ((A \to B) \to A) \to A}$$

Standard Style

$$\dfrac{\dfrac{\dfrac{\dfrac{}{(A \to B) \to A \vdash (A \to B) \to A} \qquad \dfrac{\dfrac{}{A \vdash A, B}}{\vdash A \to B, A}}{(A \to B) \to A \vdash A, A}}{(A \to B) \to A \vdash A}}{\vdash ((A \to B) \to A) \to A}$$

Standard Style

$$\dfrac{\dfrac{\dfrac{\dfrac{}{(A \to B) \to A \vdash (A \to B) \to A \mid} \qquad \dfrac{\dfrac{\dfrac{}{A \vdash A \mid B}}{A \vdash B \mid A}\,\texttt{act}}{\vdash A \to B \mid A}}{(A \to B) \to A \vdash A \mid A}}{(A \to B) \to A \vdash A \mid}\,\texttt{contrac}}{\vdash ((A \to B) \to A) \to A \mid}$$

Focussed Style

Peirce's Law in Classical Natural Deduction

$$\dfrac{\dfrac{\dfrac{\dfrac{}{(A \to B) \to A \vdash (A \to B) \to A} \qquad \dfrac{\dfrac{}{A \vdash A, B}}{\vdash A \to B, A}}{(A \to B) \to A \vdash A, A}}{(A \to B) \to A \vdash A}}{\vdash ((A \to B) \to A) \to A}$$

Standard Style

$$\dfrac{\dfrac{\dfrac{\dfrac{}{(A \to B) \to A \vdash (A \to B) \to A \mid} \qquad \dfrac{\dfrac{\dfrac{}{A \vdash A \mid {\color{red}B}}}{A \vdash {\color{red}B} \mid A}\,{\color{red}\texttt{act}}}{\vdash A \to B \mid A}}{(A \to B) \to A \vdash A \mid A}\,\texttt{contrac}}{(A \to B) \to A \vdash A \mid}}{\vdash ((A \to B) \to A) \to A \mid}$$

Focussed Style

$$\dfrac{\dfrac{\dfrac{\dfrac{}{(A \to B) \to A \vdash (A \to B) \to A} \qquad \dfrac{\dfrac{}{A \vdash A, B}}{\vdash A \to B, A}}{(A \to B) \to A \vdash A, A}}{(A \to B) \to A \vdash A}}{\vdash ((A \to B) \to A) \to A}$$

Standard Style

$$\dfrac{\dfrac{\dfrac{\dfrac{}{(A \to B) \to A \vdash (A \to B) \to A \mid} \qquad \dfrac{\dfrac{\dfrac{}{A \vdash A \mid B}}{A \vdash B \mid A}\,\texttt{act}}{\vdash A \to B \mid A}}{(A \to B) \to A \vdash A \mid A}}{(A \to B) \to A \vdash A \mid}\,\texttt{contrac}}{\vdash ((A \to B) \to A) \to A \mid}$$

Focussed Style

$$
\begin{prooftree}
\AxiomC{}
\UnaryInfC{$y : [[[A]\to B]\to A] \vdash y : [[A]\to B]\to A \mid$}
\AxiomC{}
\UnaryInfC{$x : [A] \vdash x : A \mid$}
\UnaryInfC{$x : [A] \vdash [\alpha]x : \# \mid \alpha : A$}
\UnaryInfC{$x : [A] \vdash \mu\beta.[\alpha]x : B \mid \alpha : A$}
\UnaryInfC{$\vdash \lambda x.\mu\beta.[\alpha]x : [A] \to B \mid \alpha : A$}
\UnaryInfC{$\Vdash \lambda x.\mu\beta.[\alpha]x : [[A]\to B] \mid \alpha : A$}
\BinaryInfC{$y : [[[A]\to B]\to A] \vdash y(\lambda x.\mu\beta.[\alpha]x) : A \mid \alpha : A$}
\UnaryInfC{$y : [[[A]\to B]\to A] \vdash [\alpha]y(\lambda x.\mu\beta.[\alpha]x) : \# \mid \alpha : \langle A, A\rangle$}
\UnaryInfC{$y : [[[A]\to B]\to A] \vdash \mu\alpha.[\alpha]y(\lambda x.\mu\beta.[\alpha]x) : \langle A, A\rangle \mid$}
\UnaryInfC{$\vdash \lambda y.\mu\alpha.[\alpha]y(\lambda x.\mu\beta.[\alpha]x) : [[[A]\to B]\to A] \to \langle A, A\rangle \mid$}
\end{prooftree}
$$

Typing call–cc in $\mathcal{H}_{\lambda\mu}$

$$\dfrac{\dfrac{\dfrac{\dfrac{}{y : [[[A]\to B]\to A] \vdash y : [[A]\to B]\to A \mid} \qquad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{x : [A] \vdash x : A \mid}}{x : [A] \vdash [\alpha]x : \# \mid \alpha : A}}{x : [A] \vdash \mu\beta.[\alpha]x : B \mid \alpha : A}}{\vdash \lambda x.\mu\beta.[\alpha]x : [A] \to B \mid \alpha : A}}{\Vdash \lambda x.\mu\beta.[\alpha]x : [[A]\to B] \mid \alpha : A}}{y : [[[A]\to B]\to A] \vdash y(\lambda x.\mu\beta.[\alpha]x) : A \mid \alpha : A}}{y : [[[A]\to B]\to A] \vdash [\alpha]y(\lambda x.\mu\beta.[\alpha]x) : \# \mid \alpha : \langle A, A\rangle}}{\dfrac{y : [[[A]\to B]\to A] \vdash \mu\alpha.[\alpha]y(\lambda x.\mu\beta.[\alpha]x) : \langle A, A\rangle \mid}{\vdash \lambda y.\mu\alpha.[\alpha]y(\lambda x.\mu\beta.[\alpha]x) : [[[A]\to B]\to A] \to \langle A, A\rangle \mid}}$$

Infinite formulas are unsound

Let A be *any* formula.
We then set $R_A := (((\ldots) \to A) \to A) \to A$ *i.e.* $R_A = R_A \to A$.

$$
\dfrac{\dfrac{\dfrac{\dfrac{}{R_A \vdash R_A} \quad \dfrac{}{R_A \vdash R_A}}{R_A \vdash A}}{\vdash R_A \to A} \quad \dfrac{\dfrac{\dfrac{}{R_A \vdash R_A} \quad \dfrac{}{R_A \vdash R_A}}{R_A \vdash A}}{\vdash R_A}}{\vdash A}
$$

Infinite formulas are unsound

Let A be *any* formula.
We then set $R_A := (((\ldots) \to A) \to A) \to A$ *i.e.* $R_A = R_A \to A$.

$$\dfrac{\dfrac{\dfrac{\dfrac{}{R_A \vdash R_A \ \textit{i.e.}\ R_A \to A} \quad \dfrac{}{R_A \vdash R_A}}{R_A \vdash A}}{\vdash R_A \to A \ \textit{i.e.}\ R_A} \qquad \dfrac{\dfrac{\dfrac{}{R_A \vdash R_A} \quad \dfrac{}{R_A \vdash R_A}}{R_A \vdash A}}{\vdash R_A}}{\vdash A}$$

Infinite formulas are unsound

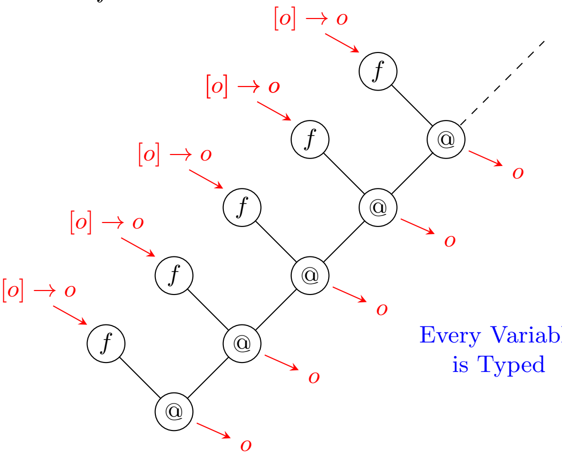
Let A be *any* formula.
We then set $R_A := (((\ldots) \to A) \to A) \to A$ *i.e.* $R_A = R_A \to A$.

$$
\dfrac{
  \dfrac{
    \dfrac{
      \dfrac{}{R_A \vdash R_A}
      \quad
      \dfrac{}{R_A \vdash R_A}
    }{R_A \vdash A}
  }{\vdash R_A \to A}
  \qquad
  \dfrac{
    \dfrac{
      \dfrac{}{R_A \vdash R_A}
      \quad
      \dfrac{}{R_A \vdash R_A}
    }{R_A \vdash A}
  }{\vdash R_A}
}{\vdash A}
$$

Let A be *any* formula.
We then set $R_A := (((\ldots) \to A) \to A) \to A$ *i.e.* $R_A = R_A \to A$.

$$
\dfrac{
\dfrac{
\dfrac{
\dfrac{}{x : R_A \vdash x : R_A}
\qquad
\dfrac{}{x : R_A \vdash x : R_A}
}{x : R_A \vdash x\,x : A}
}{\vdash \lambda x.x\,x : R_A \to A}
\qquad
\dfrac{
\dfrac{
\dfrac{}{R_A \vdash R_A}
\qquad
\dfrac{}{R_A \vdash R_A}
}{R_A \vdash A}
}{\vdash \lambda x.x\,x : R_A}
}{\vdash \Omega : A}
$$

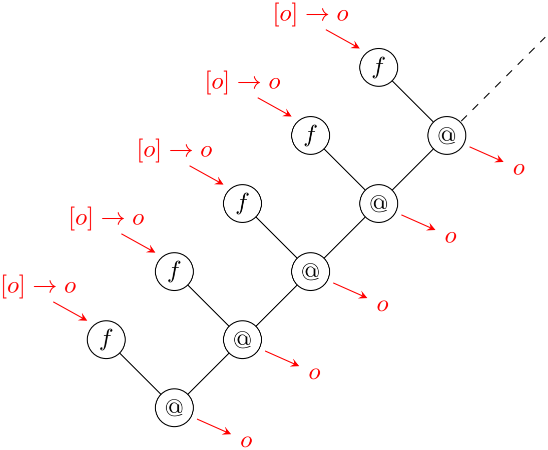
$\Pi' \rhd \Gamma \vdash f^\omega : o$



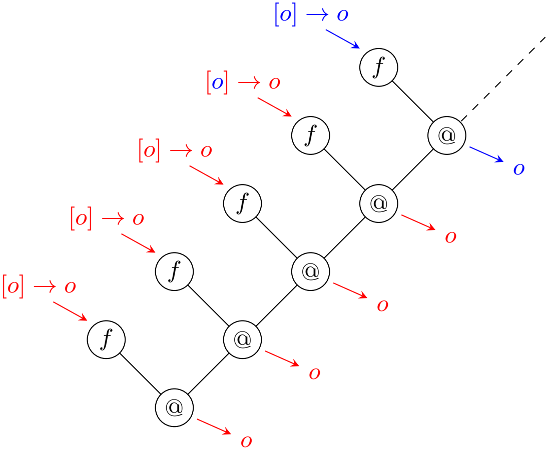
Every Variable
is Typed

$\Gamma = f : [[o] \to o]_\omega$ (infinite multiplicity)

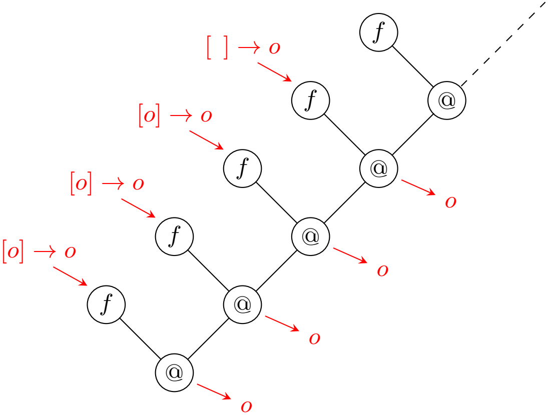
$\Pi' \rhd f : [[o] \to o]_\omega \vdash f^\omega : o$ can be **truncated** into Π'_4



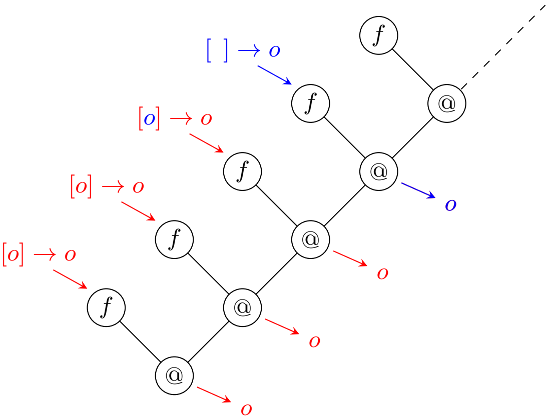
$\Pi' \rhd f : [[o] \to o]_\omega \vdash f^\omega : o$ can be **truncated** into Π'_4



$\Pi' \rhd f : [[o] \to o]_\omega \vdash f^\omega : o$ can be **truncated** into Π'_4

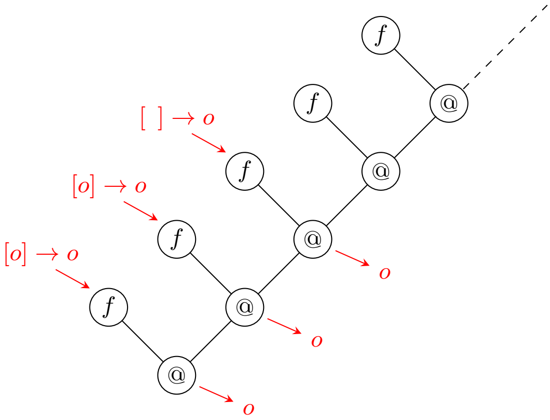


$\Pi' \rhd f : [[o] \to o]_\omega \vdash f^\omega : o$ can be **truncated** into Π'_3

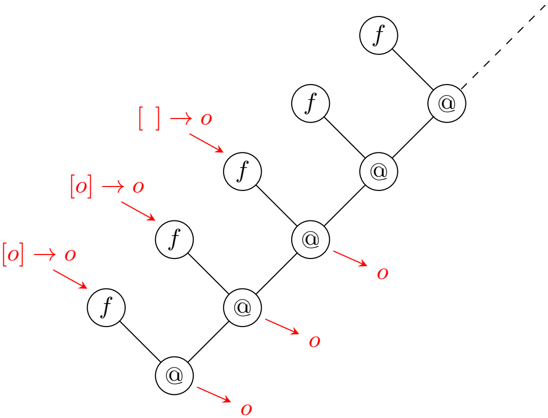


Truncation (Figures)

$\Pi' \rhd f : [[o] \to o]_\omega \vdash f^\omega : o$ can be **truncated** into Π'_3

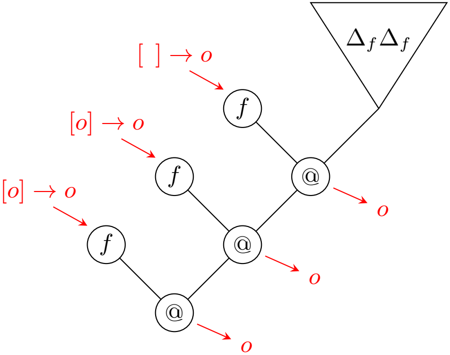


f^ω may be replaced by $f^3(\Delta_f \Delta_f)$ in Π_3', yielding Π_3^3 :

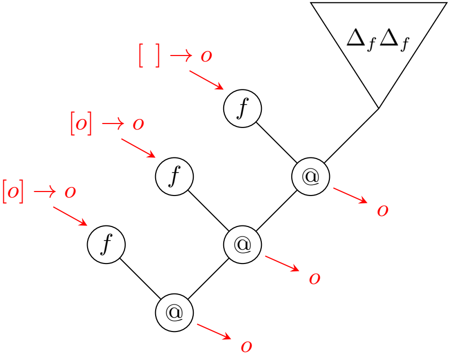


Truncation (Figures)

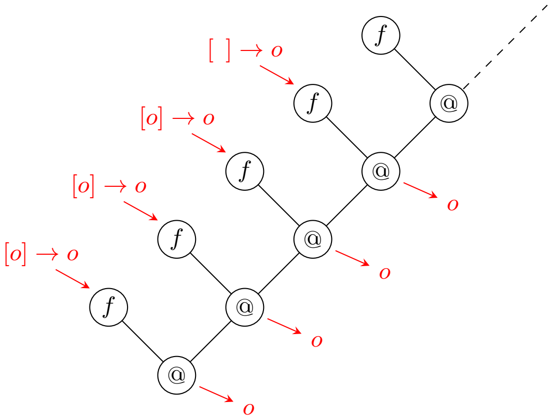
f^ω may be replaced by $f^3(\Delta_f \Delta_f)$ in Π'_3, yielding Π^3_3 :



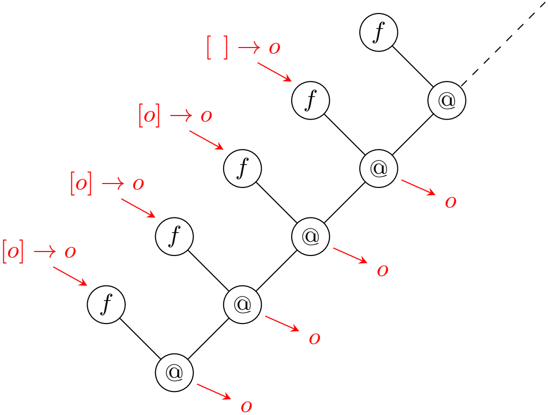
Π_3^3 may be expanded 3 times,
yielding $\Pi_3 \rhd \Delta_f \Delta_f$:



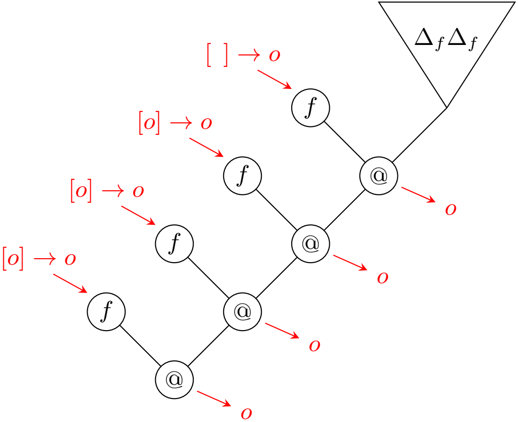
Back to Π'_4, level 4 truncation of Π' :



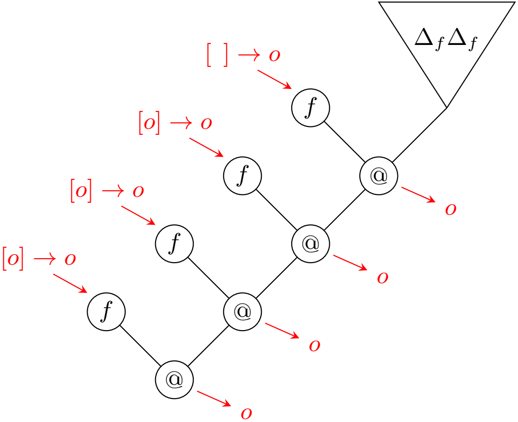
f^ω may be replaced by $f^4(\Delta_f \Delta_f)$ in Π'_3, yielding Π_4^4 :

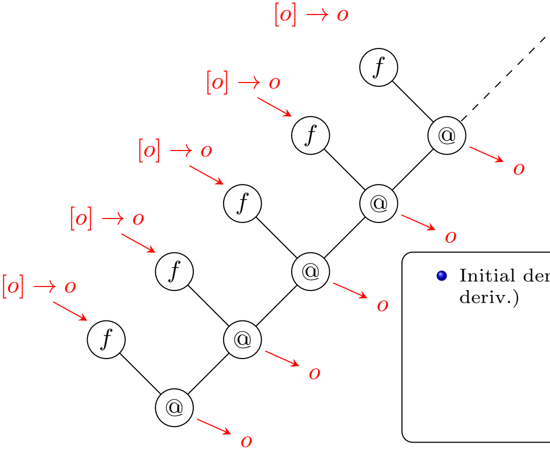


f^ω may be replaced by $f^4(\Delta_f\Delta_f)$ in Π'_3, yielding Π^4_4 :

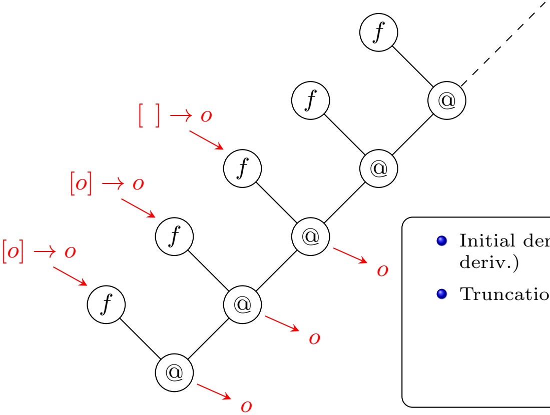


Π_4^4 may be expanded 4 times,
yielding $\Pi_4 \triangleright \Delta_f \Delta_f$:



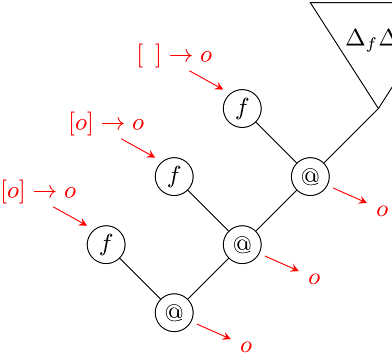


- Initial derivation (inf. term, inf. deriv.)

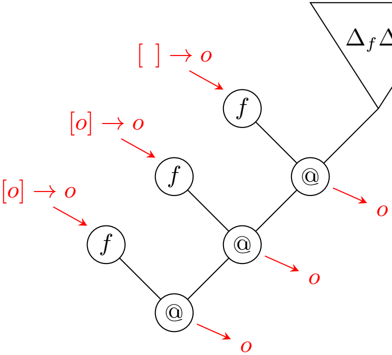


- Initial derivation (inf. term, inf. deriv.)
- Truncation (inf t., f. deriv.)

Truncation (Figures)

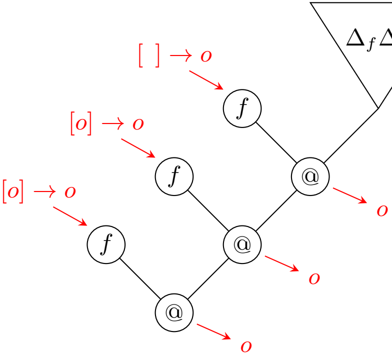


- Initial derivation (inf. term, inf. deriv.)
- Truncation (inf t., f. deriv.)
- Subject subst. (fin. t., fin. d.)



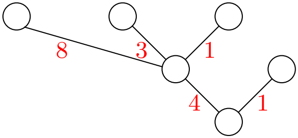
- Initial derivation (inf. term, inf. deriv.)
- Truncation (inf t., f. deriv.)
- Subject subst. (fin. t., fin. d.)
- Expansion ($\rightsquigarrow$ $\mathtt{Y}_f$ is typed)

Truncation (Figures)



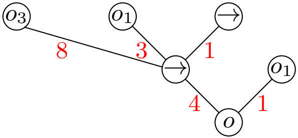
- Initial derivation (inf. term, inf. deriv.)
- Truncation (inf t., f. deriv.)
- Subject subst. (fin. t., fin. d.)
- Expansion ($\rightsquigarrow$ $\mathtt{Y}_f$ is typed)
- Take the join for all trunc.

What is a correct type ?



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

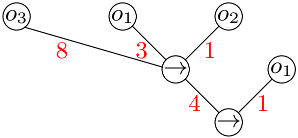
What is a correct type ?



Wrong Labels

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

What is a correct type ?

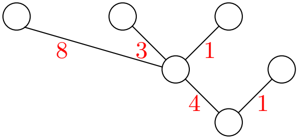


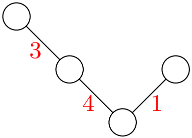
Correct Labels

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

Type: $(4 \cdot (8{\cdot}o_3, 3{\cdot}o_1) \to o_2) \to o_1$

What is a correct type ?

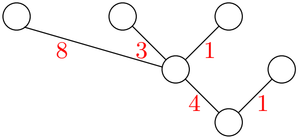


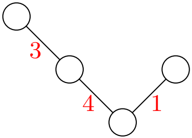


Support:
$\{\varepsilon, 1, 4, 4{\cdot}1, 4{\cdot}3, 4{\cdot}8\}$

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

What is a correct type ?



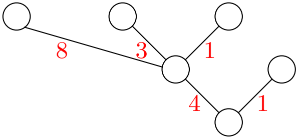


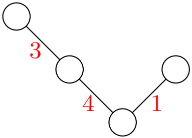
Wrong Support

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

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

What is a correct type ?





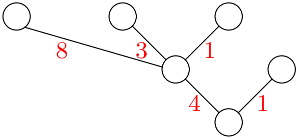
Support:
$\{\varepsilon, 1, 4, 4{\cdot}1, 4{\cdot}3, 4{\cdot}8\}$

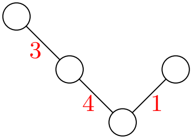
Support:
$\{\varepsilon, 1, 4, 4{\cdot}3\}$

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

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

What is a correct type ?





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

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

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

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

Lemma: Let $C \subseteq \mathbb{N}^*$. Then $\exists T$ type, $C = \mathtt{supp}(T)$ iff $C \neq \emptyset$ and C stable under $\rightarrow_{\mathtt{t1}}$, $\rightarrow_{\mathtt{t2}}$.

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

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

- *Idea:* we try to capture the notion of **bisupport candidate**: a set of pointers that is the bisupport of a $\mathtt{S}$-derivation typing t and have a proposition of the form:

  Proposition: let t be a term and B a set of bipositions. Then, $\exists P$ derivation, $B = \mathtt{bisupp}(P)$ iff $B \neq \emptyset$ and B stable under $\to_1$, $\to_2$, $\to_3$,... [see Prop. 12.3, p. 260]

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

  Proposition: let t be a term and B a set of bipositions. Then, $\exists P$ derivation, $B = \mathtt{bisupp}(P)$ iff $B \neq \emptyset$ and B stable under $\rightarrow_1, \rightarrow_2, \rightarrow_3, \ldots$ [see Prop. 12.3, p. 260]

- We must find suitable stability conditions.

Bisupport Candidates

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

- *Idea:* we try to capture the notion of **bisupport candidate**: a set of pointers that is the bisupport of a $\mathtt{S}$-derivation typing t and have a proposition of the form:

  Proposition: let t be a term and B a set of bipositions. Then, $\exists P$ derivation, $B = \mathtt{bisupp}(P)$ iff $B \neq \emptyset$ and B stable under $\rightarrow_1$, $\rightarrow_2$, $\rightarrow_3$,... [see Prop. 12.3, p. 260]

- We must find suitable stability conditions.

- Then, we show that there is actually a *non-empty* set that satisfies them.

Guidelines of the proof

- Reduce the problem ("every term is $\mathtt{S}$-typable") to a parametrized first order theory $\mathcal{T}_t$ ($t \in \Lambda$).

- Establish a "completeness-like" property:

  Prop.: let $t \in \Lambda$. Then t is $\mathtt{S}$-typable iff $\mathcal{T}_t$ is consistent.

- How do we prove that $\mathcal{T}_t$ cannot be contradictory?

Guidelines of the proof

- Reduce the problem ("every term is $\mathtt{S}$-typable") to a parametrized first order theory $\mathcal{T}_t$ ($t \in \Lambda$).

- Establish a "completeness-like" property:

  Prop.: let $t \in \Lambda$. Then t is $\mathtt{S}$-typable iff $\mathcal{T}_t$ is consistent.

- How do we prove that $\mathcal{T}_t$ cannot be contradictory?
  1. Assume *ad absurdum* that $\mathcal{T}_t$ is contradictory for some t. Then, there is a finite proof $\mathcal{C}$ (standing for **chain**) that $\mathcal{T}_t$ is contradictory.
  2. If $\mathcal{C}$ "visits" redexes, $\mathcal{C}$ is not decypherable. But we cannot eliminate redexes in all generality (*e.g.*, in mute terms). What can we do?

Guidelines of the proof

- Reduce the problem ("every term is $\mathtt{S}$-typable") to a parametrized first order theory $\mathcal{T}_t$ $(t \in \Lambda)$.

- Establish a "completeness-like" property:

  Prop.: let $t \in \Lambda$. Then t is $\mathtt{S}$-typable iff $\mathcal{T}_t$ is consistent.

- How do we prove that $\mathcal{T}_t$ cannot be contradictory?
  1. Assume *ad absurdum* that $\mathcal{T}_t$ is contradictory for some t. Then, there is a finite proof $\mathcal{C}$ (standing for **chain**) that $\mathcal{T}_t$ is contradictory.
  2. If $\mathcal{C}$ "visits" redexes, $\mathcal{C}$ is not decypherable. But we cannot eliminate redexes in all generality (*e.g.*, in mute terms). What can we do?
  3. *Fundamental idea:* There is a finite reduction strategy (called the **collapsing strategy**) $t \to t'$ such that $\mathcal{C}$ can be *residuated* into a chain $\mathcal{C}'$ of t' that does not interact with redex ($\mathcal{C}'$ is called a **normal chain**).
  4. We prove that $\mathcal{C}'$ cannot exist. So $\mathcal{C}$ does not either *i.e.* there is not proof of contradiction.
  5. Thus, $\mathcal{T}_t$ is consistent!

Guidelines of the proof

- Reduce the problem ("every term is $\mathtt{S}$-typable") to a parametrized first order theory $\mathcal{T}_t$ ($t \in \Lambda$).

- Establish a "completeness-like" property:

  Prop.: let $t \in \Lambda$. Then t is $\mathtt{S}$-typable iff $\mathcal{T}_t$ is consistent.

- How do we prove that $\mathcal{T}_t$ cannot be contradictory?
  1. Assume *ad absurdum* that $\mathcal{T}_t$ is contradictory for some t. Then, there is a finite proof $\mathcal{C}$ (standing for **chain**) that $\mathcal{T}_t$ is contradictory.
  2. If $\mathcal{C}$ "visits" redexes, $\mathcal{C}$ is not decypherable. But we cannot eliminate redexes in all generality (*e.g.*, in mute terms). What can we do?
  3. *Fundamental idea:* There is a finite reduction strategy (called the **collapsing strategy**) $t \to t'$ such that $\mathcal{C}$ can be *residuated* into a chain $\mathcal{C}'$ of t' that does not interact with redex ($\mathcal{C}'$ is called a **normal chain**).
  4. We prove that $\mathcal{C}'$ cannot exist. So $\mathcal{C}$ does not either *i.e.* there is not proof of contradiction.
  5. Thus, $\mathcal{T}_t$ is consistent!

- *Remark:* works for the infinitary λ-calculus!

Order

Theorem (complete unsoundness): in $\mathscr{R}$, every term is typable.
[Th 12.1, p. 276]

Theorem (complete unsoundness): in $\mathscr{R}$, every term is typable.
[Th 12.1, p. 276]

Theorem: if t is a zero-term, then, t is typable with o.
[Th 12.2, p. 276]

Theorem (complete unsoundness): in $\mathscr{R}$, every term is typable.
[Th 12.1, p. 276]

Theorem: if t is a zero-term, then, t is typable with o.
[Th 12.2, p. 276]

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

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

Theorem (complete unsoundness): in $\mathscr{R}$, every term is typable.
[Th 12.1, p. 276]

Theorem: if t is a zero-term, then, t is typable with o.
[Th 12.2, p. 276]

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

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

Corollary: This yields a *non-sensible* model that discriminates terms according to their order:

if t and u are two terms of different orders, then $[\![t]\!] \neq [\![u]\!]$.

First model to do this!