Quantitative Types for Higher-Order Languages

Delia KESNER

(kesner@irif.fr)

IRIF, CNRS and Université Paris-Diderot, France

Introduction

- Traditional type systems use simple types.
- Simple types guarantee (some) operational properties.
- Simple types do not characterize operational properties.
- They were refined to intersection type systems then to quantitative type systems.
- Quantitative type systems provide a clean theoretical understanding of the use of resources (like execution time and space).
- Quantitative type systems provide simple arithmetical characterizations of different operational properties (e.g. a term t is typable if and only if t is head-normalizing).

The Case of the λ-Calculus

Background

- Introduced by **Church'30**.
- Natural model of computation.
- Turing complete.
- Base of functional languages (call-by-name, call-by-value, call-by-need).
- Curry-Howard interpretation of intuitionistic logic.
- Admits powerful extensions (classical logic, type theory, etc).
- Base of modern proof assistants.

The λ-calculus

Syntax:

$$
\begin{array}{lll}
t, u & ::= & \\
 & x & \text{(variable) |} \\
 & \lambda x.t & \text{(astraction) |} \\
 & tu & \text{(application)}
\end{array}
$$

Operational Semantics: $(\lambda x.t)u \rightarrow_\beta t\{x/u\}$

Examples: Let $\mathtt{I} = \lambda z.z$.

$$
\begin{array}{llcll}
\text{(Erasing)} & (\lambda x.y)\mathtt{I} & \rightarrow & y & \\
\text{(Linear)} & (\lambda x.x)\mathtt{I} & \rightarrow & I & \\
\text{(Duplicating)} & (\lambda x.xx)\mathtt{I} & \rightarrow & \mathtt{I}\,\mathtt{I} & \\
\text{(Non-terminating)} & (\lambda x.xx)(\lambda x.xx) & \rightarrow & (\lambda x.xx)(\lambda x.xx) & \ldots
\end{array}
$$

Simple Types for The Lambda-Calculus

Simple Types: basic types (`int`, `bool`, etc) and functional types (constructor $\Rightarrow$).

Examples:

$\Gamma \vdash \lambda x.y : \texttt{int} \Rightarrow \texttt{int}$

$\Gamma \vdash \lambda x \lambda y.yxx : \texttt{int} \Rightarrow (\texttt{int} \Rightarrow \texttt{int} \Rightarrow \texttt{bool}) \Rightarrow \texttt{bool}$

$\Gamma \vdash \lambda x.xx : ??$

- Typical Curry-Howard interpretation
- Minimal monomorphic information
- Typability implies Normalization
- Normalization does not imply Typability

Simple Types for The Lambda-Calculus

Simple Types: basic types (`int`, `bool`, etc) and functional types (constructor $\Rightarrow$).

Examples:

$\Gamma \vdash \lambda x.y : \texttt{int} \Rightarrow \texttt{int}$

$\Gamma \vdash \lambda x \lambda y.yxx : \texttt{int} \Rightarrow (\texttt{int} \Rightarrow \texttt{int} \Rightarrow \texttt{bool}) \Rightarrow \texttt{bool}$

$\Gamma \vdash \lambda x.xx : ??$

- Typical Curry-Howard interpretation
- Minimal monomorphic information
- Typability implies Normalization
- Normalization does not imply Typability

Intersection Types

Intersection Types: basic types, functional types ($\Rightarrow$) and intersection types ($\wedge$).

- Provides polymorphism by listing type instances

Simple types	$x : ?? \vdash xx : \mathtt{B}$
Intersection Types	$x : (\mathtt{A} \to \mathtt{B}) \wedge \mathtt{A} \vdash xx : \mathtt{B}$

- Powerful tool to obtain (qualitative) characterizations of operational properties.

  Typically,

  Typability if and only if Normalization.

  Soundness ("Typability implies Normalization") can be difficult to prove.

Intersection Types

Intersection Types: basic types, functional types ($\Rightarrow$) and intersection types ($\wedge$).

- Provides polymorphism by listing type instances

Simple types	$x : ?? \vdash xx : \mathtt{B}$
Intersection Types	$x : (\mathtt{A} \to \mathtt{B}) \wedge \mathtt{A} \vdash xx : \mathtt{B}$

- Powerful tool to obtain (qualitative) characterizations of operational properties.

  Typically,

  Typability if and only if Normalization.

  Soundness ("Typability implies Normalization") can be difficult to prove.

From Intersection Types to Quantitative Types

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

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

From Intersection Types to Quantitative Types

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

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

Non-Idempotent Types

- Linear Logic flavour (**Girard'94**). In particular, use of a quantitative refinement of Girard's translation $A \Rightarrow B := !A \multimap B$.
- Quantitative/relational models for λ-calculi (**de Carvalho**, **Ehrhard**, . . .).
- Characterization of operational properties (solvability, head-normalization, weak-normalization, strong-normalization, linear-head normalization, etc).
  - Soundness properties "typability implies normalization" are proved by arithmetical arguments based on weighted subject reduction properties.
  - Completeness properties "normalization implies typability" are proved by subject expansion properties.
- Observational equivalence between different languages (**Kesner**).
- Consumption of ressources can be measured and bound (**de Carvalho**, **Bernadet-Lengrand**, **De Benedetti-Ronchi Della Rocca**).
- Inhabitation becomes decidable/bound (**Bucciarelli-Kesner-Ronchi Della Rocca**, **Dudenhefner-Rehof**).

Using Non-Idempotent Types

- Characterization of head-normalization.
- Characterization of strong-normalization.
- Other characterizations.
- Deriving resource aware semantics for higher-order languages.
- Observational equivalence between different languages.
- Synthesis of programs (inhabitation).
- Exact bounds for reduction sequences.

A First Characterization Result

(head-normalization)

Relating Typability and (Head)Normalization

- A head-normal form is a term $\lambda x_1 \ldots \lambda x_m.yt_1 \ldots, t_n$ (y is a variable and $n, m \geq 0$).
- A term t is head-normalizing if and only if t reduces to a head-normal form.
- The head-strategy only reduces head-redexes of the form

  $\lambda x_1 \ldots \lambda x_m.(\lambda y.t)ut_1 \ldots, t_n \rightarrow_H \lambda x_1 \ldots \lambda x_m.t\{x/u\}t_1 \ldots, t_n$.

Examples: Let $\Omega = (\lambda x.xx)(\lambda x.xx)$.

$(\lambda x.xy\Omega)(\lambda z.z)$ is head-normalizing (and typable):

$$(\lambda x.xy\Omega)(\lambda z.z) \rightarrow^* y\Omega$$

$\Omega = (\lambda x.xx)(\lambda x.xx)$ is not head-normalizing (and not typable):

$$\Omega \rightarrow (\lambda x.xx)(\lambda x.xx) \rightarrow \ldots$$

Is there any relation between typability and head-normalization?

Relating Typability and (Head)Normalization

- A head-normal form is a term $\lambda x_1 \ldots \lambda x_m.yt_1 \ldots, t_n$ (y is a variable and $n, m \geq 0$).
- A term t is head-normalizing if and only if t reduces to a head-normal form.
- The head-strategy only reduces head-redexes of the form

  $\lambda x_1 \ldots \lambda x_m.(\lambda y.t)ut_1 \ldots, t_n \rightarrow_H \lambda x_1 \ldots \lambda x_m.t\{x/u\}t_1 \ldots, t_n.$

Examples: Let $\Omega = (\lambda x.xx)(\lambda x.xx)$.

$(\lambda x.xy\Omega)(\lambda z.z)$ is head-normalizing (and typable):

$$(\lambda x.xy\Omega)(\lambda z.z) \rightarrow^* y\Omega$$

$\Omega = (\lambda x.xx)(\lambda x.xx)$ is not head-normalizing (and not typable):

$$\Omega \rightarrow (\lambda x.xx)(\lambda x.xx) \rightarrow \ldots$$

Is there any relation between typability and head-normalization?

Relating Typability and (Head)Normalization

- A head-normal form is a term $\lambda x_1 \ldots \lambda x_m.yt_1 \ldots, t_n$ (y is a variable and $n, m \geq 0$).
- A term t is head-normalizing if and only if t reduces to a head-normal form.
- The head-strategy only reduces head-redexes of the form

  $\lambda x_1 \ldots \lambda x_m.(\lambda y.t)ut_1 \ldots, t_n \rightarrow_H \lambda x_1 \ldots \lambda x_m.t\{x/u\}t_1 \ldots, t_n.$

Examples: Let $\Omega = (\lambda x.xx)(\lambda x.xx)$.

$(\lambda x.xy\Omega)(\lambda z.z)$ is head-normalizing (and typable):

$$(\lambda x.xy\Omega)(\lambda z.z) \rightarrow^* y\Omega$$

$\Omega = (\lambda x.xx)(\lambda x.xx)$ is not head-normalizing (and not typable):

$$\Omega \rightarrow (\lambda x.xx)(\lambda x.xx) \rightarrow \ldots$$

Is there any relation between typability and head-normalization?

The case of the λ-calculus

Types:

(Strict Types)	σ	::=	a (basic type) $\mid \mathtt{A} \Rightarrow \sigma$
(Intersection Types)	$\mathtt{A}$	::=	$[\sigma_k]_{k \in K}$ (K finite)

Tools:

- A **variable assignment** is a function Γ from variables to intersection types, in particular, if $x \notin \mathrm{dom}(\Gamma)$, then $\Gamma(x)$ gives $[\,]$.
- **Intersection of assignments**: $(\Gamma \wedge \Delta)(x) := \Gamma(x) \sqcup \Delta(x)$.
- **Regular Judgments** are 3-tuples of the form $\Gamma \vdash t : \sigma$
- **Auxiliary Judgments** are 3-tuples of the form $\Gamma \Vdash t : \mathtt{A}$.

The case of the λ-calculus

Types:

$$\begin{array}{lcll} \text{(Strict Types)} & \sigma & ::= & a \text{ (basic type)} \mid \mathtt{A} \Rightarrow \sigma \\ \text{(Intersection Types)} & \mathtt{A} & ::= & [\sigma_k]_{k \in K} \ (K \text{ finite}) \end{array}$$

Tools:

- A **variable assignment** is a function Γ from variables to intersection types, in particular, if $x \notin \mathtt{dom}(\Gamma)$, then $\Gamma(x)$ gives $[\ ]$.
- **Intersection of assignments**: $(\Gamma \wedge \Delta)(x) := \Gamma(x) \sqcup \Delta(x)$.
- **Regular Judgments** are 3-tuples of the form $\Gamma \vdash t : \sigma$
- **Auxiliary Judgments** are 3-tuples of the form $\Gamma \Vdash t : \mathtt{A}$.

A First Non-Idempotent Typing System for the λ-Calculus

$$\frac{}{x : [\sigma] \vdash x : \sigma}\ (\mathtt{ax}) \qquad \frac{(\Gamma_k \vdash t : \sigma_k)_{k\in K}}{\wedge_{k\in K}\Gamma_k \Vdash t : [\sigma_k]_{k\in K}}\ (\mathtt{many})$$

$$\frac{\Gamma \vdash t : \sigma}{\Gamma \setminus\setminus x \vdash \lambda x.t : \Gamma(x) \Rightarrow \sigma}\ (\rightarrow_{\mathtt{i}}) \qquad \frac{\Gamma \vdash t : \mathtt{A} \Rightarrow \sigma \qquad \Delta \Vdash \mathtt{u} : \mathtt{A}}{\Gamma \wedge \Delta \vdash tu : \sigma}\ (\rightarrow_{\mathtt{e}})$$

Observations:

- The axiom is relevant (no weakening).
- The $(\rightarrow_{\mathtt{e}})$ rule is multiplicative.
- The case $K = 0$ in rule $(\mathtt{many})$ gives an auxiliary subderivation $\emptyset \Vdash t : [\,]$ for any term t, which denotes an untyped term.
- The system is syntax-directed.

A First Non-Idempotent Typing System for the λ-Calculus

$$\frac{}{x : [\sigma] \vdash x : \sigma}\ (\mathtt{ax}) \qquad \frac{(\Gamma_k \vdash t : \sigma_k)_{k\in K}}{\wedge_{k\in K}\Gamma_k \Vdash t : [\sigma_k]_{k\in K}}\ (\mathtt{many})$$

$$\frac{\Gamma \vdash t : \sigma}{\Gamma \setminus\setminus x \vdash \lambda x.t : \Gamma(x) \Rightarrow \sigma}\ (\rightarrow_{\mathtt{i}}) \qquad \frac{\Gamma \vdash t : \mathtt{A} \Rightarrow \sigma \qquad \Delta \Vdash \mathtt{u} : \mathtt{A}}{\Gamma \wedge \Delta \vdash tu : \sigma}\ (\rightarrow_{\mathtt{e}})$$

Observations:

- The axiom is relevant (no weakening).
- The $(\rightarrow_{\mathtt{e}})$ rule is multiplicative.
- The case $K = 0$ in rule $(\mathtt{many})$ gives an auxiliary subderivation $\emptyset \Vdash t : [\,]$ for any term t, which denotes an untyped term.
- The system is syntax-directed.

Examples (I)

$$\dfrac{\dfrac{}{x : [[a] \Rightarrow a] \vdash x : [a] \Rightarrow a}\,(\mathtt{ax}) \qquad \dfrac{\dfrac{}{x : [a] \vdash x : a}\,(\mathtt{ax})}{x : [a] \Vdash x : [a]}\,(\mathtt{many})}{\dfrac{x : [[a] \Rightarrow a, a] \vdash xx : a}{\vdash \lambda x.xx : [[a] \Rightarrow a, a] \Rightarrow a}\,(\rightarrow_{\mathtt{i}})}\,(\rightarrow_{\mathtt{e}})$$

$$\dfrac{\dfrac{\dfrac{}{y : [a] \vdash y : a}\,(\mathtt{ax})}{y : [a] \vdash \lambda x.y : [\,] \Rightarrow a}\,(\rightarrow_{\mathtt{i}}) \qquad \dfrac{}{\emptyset \Vdash z : [\,]}\,(\mathtt{many})}{y : [a] \vdash (\lambda x.y)z : a}\,(\rightarrow_{\mathtt{e}})$$

Examples (I)

- $$\dfrac{\dfrac{}{x : [[a] \Rightarrow a] \vdash x : [a] \Rightarrow a}\,(\mathtt{ax}) \qquad \dfrac{\dfrac{}{x : [a] \vdash x : a}\,(\mathtt{ax})}{x : [a] \Vdash x : [a]}\,(\mathtt{many})}{\dfrac{x : [[a] \Rightarrow a, a] \vdash xx : a}{\vdash \lambda x.xx : [[a] \Rightarrow a, a] \Rightarrow a}\,(\rightarrow_{\mathtt{i}})}\,(\rightarrow_{\mathtt{e}})$$

- $$\dfrac{\dfrac{\dfrac{}{y : [a] \vdash y : a}\,(\mathtt{ax})}{y : [a] \vdash \lambda x.y : [\,] \Rightarrow a}\,(\rightarrow_{\mathtt{i}}) \qquad \dfrac{}{\emptyset \Vdash z : [\,]}\,(\mathtt{many})}{y : [a] \vdash (\lambda x.y)z : a}\,(\rightarrow_{\mathtt{e}})$$

Examples (II)

Let $\tau = [a] \Rightarrow a$

$$\dfrac{\vdash \lambda x.xx : [\tau, a] \Rightarrow a \qquad \dfrac{\dfrac{}{z : [\tau] \vdash z : \tau}\,(\mathtt{ax}) \qquad \dfrac{}{z : [a] \vdash z : a}\,(\mathtt{ax})}{z : [\tau, a] \Vdash z : [\tau, a]}}{\vdash (\lambda x.xx)(\lambda z.z) : a}\,(\rightarrow_{\mathtt{i}})$$

Properties of the Type System

- **Weighted Subject Reduction**:
  - Reduction preserves types and environments:
    If $\Pi \triangleright \Gamma \vdash t : \sigma$ and $t \to_H t'$, then $\Pi' \triangleright \Gamma \vdash t' : \sigma$.
  - Reduction strictly decreases the number of nodes of tree type derivations:
    If $\Pi \triangleright \Gamma \vdash t : \sigma$ and $t \to_H t'$, then also $\#(\Pi) > \#(\Pi')$.
- **Subject Expansion**:
  - Anti-reduction preserves types and environments:
    If $\Pi \triangleright \Gamma \vdash t' : \sigma$ and $t \to_H t'$, then $\Pi' \triangleright \Gamma \vdash t : \sigma$.

Corollary [**Characterization of Head Normalization**]:
Let t be a λ-term. Then t is typable if and only if t is head-normalizing.
Moreover, if Π is a type derivation of t, then $\#(\Pi)$ gives an upper bound to the length of the head-strategy starting at t.

Properties of the Type System

- **Weighted Subject Reduction**:
  - Reduction preserves types and environments:
    If $\Pi \triangleright \Gamma \vdash t : \sigma$ and $t \to_H t'$, then $\Pi' \triangleright \Gamma \vdash t' : \sigma$.
  - Reduction strictly decreases the number of nodes of tree type derivations:
    If $\Pi \triangleright \Gamma \vdash t : \sigma$ and $t \to_H t'$, then also $\#(\Pi) > \#(\Pi')$.
- **Subject Expansion**:
  - Anti-reduction preserves types and environments:
    If $\Pi \triangleright \Gamma \vdash t' : \sigma$ and $t \to_H t'$, then $\Pi' \triangleright \Gamma \vdash t : \sigma$.

Corollary [**Characterization of Head Normalization**]:

Let t be a λ-term. Then t is typable if and only if t is head-normalizing.

Moreover, if Π is a type derivation of t, then $\#(\Pi)$ gives an upper bound to the length of the head-strategy starting at t.

Properties of the Type System

- **Weighted Subject Reduction**:
  - Reduction preserves types and environments:

    If $\Pi \triangleright \Gamma \vdash t : \sigma$ and $t \to_H t'$, then $\Pi' \triangleright \Gamma \vdash t' : \sigma$.
  - Reduction strictly decreases the number of nodes of tree type derivations:

    If $\Pi \triangleright \Gamma \vdash t : \sigma$ and $t \to_H t'$, then also $\#(\Pi) > \#(\Pi')$.
- **Subject Expansion**:
  - Anti-reduction preserves types and environments:

    If $\Pi \triangleright \Gamma \vdash t' : \sigma$ and $t \to_H t'$, then $\Pi' \triangleright \Gamma \vdash t : \sigma$.

Corollary [**Characterization of Head Normalization**]:

Let t be a λ-term. Then t is typable if and only if t is head-normalizing.

Moreover, if Π is a type derivation of t, then $\#(\Pi)$ gives an upper bound to the length of the head-strategy starting at t.

Properties of the Type System

- **Weighted Subject Reduction**:
  - Reduction preserves types and environments:
    If $\Pi \triangleright \Gamma \vdash t : \sigma$ and $t \to_H t'$, then $\Pi' \triangleright \Gamma \vdash t' : \sigma$.
  - Reduction strictly decreases the number of nodes of tree type derivations:
    If $\Pi \triangleright \Gamma \vdash t : \sigma$ and $t \to_H t'$, then also $\#(\Pi) > \#(\Pi')$.
- **Subject Expansion**:
  - Anti-reduction preserves types and environments:
    If $\Pi \triangleright \Gamma \vdash t' : \sigma$ and $t \to_H t'$, then $\Pi' \triangleright \Gamma \vdash t : \sigma$.

Corollary [**Characterization of Head Normalization**]:

Let t be a λ-term. Then t is typable if and only if t is head-normalizing.

Moreover, if Π is a type derivation of t, then $\#(\Pi)$ gives an upper bound to the length of the head-strategy starting at t.

A Second Characterization Result

(Strong Normalization)

Relating Typing and (Strong)Normalization

A term t is strongly-normalizing iff there is no infinite reduction sequence starting at t.

Equivalently, a term t is strongly-normalizing iff every reduction sequence starting at t terminates.

- Strong-normalization implies head-normalization.
- $(\lambda x.xy\Omega)(\lambda z.z)$ is head-normalizing but not strongly-normalizing.

Examples:

$(\lambda x.xyw)(\lambda z.z)$ is strongly-normalizing (and typable):

$$(\lambda x.xyw)(\lambda z.z) \rightarrow^* yw$$

$\Omega = (\lambda x.xx)(\lambda x.xx)$ is not strongly-normalizing (and not typable):

$$\Omega \rightarrow \Omega \rightarrow \ldots$$

Is there any relation between typability and strong-normalization?

Relating Typing and (Strong)Normalization

A term t is strongly-normalizing iff there is no infinite reduction sequence starting at t.

Equivalently, a term t is strongly-normalizing iff every reduction sequence starting at t terminates.

- Strong-normalization implies head-normalization.
- $(\lambda x.xy\Omega)(\lambda z.z)$ is head-normalizing but not strongly-normalizing.

Examples:

$(\lambda x.xyw)(\lambda z.z)$ is strongly-normalizing (and typable):

$$(\lambda x.xyw)(\lambda z.z) \rightarrow^* yw$$

$\Omega = (\lambda x.xx)(\lambda x.xx)$ is not strongly-normalizing (and not typable):

$$\Omega \rightarrow \Omega \rightarrow \ldots$$

Is there any relation between typability and strong-normalization?

Relating Typing and (Strong)Normalization

A term t is strongly-normalizing iff there is no infinite reduction sequence starting at t.

Equivalently, a term t is strongly-normalizing iff every reduction sequence starting at t terminates.

- Strong-normalization implies head-normalization.
- $(\lambda x.xy\Omega)(\lambda z.z)$ is head-normalizing but not strongly-normalizing.

Examples:

$(\lambda x.xyw)(\lambda z.z)$ is strongly-normalizing (and typable):

$$(\lambda x.xyw)(\lambda z.z) \rightarrow^* yw$$

$\Omega = (\lambda x.xx)(\lambda x.xx)$ is not strongly-normalizing (and not typable):

$$\Omega \rightarrow \Omega \rightarrow \ldots$$

Is there any relation between typability and strong-normalization?

Relating Typing and (Strong)Normalization

A term t is strongly-normalizing iff there is no infinite reduction sequence starting at t.

Equivalently, a term t is strongly-normalizing iff every reduction sequence starting at t terminates.

- Strong-normalization implies head-normalization.
- $(\lambda x.xy\Omega)(\lambda z.z)$ is head-normalizing but not strongly-normalizing.

Examples:

$(\lambda x.xyw)(\lambda z.z)$ is strongly-normalizing (and typable):

$$(\lambda x.xyw)(\lambda z.z) \rightarrow^* yw$$

$\Omega = (\lambda x.xx)(\lambda x.xx)$ is not strongly-normalizing (and not typable):

$$\Omega \rightarrow \Omega \rightarrow \ldots$$

Is there any relation between typability and strong-normalization?

A Second First Non-Idempotent Typing System for the λ-Calculus

$$\frac{}{x : [\sigma] \vdash x : \sigma}\ (\mathtt{ax}) \qquad \frac{(\Gamma_k \vdash t : \sigma_k)_{k\in K}}{\wedge_{k\in K}\Gamma_k \Vdash t : [\sigma_k]_{k\in K}}\ (\mathtt{many})$$

$$\frac{\Gamma \vdash t : \sigma}{\Gamma \setminus\!\setminus x \vdash \lambda x.t : \Gamma(x) \Rightarrow \sigma}\ (\rightarrow_{\mathtt{i}}) \qquad \frac{\Gamma \vdash t : \mathtt{A} \Rightarrow \sigma \qquad \Delta \Vdash \mathtt{u} : \mathtt{A}^*}{\Gamma \wedge \Delta \vdash tu : \sigma}\ (\rightarrow_{\mathtt{e}})$$

where the choice operator is defined by:

$$\mathtt{A}^* := \begin{cases} [\sigma] & \text{if } \mathtt{A} = [\,] \text{ and } \sigma \text{ is an } \textbf{arbitrary} \text{ strict type} \\ \mathtt{A} & \text{if } \mathtt{A} \neq [\,] \end{cases}.$$

Observations:

- No auxiliary subderivations like $\emptyset \Vdash \Omega : [\,]$ inside type derivations.
- The choice operator gives a sort of weakening.

A Second First Non-Idempotent Typing System for the λ-Calculus

$$\frac{}{x : [\sigma] \vdash x : \sigma}\ (\mathtt{ax}) \qquad \frac{(\Gamma_k \vdash t : \sigma_k)_{k\in K}}{\wedge_{k\in K}\Gamma_k \Vdash t : [\sigma_k]_{k\in K}}\ (\mathtt{many})$$

$$\frac{\Gamma \vdash t : \sigma}{\Gamma \setminus\setminus x \vdash \lambda x.t : \Gamma(x) \Rightarrow \sigma}\ (\rightarrow_{\mathtt{i}}) \qquad \frac{\Gamma \vdash t : \mathtt{A} \Rightarrow \sigma \qquad \Delta \Vdash \mathtt{u} : \mathtt{A}^*}{\Gamma \wedge \Delta \vdash tu : \sigma}\ (\rightarrow_{\mathtt{e}})$$

where the choice operator is defined by:

$$\mathtt{A}^* := \begin{cases} [\sigma] & \text{if } \mathtt{A} = [\,] \text{ and } \sigma \text{ is an } \textbf{arbitrary} \text{ strict type} \\ \mathtt{A} & \text{if } \mathtt{A} \neq [\,] \end{cases} .$$

Observations:

- No auxiliary subderivations like $\emptyset \Vdash \Omega : [\,]$ inside type derivations.
- The choice operator gives a sort of weakening.

A Second First Non-Idempotent Typing System for the λ-Calculus

$$\frac{}{x : [\sigma] \vdash x : \sigma}\ (\mathtt{ax}) \qquad \frac{(\Gamma_k \vdash t : \sigma_k)_{k\in K}}{\wedge_{k\in K}\Gamma_k \Vdash t : [\sigma_k]_{k\in K}}\ (\mathtt{many})$$

$$\frac{\Gamma \vdash t : \sigma}{\Gamma \setminus\!\setminus x \vdash \lambda x.t : \Gamma(x) \Rightarrow \sigma}\ (\rightarrow_{\mathtt{i}}) \qquad \frac{\Gamma \vdash t : \mathtt{A} \Rightarrow \sigma \qquad \Delta \Vdash \mathtt{u} : \mathtt{A}^*}{\Gamma \wedge \Delta \vdash tu : \sigma}\ (\rightarrow_{\mathtt{e}})$$

where the choice operator is defined by:

$$\mathtt{A}^* := \begin{cases} [\sigma] & \text{if } \mathtt{A} = [\,] \text{ and } \sigma \text{ is an } \textbf{arbitrary} \text{ strict type} \\ \mathtt{A} & \text{if } \mathtt{A} \neq [\,] \end{cases} .$$

Observations:

- No auxiliary subderivations like $\emptyset \Vdash \Omega : [\,]$ inside type derivations.
- The choice operator gives a sort of weakening.

Example

$$\dfrac{\dfrac{\dfrac{}{y : [a] \vdash y : a}\,(\mathtt{ax})}{y : [a] \vdash \lambda x.y : [\ ] \Rightarrow a}\,(\rightarrow_{\mathtt{i}}) \qquad \dfrac{\dfrac{}{z : [b] \vdash z : b}\,(\mathtt{ax})}{z : [b] \Vdash z : [b]}\,(\mathtt{many})}{y : [a], z : [b] \vdash (\lambda x.y)z : a}\,(\rightarrow_{\mathtt{e}})$$

Towards the Key Properties of the System

- $C[(\lambda x.t)u] \to_\beta C[t\{x/u\}]$ is non-erasing since $x \in \mathtt{fv}(t)$

  Example: $(\lambda x.xxx)z \to_\beta zzz$

- $C[(\lambda x.t)u] \to_\beta C[t]$ is erasing since $x \notin \mathtt{fv}(t)$.

  Example: $(\lambda x.y)z \to_\beta y$

Towards the Key Properties of the System

- $C[(\lambda x.t)u] \to_\beta C[t\{x/u\}]$ is non-erasing since $x \in \mathtt{fv}(t)$

  Example: $(\lambda x.xxx)z \to_\beta zzz$

- $C[(\lambda x.t)u] \to_\beta C[t]$ is erasing since $x \notin \mathtt{fv}(t)$.

  Example: $(\lambda x.y)z \to_\beta y$

Properties of the Type System

- **Weighted Subject Reduction**:
  - Non-Erasing reduction preserves types and environments.
  - Non-Erasing reduction strictly decreases the number of nodes of tree type derivations.
- **Subject Expansion**:
  - Non-Erasing anti-reduction preserves types and environments.

Corollary [**Characterization of Strong-Normalization**]:

Let t be a λ-term. Then t is typable if and only if t is strongly-normalizing.

Moreover, if Π is a type derivation of t, then the number of nodes of Π gives an upper bound to the length of the maximal reduction sequence starting at t.

Properties of the Type System

- **Weighted Subject Reduction**:
  - Non-Erasing reduction preserves types and environments.
  - Non-Erasing reduction strictly decreases the number of nodes of tree type derivations.
- **Subject Expansion**:
  - Non-Erasing anti-reduction preserves types and environments.

Corollary [**Characterization of Strong-Normalization**]:

Let t be a λ-term. Then t is typable if and only if t is strongly-normalizing.

Moreover, if Π is a type derivation of t, then the number of nodes of Π gives an upper bound to the length of the maximal reduction sequence starting at t.

Properties of the Type System

- **Weighted Subject Reduction**:
  - Non-Erasing reduction preserves types and environments.
  - Non-Erasing reduction strictly decreases the number of nodes of tree type derivations.
- **Subject Expansion**:
  - Non-Erasing anti-reduction preserves types and environments.

Corollary [**Characterization of Strong-Normalization**]:

Let t be a λ-term. Then t is typable if and only if t is strongly-normalizing.

Moreover, if Π is a type derivation of t, then the number of nodes of Π gives an upper bound to the length of the maximal reduction sequence starting at t.

Properties of the Type System

- **Weighted Subject Reduction**:
  - Non-Erasing reduction preserves types and environments.
  - Non-Erasing reduction strictly decreases the number of nodes of tree type derivations.
- **Subject Expansion**:
  - Non-Erasing anti-reduction preserves types and environments.

Corollary [**Characterization of Strong-Normalization**]:

Let t be a λ-term. Then t is typable if and only if t is strongly-normalizing.

Moreover, if Π is a type derivation of t, then the number of nodes of Π gives an upper bound to the length of the maximal reduction sequence starting at t.

Other Existing Characterizations

Characterization of Properties by means of Quantitative Types

- Solvability (**Bucciarelli-Kesner-RonchiDellaRocca**)
- Weak-normalization (**Bucciarelli-Kesner-Ventura**)
- Linear head normalization (**Kesner-Ventura**)
- Leftmost normalization (**Accattoli-Bernadet-Kesner-Lengrand**)
- Call-by-value normalization (**Guerrieri, Manzonetto-Pagani-RonchiDellaRocca**)
- Call-by-need normalization (**Kesner, Balabonski-Barenbaum-Bonelli-Kesner**)

Variants

- Calculi with pattern matching (**Bucciarelli-Kesner-RonchiDellaRocca**).
  - Based on a simple λ-calculus with pair patterns.
  - Another typical example of the Curry-Howard isomorphism.
  - The (small-step) operational semantics uses explicit pattern matching.
  - Can be used as a model for implementation functional languages.
- Sequent style calculi (**Kesner-Ventura**).
  - Based on the intuitionistic LJ calculus (**Herbelin**).
  - Another typical example of the Curry-Howard isomorphism.
  - The type system uses focusing.
- Classical calculi in Natural Deduction Style (**Kesner-Vial**).
  - Based on the $\lambda\mu$-calculus (**Parigot**).
  - Another typical example of the Curry-Howard isomorphism.
  - Types terms with classical types (e.g. with the Peirce's law $((a \Rightarrow b) \Rightarrow b) \Rightarrow b$).
  - Captures continuations (return to some place of the program).
  - Based on intersection and union types.

A Resource Aware Semantics for the λ-calculus

Small Step Operational Semantics

- Relation with Linear Logic Proof-Nets.
- Design of reduction strategies.
- Abstract machines.
- Calculi based on diferential logic (**Pagani-RonchiDellaRocca**).
- Calculi with propagation of substitutions (**Bernadet-Lengrand**).
- Calculi with substitution at a distance (**Accattoli-Kesner**).

Towards a Resource Aware Semantics

- **Bad Attempt**: based on traditional calculi with explicit substitutions
  - Propagation of (void) substitutions: $(tu)[x/v] \to t[x/v]u[x/v]$
- **Good Attempt**: based on the linear substitution calculus (LSC)
  - Goes back to **Milner**.
  - Implements linear substitution on demand, e.g.

    $(\dots x \dots x \dots)[x/u] \to (\dots u \dots x \dots)[x/u] \to (\dots u \dots u \dots)[x/u].$

The Linear Substitution Calculus

$$LSC \begin{cases} \mathsf{L}\langle \lambda x.t\rangle u & \to_{\mathtt{dB}} & \mathsf{L}\langle t[x/u]\rangle & \\ \mathsf{C}\langle x\rangle[x/u] & \to_{\mathtt{ls}} & \mathsf{C}\langle u\rangle[x/u] & \\ t[x/u] & \to_{\mathtt{gc}} & t & \text{if } x \notin \mathtt{fv}(t) \end{cases}$$

Example:

$$\begin{array}{ll} (\lambda x_1.yx_1x_1)(\lambda y.y) & \to_{\mathtt{B}} \\ (yx_1\ x_1\ )\ [x_1/\lambda y.y] & \to_{\mathtt{ls}} \\ (y\ x_1\ (\lambda y.y))\ [x_1/\lambda y.y] & \to_{\mathtt{ls}} \\ (y(\lambda y.y)(\lambda y.y))\ [x_1/\lambda y.y] & \to_{\mathtt{gc}} \\ (y(\lambda y.y)(\lambda y.y)) & \end{array}$$

Small vs Big

Big – Step

λ – calculus

$\downarrow$

Linear Substitution Calculus

Small – Step

- The **Big-Step** calculus can be **implemented** by the **Small-Step** calculus.
- The **Small-Step** calculus can be **projected** into the **Big-Step** calculus.
- The quantitative type systems of the λ-calculus can be endowed with **explicit substitutions**.
- The resulting quantitative type system allows to characterize different operational properties of the **Small-Step** calculus.
- The resulting quantitative type system provides **upper bounds** for some reduction strategies of the **Small-Step** calculus.

Small vs Big

Big – Step

λ – calculus

$\downarrow$

Linear Substitution Calculus

Small – Step

- The **Big-Step** calculus can be **implemented** by the **Small-Step** calculus.
- The **Small-Step** calculus can be **projected** into the **Big-Step** calculus.
- The quantitative type systems of the λ-calculus can be endowed with **explicit substitutions**.
- The resulting quantitative type system allows to characterize different operational properties of the **Small-Step** calculus.
- The resulting quantitative type system provides **upper bounds** for some reduction strategies of the **Small-Step** calculus.

Reasoning about Observational Equivalence

Lazy Evaluation and Call-by-Need

- Lazy evaluation was introduced by **Wadsworth** in 1971.
- Lazy evaluation models functional languages as **Haskell**.
- Modeled by call-by-need (**Ariola-Felleisen**, **Maraist-Odersky-Wadler**, ...)
- Different from call-by-name: first evaluates arguments (like call-by-value).

$$\Delta(\mathtt{II}) \to_{\mathtt{name}} (\mathtt{II})(\mathtt{II})\ldots$$

$$\Delta(\mathtt{II}) \to_{\mathtt{need}} \Delta\mathtt{I} \to_{\mathtt{need}} \mathtt{II}\ldots$$

- Different from call-by-value: values are only consumed when required

$$(\lambda x.\mathtt{I})(\mathtt{II}) \to_{\mathtt{value}} (\lambda x.\mathtt{I})\mathtt{I} \to_{\mathtt{value}} \mathtt{I}$$

$$(\lambda x.\mathtt{I})(\mathtt{II}) \not\to_{\mathtt{need}}$$

where $\Delta = \lambda x.xx$ and $\mathtt{I} = \lambda z.z$.

Properties

Call-by-need is **observationally equivalent** to call-by-name.

- Given a program t, the call-by-name interpreter on t stops in a value if and only if the call-by-need interpreter on t stops in an answer.
- Soundness ($\Leftarrow$ implication): trivial.
- Completeness ($\Rightarrow$ implication): Call-by-need turns out to be a correct implementation of call-by-name.
- Completeness Proof: complex proof making using of standardization, residual theory, sharing, etc.

In **Kesner 16** we use intersection types in a call-by-need framework to

- characterize some operational properties of call-by-need
- derive a completeness proof of call-by-need w.r.t. call-by-name in a more abstract (semantical) way.

Properties

Call-by-need is **observationally equivalent** to call-by-name.

- Given a program t, the call-by-name interpreter on t stops in a value if and only if the call-by-need interpreter on t stops in an answer.
- Soundness ($\Leftarrow$ implication): trivial.
- Completeness ($\Rightarrow$ implication): Call-by-need turns out to be a correct implementation of call-by-name.
- Completeness Proof: complex proof making using of standardization, residual theory, sharing, etc.

In **Kesner 16** we use intersection types in a call-by-need framework to

- characterize some operational properties of call-by-need
- derive a completeness proof of call-by-need w.r.t. call-by-name in a more abstract (semantical) way.

Inhabitation

The inhabitation problem for the lambda-calculus

Given a typing system, a variable assignment Γ and a type σ,

is there a lambda-term t such that $\Gamma \vdash t : \sigma$?

Theorem (Urzyczyn 1999)

The inhabitation problem for the idempotent type system is undecidable.

Towards Decidability

Theorem (Bucciarelli-Kesner-RonchiDellaRocca)

The inhabitation problem for the non-idempotent type system is decidable.

Proof.

- We give an inductive algorithm.
- The algorithm terminates.
- The algorithm is sound and complete.
- The algorithm is conservative w.r.t. the original inhabitation algorithm for simply typed lambda-calculus by **Ben-Yelles**.

□

Exact Bounds for Reduction Sequences

Quantitative Types for Exact Bounds

- The typing systems shown before give upper bounds for head-normalizing sequences and maximal sequences.
- Is there a typing system giving exact information about reduction lenghts?
- **de Carvalho**:

  $\Gamma \vdash^n t : \sigma$ if and only if n is the lenght of the head-reduction sequence from t to head-normal form.
- **Bernadet-Lengrand**:

  $\Gamma \vdash^n t : \sigma$ if and only if n is the maximal lenght of a reduction sequence from t to normal-form.
- Ongoing work, exact bounds for:
  - Left-reduction.
  - Linear-head reduction.
  - Call-by-need.
  - Control operators (classical logic).

Contributions of Quantitative Types

- Foundational aspect because of relation with Linear Logic.
- Model to design resource consumption semantics.
- Simplicity of reasoning by means of arithmetical decreasing measures.
- Power of non-idempotent intersection types to reason about:
  - Solvability
  - Normalization (head, head-linear, head-needed, weak, strong, value, infinitary etc)
  - Observational equivalence
  - Resource aware semantics
  - Synthesis of programs (inhabitation)

Future Work

- Extend to a powerful pattern matching model.
- Investigate the underlying models for $\lambda\mu$ and $\lambda\mu r$.
- Exact bounds for reduction sequences in languages with control operators.
- Non-idempotent types for classical term calculi based on sequent calculus.
- Inhabitation for classical systems?
- In contrast to the intuitionistic case, classical call-by-need and classical call-by-name are not observationally equivalent. It would be interesting to understand this result by means of non-idempotent types.