A Complete Polynomial λ-Calculus

E. De Benedetti and S. Ronchi Della Rocca

Università degli Studi di Torino

31 March 2012

Introduction

- **ICC**: Implicit Computational Complexity
- The problem: the design of programming languages with bounded computational complexity
- The proposed solution: a *ML-like* approach
  - λ-calculus as paradigmatic programming language
  - Types as semantic properties of terms
  - Type assignment for λ-calculus such that:
    - types guarantee the correctness of terms, in particular their complexity bound
    - if the type inference is decidable, the desired properties can be checked statically at compilation time
  - The tecnical tool: the Light Logics (derived from the Linear Logic of Girard) where the cut-elimination procedure is bounded in time by the size of the proof, exploiting the isomorphism:

FORMULAE as **TYPES**

Introduction

- **ICC**: Implicit Computational Complexity
- The problem: the design of programming languages with bounded computational complexity
- The proposed solution: a *ML-like* approach
  - λ-calculus as paradigmatic programming language
  - Types as semantic properties of terms
  - Type assignment for λ-calculus such that:
    - types guarantee the correctness of terms, in particular their complexity bound
    - if the type inference is decidable, the desired properties can be checked statically at compilation time
  - The tecnical tool: the Light Logics (derived from the Linear Logic of Girard) where the cut-elimination procedure is bounded in time by the size of the proof, exploiting the isomorphism:

FORMULAE as **TYPES**

Introduction

- **ICC**: Implicit Computational Complexity
- The problem: the design of programming languages with bounded computational complexity
- The proposed solution: a *ML-like* approach
  - λ-calculus as paradigmatic programming language
  - Types as semantic properties of terms
  - Type assignment for λ-calculus such that:
    - types guarantee the correctness of terms, in particular their complexity bound
    - if the type inference is decidable, the desired properties can be checked statically at compilation time
  - The tecnical tool: the Light Logics (derived from the Linear Logic of Girard) where the cut-elimination procedure is bounded in time by the size of the proof, exploiting the isomorphism:

FORMULAE as **TYPES**

Introduction

- **ICC**: Implicit Computational Complexity
- The problem: the design of programming languages with bounded computational complexity
- The proposed solution: a *ML-like* approach
  - λ-calculus as paradigmatic programming language
  - Types as semantic properties of terms
  - Type assignment for λ-calculus such that:
    - types guarantee the correctness of terms, in particular their complexity bound
    - if the type inference is decidable, the desired properties can be checked statically at compilation time
  - The tecnical tool: the Light Logics (derived from the Linear Logic of Girard) where the cut-elimination procedure is bounded in time by the size of the proof, exploiting the isomorphism:

FORMULAE as **TYPES**

A system with stratification types: ISTA

- ISTA: a type system with stratification types, "morally" equivalent to intersection types[1]
- Properties of ISTA:
  - sound and complete w.r.t. FPTIME
  - sound and complete w.r.t. strong normalization

[1] Coppo-Dezani 1978

A system with stratification types: ISTA

- ISTA: a type system with stratification types, “morally“ equivalent to intersection types[1]
- Properties of ISTA:
  - sound and complete w.r.t. FPTIME
  - sound and complete w.r.t. strong normalization

[1] Coppo-Dezani 1978

A system with stratification types: ISTA

- ISTA: a type system with stratification types, “morally“ equivalent to intersection types[1]
- Properties of ISTA:
  - sound and complete w.r.t. FPTIME
  - sound and complete w.r.t. strong normalization

[1] Coppo-Dezani 1978

Definitions

Type grammar

The set of types of ISTA is defined by the following syntax:

- $\mathtt{A} ::= \mathtt{a} \mid \sigma \multimap \mathtt{A} \mid \forall \mathtt{a}.\mathtt{A}$ (linear types)
- $\sigma ::= \mathtt{A} \mid \{\sigma_1, ..., \sigma_n\} \quad n > 0$ (stratified types)
- Comma , is associative, {} is not: so $\{\sigma_1, ..., \sigma_n\}$ is treated as a set
- $\{^j \mathtt{A}\}^j$ stands for $\underbrace{\{...\{}_{j} \mathtt{A} \underbrace{\}...\}}_{j}$
- ISTA proves judgments of the shape $\pi : \Gamma \mid \Delta \vdash \mathtt{M} : \mathtt{A}$
- $\perp$ denotes the empty basis
- $\{\Gamma\}$ denotes the basis such that $\Gamma(\mathtt{x}) = \sigma$ implies $\{\Gamma\}(\mathtt{x}) = \{\sigma\}$

Definitions

Type grammar

The set of types of ISTA is defined by the following syntax:

- $\mathtt{A} ::= \mathtt{a} \mid \sigma \multimap \mathtt{A} \mid \forall \mathtt{a}.\mathtt{A}$ (linear types)
- $\sigma ::= \mathtt{A} \mid \{\sigma_1, ..., \sigma_n\} \quad n > 0$ (stratified types)

- Comma , is associative, {} is not: so $\{\sigma_1, ..., \sigma_n\}$ is treated as a set
- $\{^j\mathtt{A}\}^j$ stands for $\underbrace{\{...\{}_{j}\ \mathtt{A}\ \underbrace{\}...\}}_{j}$
- ISTA proves judgments of the shape $\pi : \Gamma \mid \Delta \vdash \mathtt{M} : \mathtt{A}$
- $\perp$ denotes the empty basis
- $\{\Gamma\}$ denotes the basis such that $\Gamma(\mathtt{x}) = \sigma$ implies $\{\Gamma\}(\mathtt{x}) = \{\sigma\}$

Definitions

Type grammar

The set of types of ISTA is defined by the following syntax:

- $\mathtt{A} ::= \mathtt{a} \mid \sigma \multimap \mathtt{A} \mid \forall \mathtt{a}.\mathtt{A}$ (linear types)
- $\sigma ::= \mathtt{A} \mid \{\sigma_1, ..., \sigma_n\} \quad n > 0$ (stratified types)

- Comma , is associative, {} is not: so $\{\sigma_1, ..., \sigma_n\}$ is treated as a set
- $\{^j\mathtt{A}\}^j$ stands for $\underbrace{\{...\{}_{j}\ \mathtt{A}\ \underbrace{\}...\}}_{j}$
- ISTA proves judgments of the shape $\pi : \Gamma \mid \Delta \vdash \mathtt{M} : \mathtt{A}$
- $\perp$ denotes the empty basis
- $\{\Gamma\}$ denotes the basis such that $\Gamma(\mathtt{x}) = \sigma$ implies $\{\Gamma\}(\mathtt{x}) = \{\sigma\}$

Definitions

Type grammar

The set of types of ISTA is defined by the following syntax:

- $\mathtt{A} ::= \mathtt{a} \mid \sigma \multimap \mathtt{A} \mid \forall \mathtt{a}.\mathtt{A}$ (linear types)
- $\sigma ::= \mathtt{A} \mid \{\sigma_1, ..., \sigma_n\} \quad n > 0$ (stratified types)

- Comma , is associative, $\{\}$ is not: so $\{\sigma_1, ..., \sigma_n\}$ is treated as a set
- $\{^j\mathtt{A}\}^j$ stands for $\underbrace{\{...\{}_{j} \ \mathtt{A} \ \underbrace{\}...\}}_{j}$
- ISTA proves judgments of the shape $\pi : \Gamma \mid \Delta \vdash \mathtt{M} : \mathtt{A}$
- $\perp$ denotes the empty basis
- $\{\Gamma\}$ denotes the basis such that $\Gamma(\mathtt{x}) = \sigma$ implies $\{\Gamma\}(\mathtt{x}) = \{\sigma\}$

Definitions

Type grammar

The set of types of ISTA is defined by the following syntax:

- $\mathtt{A} ::= \mathtt{a} \mid \sigma \multimap \mathtt{A} \mid \forall \mathtt{a}.\mathtt{A}$ (linear types)
- $\sigma ::= \mathtt{A} \mid \{\sigma_1, ..., \sigma_n\} \quad n > 0$ (stratified types)

- Comma , is associative, $\{\}$ is not: so $\{\sigma_1, ..., \sigma_n\}$ is treated as a set
- $\{^j\mathtt{A}\}^j$ stands for $\underbrace{\{...\{}_{j} \ \mathtt{A} \ \underbrace{\}...\}}_{j}$
- ISTA proves judgments of the shape $\pi : \Gamma \mid \Delta \vdash \mathtt{M} : \mathtt{A}$
- $\perp$ denotes the empty basis
- $\{\Gamma\}$ denotes the basis such that $\Gamma(\mathtt{x}) = \sigma$ implies $\{\Gamma\}(\mathtt{x}) = \{\sigma\}$

Definitions

Type grammar

The set of types of ISTA is defined by the following syntax:

- $\mathtt{A} ::= \mathtt{a} \mid \sigma \multimap \mathtt{A} \mid \forall \mathtt{a}.\mathtt{A}$ (linear types)
- $\sigma ::= \mathtt{A} \mid \{\sigma_1, ..., \sigma_n\} \quad n > 0$ (stratified types)

- Comma , is associative, $\{\}$ is not: so $\{\sigma_1, ..., \sigma_n\}$ is treated as a set
- $\{^j\mathtt{A}\}^j$ stands for $\underbrace{\{...\{}_{j} \ \mathtt{A} \ \underbrace{\}...\}}_{j}$
- ISTA proves judgments of the shape $\pi : \Gamma \mid \Delta \vdash \mathtt{M} : \mathtt{A}$
- $\perp$ denotes the empty basis
- $\{\Gamma\}$ denotes the basis such that $\Gamma(\mathtt{x}) = \sigma$ implies $\{\Gamma\}(\mathtt{x}) = \{\sigma\}$

Definitions

Type grammar

The set of types of ISTA is defined by the following syntax:

- $\mathtt{A} ::= \mathtt{a} \mid \sigma \multimap \mathtt{A} \mid \forall \mathtt{a}.\mathtt{A}$ (linear types)
- $\sigma ::= \mathtt{A} \mid \{\sigma_1, ..., \sigma_n\} \quad n > 0$ (stratified types)

- Comma , is associative, $\{\}$ is not: so $\{\sigma_1, ..., \sigma_n\}$ is treated as a set
- $\{^j\mathtt{A}\}^j$ stands for $\underbrace{\{...\{}_{j} \ \mathtt{A} \ \underbrace{\}...\}}_{j}$
- ISTA proves judgments of the shape $\pi : \Gamma \mid \Delta \vdash \mathtt{M} : \mathtt{A}$
- $\perp$ denotes the empty basis
- $\{\Gamma\}$ denotes the basis such that $\Gamma(\mathtt{x}) = \sigma$ implies $\{\Gamma\}(\mathtt{x}) = \{\sigma\}$

Typing rules for ISTA (1)

$$\frac{}{\mathtt{x} : \mathtt{A} \mid \bot \vdash \mathtt{x} : \mathtt{A}}\ (Ax) \qquad \frac{\Gamma \mid \Delta \vdash \mathtt{M} : \mathtt{B} \quad \mathtt{x} \notin \mathsf{dom}(\Gamma, \Delta)}{\Gamma \mid \Delta \vdash \lambda \mathtt{x}.\mathtt{M} : \mathtt{A} \multimap \mathtt{B}}\ (\multimap I_w)$$

$$\frac{\Gamma, \mathtt{x} : \mathtt{A} \mid \Delta \vdash \mathtt{M} : \mathtt{B}}{\Gamma \mid \Delta \vdash \lambda \mathtt{x}.\mathtt{M} : \mathtt{A} \multimap \mathtt{B}}\ (\multimap I_l) \qquad \frac{\Gamma \mid \Delta, \mathtt{x} : \sigma \vdash \mathtt{M} : \mathtt{B}}{\Gamma \mid \Delta \vdash \lambda \mathtt{x}.\mathtt{M} : \sigma \multimap \mathtt{B}}\ (\multimap I_s)$$

$$\frac{\Gamma_1 \mid \Delta_1 \vdash \mathtt{M} : \sigma \multimap \mathtt{A} \quad \Gamma_2 \mid \Delta_2 \vdash \mathtt{N} : \sigma \quad \Gamma_1, \Delta_1 \# \Gamma_2, \Delta_2}{\Gamma_1, \Gamma_2 \mid \Delta_1, \Delta_2 \vdash \mathtt{MN} : \mathtt{A}}\ (\multimap E)$$

$$\frac{\Gamma \mid \Delta \vdash \mathtt{M} : \mathtt{A} \quad \mathtt{a} \notin \mathsf{FV}(\Gamma, \Delta)}{\Gamma \mid \Delta \vdash \mathtt{M} : \forall \mathtt{a}.A}\ (\forall I) \qquad \frac{\Gamma \mid \Delta \vdash \mathtt{M} : \forall \mathtt{a}.\mathtt{B}}{\Gamma \mid \Delta \vdash \mathtt{M} : \mathtt{B}[\mathtt{A}/\mathtt{a}]}\ (\forall E)$$

Typing rules for ISTA (1)

$$\frac{}{\mathtt{x} : \mathtt{A} \mid \bot \vdash \mathtt{x} : \mathtt{A}}\ (Ax) \qquad \frac{\Gamma \mid \Delta \vdash \mathtt{M} : \mathtt{B} \quad \mathtt{x} \notin \mathsf{dom}(\Gamma, \Delta)}{\Gamma \mid \Delta \vdash \lambda \mathtt{x}.\mathtt{M} : \mathtt{A} \multimap \mathtt{B}}\ (\multimap I_w)$$

$$\frac{\Gamma, \mathtt{x} : \mathtt{A} \mid \Delta \vdash \mathtt{M} : \mathtt{B}}{\Gamma \mid \Delta \vdash \lambda \mathtt{x}.\mathtt{M} : \mathtt{A} \multimap \mathtt{B}}\ (\multimap I_l) \qquad \frac{\Gamma \mid \Delta, \mathtt{x} : \sigma \vdash \mathtt{M} : \mathtt{B}}{\Gamma \mid \Delta \vdash \lambda \mathtt{x}.\mathtt{M} : \sigma \multimap \mathtt{B}}\ (\multimap I_s)$$

$$\frac{\Gamma_1 \mid \Delta_1 \vdash \mathtt{M} : \sigma \multimap \mathtt{A} \quad \Gamma_2 \mid \Delta_2 \vdash \mathtt{N} : \sigma \quad \Gamma_1, \Delta_1 \# \Gamma_2, \Delta_2}{\Gamma_1, \Gamma_2 \mid \Delta_1, \Delta_2 \vdash \mathtt{MN} : \mathtt{A}}\ (\multimap E)$$

$$\frac{\Gamma \mid \Delta \vdash \mathtt{M} : \mathtt{A} \quad \mathtt{a} \notin \mathsf{FV}(\Gamma, \Delta)}{\Gamma \mid \Delta \vdash \mathtt{M} : \forall \mathtt{a}.A}\ (\forall I) \qquad \frac{\Gamma \mid \Delta \vdash \mathtt{M} : \forall \mathtt{a}.\mathtt{B}}{\Gamma \mid \Delta \vdash \mathtt{M} : \mathtt{B}[\mathtt{A}/\mathtt{a}]}\ (\forall E)$$

Typing rules for ISTA (2)

$$\frac{\Gamma, \mathtt{x}_1 : \mathtt{A}, ..., \mathtt{x}_n : \mathtt{A} \mid \Delta \vdash \mathtt{M} : \tau}{\Gamma \mid \Delta, \mathtt{x} : \{\mathtt{A}\} \vdash \mathtt{M}[\mathtt{x}/\mathtt{x}_1, ..., \mathtt{x}_n] : \tau} \ (m_l)$$

$$\frac{\Gamma \mid \Delta, \mathtt{x}_1 : \sigma_1, ..., \mathtt{x}_n : \sigma_n \vdash \mathtt{M} : \tau}{\Gamma \mid \Delta, \mathtt{x} : \{\sigma_1 ..., \sigma_n\} \vdash \mathtt{M}[\mathtt{x}/\mathtt{x}_1, ..., \mathtt{x}_n] : \tau} \ (m_s)$$

$$\frac{\Gamma_i \mid \Delta_i \vdash \mathtt{M} : \sigma_i \quad n \geq 1}{\bot \mid \{\Gamma_i\} \cup_i \{\Delta_i\} \vdash \mathtt{M} : \{\sigma_1, ..., \sigma_n\}} \ (sp)$$

where $\{\Gamma_i\} \cup_i \{\Delta_i\} = \{\Gamma_1\} \cup ... \cup \{\Gamma_n\} \cup \{\Delta_1\} \cup ... \cup \{\Delta_n\}$

Typing rules for ISTA (2)

$$\frac{\Gamma, \mathtt{x}_1 : \mathtt{A}, ..., \mathtt{x}_n : \mathtt{A} \mid \Delta \vdash \mathtt{M} : \tau}{\Gamma \mid \Delta, \mathtt{x} : \{\mathtt{A}\} \vdash \mathtt{M}[\mathtt{x}/\mathtt{x}_1, ..., \mathtt{x}_n] : \tau} \ (m_l)$$

$$\frac{\Gamma \mid \Delta, \mathtt{x}_1 : \sigma_1, ..., \mathtt{x}_n : \sigma_n \vdash \mathtt{M} : \tau}{\Gamma \mid \Delta, \mathtt{x} : \{\sigma_1 ..., \sigma_n\} \vdash \mathtt{M}[\mathtt{x}/\mathtt{x}_1, ..., \mathtt{x}_n] : \tau} \ (m_s)$$

$$\frac{\Gamma_i \mid \Delta_i \vdash \mathtt{M} : \sigma_i \quad n \geq 1}{\bot \mid \{\Gamma_i\} \cup_i \{\Delta_i\} \vdash \mathtt{M} : \{\sigma_1, ..., \sigma_n\}} \ (sp)$$

where $\{\Gamma_i\} \cup_i \{\Delta_i\} = \{\Gamma_1\} \cup ... \cup \{\Gamma_n\} \cup \{\Delta_1\} \cup ... \cup \{\Delta_n\}$

Typing rules for ISTA (2)

$$\frac{\Gamma, \mathtt{x}_1 : \mathtt{A}, ..., \mathtt{x}_n : \mathtt{A} \mid \Delta \vdash \mathtt{M} : \tau}{\Gamma \mid \Delta, \mathtt{x} : \{\mathtt{A}\} \vdash \mathtt{M}[\mathtt{x}/\mathtt{x}_1, ..., \mathtt{x}_n] : \tau} \ (m_l)$$

$$\frac{\Gamma \mid \Delta, \mathtt{x}_1 : \sigma_1, ..., \mathtt{x}_n : \sigma_n \vdash \mathtt{M} : \tau}{\Gamma \mid \Delta, \mathtt{x} : \{\sigma_1 ..., \sigma_n\} \vdash \mathtt{M}[\mathtt{x}/\mathtt{x}_1, ..., \mathtt{x}_n] : \tau} \ (m_s)$$

$$\frac{\Gamma_i \mid \Delta_i \vdash \mathtt{M} : \sigma_i \quad n \geq 1}{\bot \mid \{\Gamma_i\} \cup_i \{\Delta_i\} \vdash \mathtt{M} : \{\sigma_1, ..., \sigma_n\}} \ (sp)$$

where $\{\Gamma_i\} \cup_i \{\Delta_i\} = \{\Gamma_1\} \cup ... \cup \{\Gamma_n\} \cup \{\Delta_1\} \cup ... \cup \{\Delta_n\}$

Properties of ISTA

Property: Let $\pi : \Gamma \mid \Delta \vdash \mathtt{M} : \{\sigma_1, \ldots, \sigma_n\}$ where $n > 0$. Then $\Gamma = \perp$, and π ends with an application of rule (sp), followed by a (possibly empty) sequence of applications of rule (m_s).

Lemma: Substitution

i) $\pi : \Gamma_1, \mathtt{x} : \mathtt{A} \mid \Delta_1 \vdash \mathtt{M} : \tau, \rho : \Gamma_2 \mid \Delta_2 \vdash \mathtt{N} : \mathtt{A}$ $(\Gamma_1 \# \Gamma_2)$ imply

$S(\rho, \pi) : \Gamma_1, \Gamma_2 \mid \Delta_1 \cup \Delta_2 \vdash \mathtt{M}[\mathtt{N}/\mathtt{x}] : \tau$

ii) $\pi : \Gamma_1 \mid \Delta_1, \mathtt{x} : \sigma \vdash \mathtt{M} : \tau, \rho : \Gamma_2 \mid \Delta_2 \vdash \mathtt{N} : \sigma$ $(\Gamma_1 \# \Gamma_2)$ imply

$S(\rho, \pi) : \Gamma_1, \Gamma_2 \mid \Delta_1 \cup \Delta_2 \vdash \mathtt{M}[\mathtt{N}/\mathtt{x}] : \tau$

Theorem: Subject Reduction

$\Gamma \mid \Delta \vdash \mathtt{M} : \sigma$ and $\mathtt{M} \underset{\beta}{\longrightarrow} \mathtt{N}$ implies $\Gamma \mid \Delta \vdash \mathtt{N} : \sigma$.

Properties of ISTA

Property: Let $\pi : \Gamma \mid \Delta \vdash \mathtt{M} : \{\sigma_1, \ldots, \sigma_n\}$ where $n > 0$. Then $\Gamma = \bot$, and π ends with an application of rule (sp), followed by a (possibly empty) sequence of applications of rule (m_s).

Lemma: Substitution

i) $\pi : \Gamma_1, \mathtt{x} : \mathtt{A} \mid \Delta_1 \vdash \mathtt{M} : \tau, \rho : \Gamma_2 \mid \Delta_2 \vdash \mathtt{N} : \mathtt{A}$ $(\Gamma_1 \# \Gamma_2)$ imply

$S(\rho, \pi) : \Gamma_1, \Gamma_2 \mid \Delta_1 \cup \Delta_2 \vdash \mathtt{M}[\mathtt{N}/\mathtt{x}] : \tau$

ii) $\pi : \Gamma_1 \mid \Delta_1, \mathtt{x} : \sigma \vdash \mathtt{M} : \tau, \rho : \Gamma_2 \mid \Delta_2 \vdash \mathtt{N} : \sigma$ $(\Gamma_1 \# \Gamma_2)$ imply

$S(\rho, \pi) : \Gamma_1, \Gamma_2 \mid \Delta_1 \cup \Delta_2 \vdash \mathtt{M}[\mathtt{N}/\mathtt{x}] : \tau$

Theorem: Subject Reduction

$\Gamma \mid \Delta \vdash \mathtt{M} : \sigma$ and $\mathtt{M} \underset{\beta}{\longrightarrow} \mathtt{N}$ implies $\Gamma \mid \Delta \vdash \mathtt{N} : \sigma$.

Properties of ISTA

Property: Let $\pi : \Gamma \mid \Delta \vdash \mathtt{M} : \{\sigma_1, \ldots, \sigma_n\}$ where $n > 0$. Then $\Gamma = \perp$, and π ends with an application of rule (sp), followed by a (possibly empty) sequence of applications of rule (m_s).

Lemma: Substitution

i) $\pi : \Gamma_1, \mathtt{x} : \mathtt{A} \mid \Delta_1 \vdash \mathtt{M} : \tau, \rho : \Gamma_2 \mid \Delta_2 \vdash \mathtt{N} : \mathtt{A}$ $(\Gamma_1 \# \Gamma_2)$ imply

$S(\rho, \pi) : \Gamma_1, \Gamma_2 \mid \Delta_1 \cup \Delta_2 \vdash \mathtt{M}[\mathtt{N}/\mathtt{x}] : \tau$

ii) $\pi : \Gamma_1 \mid \Delta_1, \mathtt{x} : \sigma \vdash \mathtt{M} : \tau, \rho : \Gamma_2 \mid \Delta_2 \vdash \mathtt{N} : \sigma$ $(\Gamma_1 \# \Gamma_2)$ imply

$S(\rho, \pi) : \Gamma_1, \Gamma_2 \mid \Delta_1 \cup \Delta_2 \vdash \mathtt{M}[\mathtt{N}/\mathtt{x}] : \tau$

Theorem: Subject Reduction

$\Gamma \mid \Delta \vdash \mathtt{M} : \sigma$ and $\mathtt{M} \underset{\beta}{\longrightarrow} \mathtt{N}$ implies $\Gamma \mid \Delta \vdash \mathtt{N} : \sigma$.

Strong normalization

ISTA is sound and complete w.r.t. strong normalization:

- if a term is typable in ISTA then it is strongly normalizing (soundness)

Proof. Termination of reduction does not depend on the strategy

- all strongly normalizing terms can be typed in ISTA (completeness)

Proof. Following the guidelines of Neergaard[2]:

1. Longest reduction strategy
2. Subject expansion under perpetual strategy
3. Completeness follows as a corollary

[2] Neergaard 2005

Strong normalization

`ISTA` is sound and complete w.r.t. strong normalization:

- if a term is typable in `ISTA` then it is strongly normalizing (soundness)

  Proof. Termination of reduction does not depend on the strategy

- all strongly normalizing terms can be typed in `ISTA` (completeness)

  Proof. Following the guidelines of Neergaard[2]:

  1. Longest reduction strategy
  2. Subject expansion under perpetual strategy
  3. Completeness follows as a corollary

[2] Neergaard 2005

Strong normalization

`ISTA` is sound and complete w.r.t. strong normalization:

- if a term is typable in `ISTA` then it is strongly normalizing (soundness)

  Proof. Termination of reduction does not depend on the strategy

- all strongly normalizing terms can be typed in `ISTA` (completeness)

  Proof. Following the guidelines of Neergaard[2]:

  1. Longest reduction strategy
  2. Subject expansion under perpetual strategy
  3. Completeness follows as a corollary

[2] Neergaard 2005

Typability w.r.t. System F

- More terms can be typed

  e.g. $\vdash (\lambda \mathtt{xy.y}(\mathtt{x}I)(\mathtt{x}K))\Delta : ((\mathtt{A} \multimap \mathtt{A}) \multimap \mathtt{B} \multimap \mathtt{C}) \multimap \mathtt{C}$

- More types can be assigned

  e.g. $\vdash \lambda \mathtt{x.xx} : \{\{\sigma\}, \{\sigma \multimap \mathtt{B}\}\} \multimap \mathtt{B}$

- Stratification does not limit the number of terms that can be typed, but it is necessary to forbid the iteration of "dangerous" functions, e.g. multiplication

Typability w.r.t. System F

- More terms can be typed

  e.g. $\vdash (\lambda \mathtt{xy.y}(\mathtt{x}I)(\mathtt{x}K))\Delta : ((\mathtt{A} \multimap \mathtt{A}) \multimap \mathtt{B} \multimap \mathtt{C}) \multimap \mathtt{C}$

- More types can be assigned

  e.g. $\vdash \lambda \mathtt{x.xx} : \{\{\sigma\}, \{\sigma \multimap \mathtt{B}\}\} \multimap \mathtt{B}$

- Stratification does not limit the number of terms that can be typed, but it is necessary to forbid the iteration of "dangerous" functions, e.g. multiplication

Typability w.r.t. System F

- More terms can be typed

  e.g. $\vdash (\lambda \mathtt{xy.y}(\mathtt{x}I)(\mathtt{x}K))\Delta : ((\mathtt{A} \multimap \mathtt{A}) \multimap \mathtt{B} \multimap \mathtt{C}) \multimap \mathtt{C}$

- More types can be assigned

  e.g. $\vdash \lambda \mathtt{x.xx} : \{\{\sigma\}, \{\sigma \multimap \mathtt{B}\}\} \multimap \mathtt{B}$

- Stratification does not limit the number of terms that can be typed, but it is necessary to forbid the iteration of "dangerous" functions, e.g. multiplication

Normalization bound

- A term in ISTA can be reduced to its normal form in a number of steps that depends on the size of the term
- **Measure of reduction**: if $\pi : \Gamma \mid \Delta \vdash \mathtt{M} : \sigma$, and $\mathtt{M} \xrightarrow[\beta]{*} \mathtt{M}'$ in m steps, then $m \leq |\mathtt{M}|^{\mathrm{d}}$
- d may depend on the degree of the derivation

Normalization bound

- A term in ISTA can be reduced to its normal form in a number of steps that depends on the size of the term
- **Measure of reduction**: if $\pi : \Gamma \mid \Delta \vdash \mathtt{M} : \sigma$, and $\mathtt{M} \xrightarrow[\beta]{*} \mathtt{M}'$ in m steps, then $m \leq |\mathtt{M}|^{\mathrm{d}}$
- d may depend on the degree of the derivation

Normalization bound

- A term in ISTA can be reduced to its normal form in a number of steps that depends on the size of the term
- **Measure of reduction**: if $\pi : \Gamma \mid \Delta \vdash \mathtt{M} : \sigma$, and $\mathtt{M} \xrightarrow[\beta]{*} \mathtt{M}'$ in m steps, then $m \leq |\mathtt{M}|^{\mathtt{d}}$
- $\mathtt{d}$ may depend on the degree of the derivation

Soft Type Assignment system (STA)

- To prove soundness and completeness w.r.t FPTIME, we reduce ISTA to STA[3]
- STA: a type assignment for λ-calculus derived from SLL[4]
- Properties of STA:
  - Subject reduction
  - Soundness: a term typable in STA reduces to normal form in a number of steps polynomial in its size
  - Completeness: every polynomial time function can be encoded by a term typable in STA
  - A proper subset of the strongly normalizing terms is typed
- Is has been proved[5] that adding intersection types to simple types does not increase the definability of functions

[3] Gaboardi-Ronchi 2007
[4] Lafont 1988
[5] Bucciarelli-Piperno 2003

Soft Type Assignment system (STA)

- To prove soundness and completeness w.r.t FPTIME, we reduce ISTA to STA[3]
- STA: a type assignment for λ-calculus derived from SLL[4]
- Properties of STA:
  - Subject reduction
  - Soundness: a term typable in STA reduces to normal form in a number of steps polynomial in its size
  - Completeness: every polynomial time function can be encoded by a term typable in STA
  - A proper subset of the strongly normalizing terms is typed
- Is has been proved[5] that adding intersection types to simple types does not increase the definability of functions

[3] Gaboardi-Ronchi 2007
[4] Lafont 1988
[5] Bucciarelli-Piperno 2003

Soft Type Assignment system (STA)

- To prove soundness and completeness w.r.t FPTIME, we reduce ISTA to STA[3]
- STA: a type assignment for λ-calculus derived from SLL[4]
- Properties of STA:
  - Subject reduction
  - Soundness: a term typable in STA reduces to normal form in a number of steps polynomial in its size
  - Completeness: every polynomial time function can be encoded by a term typable in STA
  - A proper subset of the strongly normalizing terms is typed
- Is has been proved[5] that adding intersection types to simple types does not increase the definability of functions

[3] Gaboardi-Ronchi 2007
[4] Lafont 1988
[5] Bucciarelli-Piperno 2003

Soft Type Assignment system (STA)

- To prove soundness and completeness w.r.t FPTIME, we reduce ISTA to STA[3]
- STA: a type assignment for λ-calculus derived from SLL[4]
- Properties of STA:
  - Subject reduction
  - Soundness: a term typable in STA reduces to normal form in a number of steps polynomial in its size
  - Completeness: every polynomial time function can be encoded by a term typable in STA
  - A proper subset of the strongly normalizing terms is typed
- Is has been proved[5] that adding intersection types to simple types does not increase the definability of functions

[3] Gaboardi-Ronchi 2007
[4] Lafont 1988
[5] Bucciarelli-Piperno 2003

Soft Type Assignment system (STA)

- To prove soundness and completeness w.r.t FPTIME, we reduce ISTA to STA[3]
- STA: a type assignment for λ-calculus derived from SLL[4]
- Properties of STA:
  - Subject reduction
  - Soundness: a term typable in STA reduces to normal form in a number of steps polynomial in its size
  - Completeness: every polynomial time function can be encoded by a term typable in STA
  - A proper subset of the strongly normalizing terms is typed
- Is has been proved[5] that adding intersection types to simple types does not increase the definability of functions

[3] Gaboardi-Ronchi 2007
[4] Lafont 1988
[5] Bucciarelli-Piperno 2003

Soft Type Assignment system (STA)

- To prove soundness and completeness w.r.t FPTIME, we reduce ISTA to STA[3]
- STA: a type assignment for λ-calculus derived from SLL[4]
- Properties of STA:
  - Subject reduction
  - Soundness: a term typable in STA reduces to normal form in a number of steps polynomial in its size
  - Completeness: every polynomial time function can be encoded by a term typable in STA
  - A proper subset of the strongly normalizing terms is typed
- Is has been proved[5] that adding intersection types to simple types does not increase the definability of functions

[3] Gaboardi-Ronchi 2007
[4] Lafont 1988
[5] Bucciarelli-Piperno 2003

Soft Type Assignment system (STA)

- To prove soundness and completeness w.r.t FPTIME, we reduce ISTA to STA[3]
- STA: a type assignment for λ-calculus derived from SLL[4]
- Properties of STA:
  - Subject reduction
  - Soundness: a term typable in STA reduces to normal form in a number of steps polynomial in its size
  - Completeness: every polynomial time function can be encoded by a term typable in STA
  - A proper subset of the strongly normalizing terms is typed
- Is has been proved[5] that adding intersection types to simple types does not increase the definability of functions

[3] Gaboardi-Ronchi 2007
[4] Lafont 1988
[5] Bucciarelli-Piperno 2003

Reminder: STA

The set of types of STA is defined by the following syntax:

- $\mathtt{U} ::= \mathtt{a} \mid \mu \multimap \mathtt{U} \mid \forall \mathtt{a}.\mathtt{U}$ (linear types)
- $\mu ::= \mathtt{U} \mid !\mu$ (modal types)

STA

$$\frac{}{\mathtt{x} : \mathtt{U} \vdash_{\mathrm{STA}} \mathtt{x} : \mathtt{U}}\ (Ax) \qquad \frac{\Theta \vdash_{\mathrm{STA}} \mathtt{M} : \mu \quad \mathtt{x} \notin dom\Theta}{\Theta, \mathtt{x} : \mathtt{U} \vdash_{\mathrm{STA}} \mathtt{M} : \mu}\ (w)$$

$$\frac{\Theta, \mathtt{x} : \mu \vdash_{\mathrm{STA}} \mathtt{M} : \mathtt{U}}{\Theta \vdash_{\mathrm{STA}} \lambda \mathtt{x}.\mathtt{M} : \mu \multimap \mathtt{U}}\ (\multimap I) \qquad \frac{\Theta \vdash_{\mathrm{STA}} \mathtt{M} : \mu \multimap \mathtt{U} \quad \Xi \vdash_{\mathrm{STA}} \mathtt{N} : \mu \quad \Theta \# \Xi}{\Theta, \Xi \vdash_{\mathrm{STA}} \mathtt{MN} : \mathtt{U}}\ (\multimap E)$$

$$\frac{\Theta \vdash_{\mathrm{STA}} \mathtt{M} : \mathtt{U} \quad \mathtt{a} \notin FV(\Theta)}{\Theta \vdash_{\mathrm{STA}} \mathtt{M} : \forall \mathtt{a}.\mathtt{U}}\ (\forall I) \qquad \frac{\Theta \vdash_{\mathrm{STA}} \mathtt{M} : \forall \mathtt{a}.\mathtt{B}}{\Theta \vdash_{\mathrm{STA}} \mathtt{M} : \mathtt{B}[\mathtt{U}/\mathtt{a}]}\ (\forall E)$$

$$\frac{\Theta, \mathtt{x}_1 : \mu, \ldots, \mathtt{x}_n : \mu \vdash_{\mathrm{STA}} \mathtt{M} : \nu}{\Theta, \mathtt{x} : !\mu \vdash_{\mathrm{STA}} \mathtt{M}[\mathtt{x}/\mathtt{x}_1, \ldots, \mathtt{x}_n] : \nu}\ (m) \qquad \frac{\Theta \vdash \mathtt{M} : \mu}{!\Theta \vdash_{\mathrm{STA}} \mathtt{M} : !\mu}\ (sp)$$

Reminder: STA

The set of types of STA is defined by the following syntax:

- $\mathtt{U} ::= \mathtt{a} \mid \mu \multimap \mathtt{U} \mid \forall \mathtt{a}.\mathtt{U}$ (linear types)
- $\mu ::= \mathtt{U} \mid !\mu$ (modal types)

STA

$$\frac{}{\mathtt{x} : \mathtt{U} \vdash_{\mathtt{STA}} \mathtt{x} : \mathtt{U}}\ (Ax) \qquad \frac{\Theta \vdash_{\mathtt{STA}} \mathtt{M} : \mu \quad \mathtt{x} \notin dom\Theta}{\Theta, \mathtt{x} : \mathtt{U} \vdash_{\mathtt{STA}} \mathtt{M} : \mu}\ (w)$$

$$\frac{\Theta, \mathtt{x} : \mu \vdash_{\mathtt{STA}} \mathtt{M} : \mathtt{U}}{\Theta \vdash_{\mathtt{STA}} \lambda \mathtt{x}.\mathtt{M} : \mu \multimap \mathtt{U}}\ (\multimap I) \qquad \frac{\Theta \vdash_{\mathtt{STA}} \mathtt{M} : \mu \multimap \mathtt{U} \quad \Xi \vdash_{\mathtt{STA}} \mathtt{N} : \mu \quad \Theta \# \Xi}{\Theta, \Xi \vdash_{\mathtt{STA}} \mathtt{MN} : \mathtt{U}}\ (\multimap E)$$

$$\frac{\Theta \vdash_{\mathtt{STA}} \mathtt{M} : \mathtt{U} \quad \mathtt{a} \notin FV(\Theta)}{\Theta \vdash_{\mathtt{STA}} \mathtt{M} : \forall \mathtt{a}.\mathtt{U}}\ (\forall I) \qquad \frac{\Theta \vdash_{\mathtt{STA}} \mathtt{M} : \forall \mathtt{a}.\mathtt{B}}{\Theta \vdash_{\mathtt{STA}} \mathtt{M} : \mathtt{B}[\mathtt{U}/\mathtt{a}]}\ (\forall E)$$

$$\frac{\Theta, \mathtt{x}_1 : \mu, \ldots, \mathtt{x}_n : \mu \vdash_{\mathtt{STA}} \mathtt{M} : \nu}{\Theta, \mathtt{x} : !\mu \vdash_{\mathtt{STA}} \mathtt{M}[\mathtt{x}/\mathtt{x}_1, \ldots, \mathtt{x}_n] : \nu}\ (m) \qquad \frac{\Theta \vdash \mathtt{M} : \mu}{!\Theta \vdash_{\mathtt{STA}} \mathtt{M} : !\mu}\ (sp)$$

Representation of numerals

- We represent numerals in Church style, so m is represented by $\underline{m} = \lambda \mathbf{xy}.\mathbf{x}^m\mathbf{y}$
- In STA, $\underline{m}$ can be typed either
  - uniformely by $\mathbf{N} = \forall a.!(a \multimap a) \multimap a \multimap a$, or
  - by $\mathbf{N}_n = \forall a.!^n(a \multimap a) \multimap a \multimap a$, for all $n \geq 1$
- In ISTA, $\underline{m}$ can be typed either
  - uniformely by $\mathbf{NI} = \forall a.\{a \multimap a\} \multimap a \multimap a$, or
  - by $\mathbf{NI}_n = \forall a.\{^n a \multimap a\}^n \multimap a \multimap a$, for all $n \geq 1$

Note the similarity between $\mathbf{N}_n$ and $\mathbf{NI}_n$

Representation of numerals

- We represent numerals in Church style, so m is represented by

  $\underline{m} = \lambda \mathbf{xy}.\mathbf{x}^m\mathbf{y}$

- In STA, $\underline{m}$ can be typed either

  uniformely by $\mathbf{N} = \forall \mathrm{a}.!(\mathrm{a} \multimap \mathrm{a}) \multimap \mathrm{a} \multimap \mathrm{a}$, or

  by $\mathbf{N}_n = \forall \mathrm{a}.!^n(\mathrm{a} \multimap \mathrm{a}) \multimap \mathrm{a} \multimap \mathrm{a}$, for all $n \geq 1$

- In ISTA, $\underline{m}$ can be typed either

  uniformely by $\mathbf{NI} = \forall \mathrm{a}.\{\mathrm{a} \multimap \mathrm{a}\} \multimap \mathrm{a} \multimap \mathrm{a}$, or

  by $\mathbf{NI}_n = \forall \mathrm{a}.\{^n\mathrm{a} \multimap \mathrm{a}\}^n \multimap \mathrm{a} \multimap \mathrm{a}$, for all $n \geq 1$

Note the similarity between $\mathbf{N}_n$ and $\mathbf{NI}_n$

Representation of numerals

- We represent numerals in Church style, so m is represented by

  $\underline{m} = \lambda \mathbf{xy}.\mathbf{x}^m\mathbf{y}$

- In STA, $\underline{m}$ can be typed either

  uniformely by $\mathbf{N} = \forall \mathrm{a}.!(\mathrm{a} \multimap \mathrm{a}) \multimap \mathrm{a} \multimap \mathrm{a}$, or

  by $\mathbf{N}_n = \forall \mathrm{a}.!^n(\mathrm{a} \multimap \mathrm{a}) \multimap \mathrm{a} \multimap \mathrm{a}$, for all $n \geq 1$

- In ISTA, $\underline{m}$ can be typed either

  uniformely by $\mathbf{NI} = \forall \mathrm{a}.\{\mathrm{a} \multimap \mathrm{a}\} \multimap \mathrm{a} \multimap \mathrm{a}$, or

  by $\mathbf{NI}_n = \forall \mathrm{a}.\{^n\mathrm{a} \multimap \mathrm{a}\}^n \multimap \mathrm{a} \multimap \mathrm{a}$, for all $n \geq 1$

Note the similarity between $\mathbf{N}_n$ and $\mathbf{NI}_n$

Representation of numerals

- We represent numerals in Church style, so m is represented by

  $\underline{m} = \lambda \mathbf{xy}.\mathbf{x}^m\mathbf{y}$

- In STA, $\underline{m}$ can be typed either

  uniformely by $\mathbf{N} = \forall \mathrm{a}.!(\mathrm{a} \multimap \mathrm{a}) \multimap \mathrm{a} \multimap \mathrm{a}$, or

  by $\mathbf{N}_n = \forall \mathrm{a}.!^n(\mathrm{a} \multimap \mathrm{a}) \multimap \mathrm{a} \multimap \mathrm{a}$, for all $n \geq 1$

- In ISTA, $\underline{m}$ can be typed either

  uniformely by $\mathbf{NI} = \forall \mathrm{a}.\{\mathrm{a} \multimap \mathrm{a}\} \multimap \mathrm{a} \multimap \mathrm{a}$, or

  by $\mathbf{NI}_n = \forall \mathrm{a}.\{^n\mathrm{a} \multimap \mathrm{a}\}^n \multimap \mathrm{a} \multimap \mathrm{a}$, for all $n \geq 1$

Note the similarity between $\mathbf{N}_n$ and $\mathbf{NI}_n$

Representation of functions in STA

Let $\phi : \mathcal{N}^p \longrightarrow \mathcal{N}$ be a function of arity p.

- $\mathtt{M}$ represents ϕ in STA iff
  1. $\mathtt{M}\underline{n_1}...\underline{n_p} = \underline{\phi(n_1, ..., n_p)}$
  2. $\mathtt{x}_1 :!^{i_1}\mathbf{N}_{j_1}, ..., \mathtt{x}_p :!^{i_p}\mathbf{N}_{j_p} \vdash_{\mathtt{STA}} \mathtt{M}\mathtt{x}_1...\mathtt{x}_p : \mathbf{N}_j$ for some i_h, j_h, j, $1 \leq h \leq p$

Example:

- Multiplication of numerals has type $\mathbf{N}_i \multimap !^i\mathbf{N}_j \multimap \mathbf{N}_{i+j}$ for all $i, j \geq 1$
- So $\mathtt{x} : \mathbf{N}_i, \mathtt{y} : !^i\mathbf{N}_j \vdash \underline{\mathtt{mult}}\ \mathtt{x}\ \mathtt{y} : \mathbf{N}_{i+j}$

Representation of functions in STA

Let $\phi : \mathcal{N}^p \longrightarrow \mathcal{N}$ be a function of arity p.

- $\mathtt{M}$ represents ϕ in STA iff
  1. $\mathtt{M}\underline{n_1}...\underline{n_p} = \underline{\phi(n_1, ..., n_p)}$
  2. $\mathtt{x}_1 :!^{i_1}\mathbf{N}_{j_1}, ..., \mathtt{x}_p :!^{i_p}\mathbf{N}_{j_p} \vdash_{\mathtt{STA}} \mathtt{M}\mathtt{x}_1...\mathtt{x}_p : \mathbf{N}_j$ for some i_h, j_h, j, $1 \leq h \leq p$

Example:

- Multiplication of numerals has type $\mathbf{N}_i \multimap !^i\mathbf{N}_j \multimap \mathbf{N}_{i+j}$ for all $i, j \geq 1$
- So $\mathtt{x} : \mathbf{N}_i, \mathtt{y} : !^i\mathbf{N}_j \vdash \underline{\mathtt{mult}}\ \mathtt{x}\ \mathtt{y} : \mathbf{N}_{i+j}$

Representation of functions in STA

Let $\phi : \mathcal{N}^p \longrightarrow \mathcal{N}$ be a function of arity p.

- $\mathtt{M}$ represents ϕ in STA iff
  1. $\mathtt{M}\underline{n_1}...\underline{n_p} = \underline{\phi(n_1, ..., n_p)}$
  2. $\mathtt{x}_1 :!^{i_1}\mathbf{N}_{j_1}, ..., \mathtt{x}_p :!^{i_p}\mathbf{N}_{j_p} \vdash_{\mathtt{STA}} \mathtt{M}\mathtt{x}_1...\mathtt{x}_p : \mathbf{N}_j$ for some i_h, j_h, j, $1 \leq h \leq p$

Example:

- Multiplication of numerals has type $\mathbf{N}_i \multimap !^i\mathbf{N}_j \multimap \mathbf{N}_{i+j}$ for all $i, j \geq 1$
- So $\mathtt{x} : \mathbf{N}_i, \mathtt{y} : !^i\mathbf{N}_j \vdash \underline{\mathtt{mult}}\ \mathtt{x}\ \mathtt{y} : \mathbf{N}_{i+j}$

Representation of functions in STA

Let $\phi : \mathcal{N}^p \longrightarrow \mathcal{N}$ be a function of arity p.

- $\mathtt{M}$ represents ϕ in STA iff
  1. $\mathtt{M}\underline{n_1}...\underline{n_p} = \underline{\phi(n_1, ..., n_p)}$
  2. $\mathtt{x}_1 :!^{i_1}\mathbf{N}_{j_1}, ..., \mathtt{x}_p :!^{i_p}\mathbf{N}_{j_p} \vdash_{\mathtt{STA}} \mathtt{M}\mathtt{x}_1...\mathtt{x}_p : \mathbf{N}_j$ for some i_h, j_h, j, $1 \leq h \leq p$

Example:

- Multiplication of numerals has type $\mathbf{N}_i \multimap !^i\mathbf{N}_j \multimap \mathbf{N}_{i+j}$ for all $i, j \geq 1$
- So $\mathtt{x} : \mathbf{N}_i, \mathtt{y} : !^i\mathbf{N}_j \vdash \underline{\mathtt{mult}}\ \mathtt{x}\ \mathtt{y} : \mathbf{N}_{i+j}$

Representation of functions in ISTA

Let $\phi : \mathcal{N}^p \longrightarrow \mathcal{N}$ be a function of arity p.

- $\mathtt{M}$ represents ϕ in ISTA iff
  1. $\mathtt{M}\underline{n_1}...\underline{n_p} = \underline{\phi(n_1, ..., n_p)}$
  2. $\Gamma \mid \Delta \vdash \mathtt{M}\mathbf{x}_1...\mathbf{x}_p : \mathbf{NI}_j$, where $\mathrm{dom}(\Gamma) \cup \mathrm{dom}(\Delta) = \{\mathbf{x}_1, ..., \mathbf{x}_p\}$, and either $\Gamma(\mathbf{x}_i) = \mathbf{NI}_k$ or $\Delta(\mathbf{x}_i) = \sigma_i$ s.t. the linear types of σ are $\mathbf{NI}_{i_1}, ...\mathbf{NI}_{i_m}$ for some k, i_h, j, $1 \leq h \leq p$, $m > 0$

Representation of functions in ISTA

Let $\phi : \mathcal{N}^p \longrightarrow \mathcal{N}$ be a function of arity p.

- M represents ϕ in ISTA iff
  1. $\mathtt{M}\underline{n_1}...\underline{n_p} = \underline{\phi(n_1, ..., n_p)}$
  2. $\Gamma \mid \Delta \vdash \mathtt{M}\mathtt{x}_1...\mathtt{x}_p : \mathbf{NI}_j$, where $\mathrm{dom}(\Gamma) \cup \mathrm{dom}(\Delta) = \{\mathtt{x}_1, ..., \mathtt{x}_p\}$, and either $\Gamma(\mathtt{x}_i) = \mathbf{NI}_k$ or $\Delta(\mathtt{x}_i) = \sigma_i$ s.t. the linear types of σ are $\mathbf{NI}_{i_1}, ...\mathbf{NI}_{i_m}$ for some k, i_h, j, $1 \leq h \leq p$, $m > 0$

Representation of functions in ISTA

Let $\phi : \mathcal{N}^p \longrightarrow \mathcal{N}$ be a function of arity p.

- $\mathtt{M}$ represents ϕ in ISTA iff
  1. $\mathtt{M}\underline{n_1}...\underline{n_p} = \underline{\phi(n_1, ..., n_p)}$
  2. $\Gamma \mid \Delta \vdash \mathtt{M}\mathtt{x}_1...\mathtt{x}_p : \mathbf{NI}_j$, where $\mathrm{dom}(\Gamma) \cup \mathrm{dom}(\Delta) = \{\mathtt{x}_1, ..., \mathtt{x}_p\}$, and either $\Gamma(\mathtt{x}_i) = \mathbf{NI}_k$ or $\Delta(\mathtt{x}_i) = \sigma_i$ s.t. the linear types of σ are $\mathbf{NI}_{i_1}, ...\mathbf{NI}_{i_m}$ for some k, i_h, j, $1 \leq h \leq p$, $m > 0$

FPTIME soundness

Translation from types of STA to types of ISTA

$(a)^\circ = a$; $(\mu \multimap \mathtt{U})^\circ = (\mu)^\circ \multimap (\mathtt{U})^\circ$ or $\mathtt{A} \multimap (\mathtt{U})^\circ$ for some $\mathtt{A}$; $(!\mu)^\circ = \{(\mu)^\circ\}$

but the translation is not deterministic, i.e. it depends on the derivation!

$(.)^\circ$ can be easily extended to derivations, so

$$(\Theta \vdash_{\mathtt{STA}} \mathtt{M} : \mu)^\circ = (\Theta_l)^\circ \mid (\Theta_s)^\circ \vdash \mathtt{M} : (\mu)^\circ$$

FPTIME soundness

In ISTA all the FPTIME functions are definable.

Proof. Every proof in STA can be translated in a proof in ISTA with the same subject.

FPTIME soundness

Translation from types of STA to types of ISTA

$(a)^\circ = a$; $(\mu \multimap \mathtt{U})^\circ = (\mu)^\circ \multimap (\mathtt{U})^\circ$ or $\mathtt{A} \multimap (\mathtt{U})^\circ$ for some $\mathtt{A}$; $(!\mu)^\circ = \{(\mu)^\circ\}$

but the translation is not deterministic, i.e. it depends on the derivation!

$(.)^\circ$ can be easily extended to derivations, so

$$(\Theta \vdash_{\mathtt{STA}} \mathtt{M} : \mu)^\circ = (\Theta_l)^\circ \mid (\Theta_s)^\circ \vdash \mathtt{M} : (\mu)^\circ$$

FPTIME soundness

In ISTA all the FPTIME functions are definable.

Proof. Every proof in STA can be translated in a proof in ISTA with the same subject.

FPTIME soundness

Translation from types of STA to types of ISTA

$(a)^\circ = a$; $(\mu \multimap \mathtt{U})^\circ = (\mu)^\circ \multimap (\mathtt{U})^\circ$ or $\mathtt{A} \multimap (\mathtt{U})^\circ$ for some $\mathtt{A}$; $(!\mu)^\circ = \{(\mu)^\circ\}$

but the translation is not deterministic, i.e. it depends on the derivation!

$(.)^\circ$ can be easily extended to derivations, so

$$(\Theta \vdash_{\mathtt{STA}} \mathtt{M} : \mu)^\circ = (\Theta_l)^\circ \mid (\Theta_s)^\circ \vdash \mathtt{M} : (\mu)^\circ$$

FPTIME soundness

In ISTA all the FPTIME functions are definable.

Proof. Every proof in STA can be translated in a proof in ISTA with the same subject.

Translation from ISTA to STA

There is a translation $(.)^*$ of derivations ending in linear types, so

$$(\pi : \mathbf{\Gamma} \mid \mathbf{\Delta} \vdash \mathtt{M} : \mathtt{A})^* = (\pi)^* : \mathbf{\Theta} \vdash_{\mathtt{STA}} (\mathtt{M})^* : (\mathtt{A})^*$$

such that $(\mathtt{M})^*$ is a partial linearization of $\mathtt{M}$

Translation from ISTA to STA

There is a translation $(.)^*$ of derivations ending in linear types, so

$$(\pi : \mathbf{\Gamma} \mid \mathbf{\Delta} \vdash \mathtt{M} : \mathtt{A})^* = (\pi)^* : \mathbf{\Theta} \vdash_{\mathtt{STA}} (\mathtt{M})^* : (\mathtt{A})^*$$

such that $(\mathtt{M})^*$ is a partial linearization of $\mathtt{M}$

Linearization

Example 1

Let $\mathtt{M} = \lambda \mathtt{x}.\mathtt{xx}$:

- in ISTA, $\vdash \lambda \mathtt{x}.\mathtt{xx} : \{\{\mathtt{b} \multimap \mathtt{a}\}, \{\mathtt{b}\}\} \multimap \mathtt{a}$
  - the translated proof in STA is $\vdash \lambda \mathtt{x}_1.\lambda \mathtt{x}_2.\mathtt{x}_1\mathtt{x}_2 : !!(\mathtt{b} \multimap \mathtt{a}) \multimap !!\mathtt{b} \multimap \mathtt{a}$

So $(\mathtt{M})^*$ is the term $\mathtt{M}$ in which the two occurrences of $\mathtt{x}$ are substituted by $\mathtt{x}_1$ and $\mathtt{x}_2$

Example 2

Let $\mathtt{N} = \lambda \mathtt{x}.\lambda \mathtt{y}.\mathtt{yxx}$:

- in ISTA, $\vdash \lambda \mathtt{x}.\lambda \mathtt{y}.\mathtt{yxx} : \{\mathtt{a}\} \multimap (\mathtt{a} \multimap \mathtt{a} \multimap \mathtt{a}) \multimap \mathtt{a}$
  - the translated proof in STA is $\vdash \lambda \mathtt{x}.\lambda \mathtt{y}.\mathtt{yxx} : !\mathtt{a} \multimap (\mathtt{a} \multimap \mathtt{a} \multimap \mathtt{a}) \multimap \mathtt{a}$

So $(\mathtt{N})^* = \mathtt{N}$

Linearization

Example 1

Let $\mathtt{M} = \lambda \mathtt{x}.\mathtt{xx}$:

- in $\mathtt{ISTA}$, $\vdash \lambda \mathtt{x}.\mathtt{xx} : \{\{\mathtt{b} \multimap \mathtt{a}\}, \{\mathtt{b}\}\} \multimap \mathtt{a}$
- the translated proof in $\mathtt{STA}$ is $\vdash \lambda \mathtt{x}_1.\lambda \mathtt{x}_2.\mathtt{x}_1\mathtt{x}_2 : !!(\mathtt{b} \multimap \mathtt{a}) \multimap !!\mathtt{b} \multimap \mathtt{a}$

So $(\mathtt{M})^*$ is the term $\mathtt{M}$ in which the two occurrences of $\mathtt{x}$ are substituted by $\mathtt{x}_1$ and $\mathtt{x}_2$

Example 2

Let $\mathtt{N} = \lambda \mathtt{x}.\lambda \mathtt{y}.\mathtt{yxx}$:

- in $\mathtt{ISTA}$, $\vdash \lambda \mathtt{x}.\lambda \mathtt{y}.\mathtt{yxx} : \{\mathtt{a}\} \multimap (\mathtt{a} \multimap \mathtt{a} \multimap \mathtt{a}) \multimap \mathtt{a}$
- the translated proof in $\mathtt{STA}$ is $\vdash \lambda \mathtt{x}.\lambda \mathtt{y}.\mathtt{yxx} : !\mathtt{a} \multimap (\mathtt{a} \multimap \mathtt{a} \multimap \mathtt{a}) \multimap \mathtt{a}$

So $(\mathtt{N})^* = \mathtt{N}$

Linearization

Example 1

Let $\mathtt{M} = \lambda \mathtt{x}.\mathtt{xx}$:

- in $\mathtt{ISTA}$, $\vdash \lambda \mathtt{x}.\mathtt{xx} : \{\{\mathtt{b} \multimap \mathtt{a}\}, \{\mathtt{b}\}\} \multimap \mathtt{a}$
- the translated proof in $\mathtt{STA}$ is $\vdash \lambda \mathtt{x}_1.\lambda \mathtt{x}_2.\mathtt{x}_1\mathtt{x}_2 : !!(\mathtt{b} \multimap \mathtt{a}) \multimap !!\mathtt{b} \multimap \mathtt{a}$

So $(\mathtt{M})^*$ is the term $\mathtt{M}$ in which the two occurrences of $\mathtt{x}$ are substituted by $\mathtt{x}_1$ and $\mathtt{x}_2$

Example 2

Let $\mathtt{N} = \lambda \mathtt{x}.\lambda \mathtt{y}.\mathtt{yxx}$:

- in $\mathtt{ISTA}$, $\vdash \lambda \mathtt{x}.\lambda \mathtt{y}.\mathtt{yxx} : \{\mathtt{a}\} \multimap (\mathtt{a} \multimap \mathtt{a} \multimap \mathtt{a}) \multimap \mathtt{a}$
- the translated proof in $\mathtt{STA}$ is $\vdash \lambda \mathtt{x}.\lambda \mathtt{y}.\mathtt{yxx} : !\mathtt{a} \multimap (\mathtt{a} \multimap \mathtt{a} \multimap \mathtt{a}) \multimap \mathtt{a}$

So $(\mathtt{N})^* = \mathtt{N}$

Linearization

Example 1

Let $\mathtt{M} = \lambda \mathtt{x}.\mathtt{xx}$:

- in ISTA, $\vdash \lambda \mathtt{x}.\mathtt{xx} : \{\{\mathtt{b} \multimap \mathtt{a}\}, \{\mathtt{b}\}\} \multimap \mathtt{a}$
- the translated proof in STA is $\vdash \lambda \mathtt{x}_1.\lambda \mathtt{x}_2.\mathtt{x}_1\mathtt{x}_2 : !!(\mathtt{b} \multimap \mathtt{a}) \multimap !!\mathtt{b} \multimap \mathtt{a}$

So $(\mathtt{M})^*$ is the term $\mathtt{M}$ in which the two occurrences of $\mathtt{x}$ are substituted by $\mathtt{x}_1$ and $\mathtt{x}_2$

Example 2

Let $\mathtt{N} = \lambda \mathtt{x}.\lambda \mathtt{y}.\mathtt{yxx}$:

- in ISTA, $\vdash \lambda \mathtt{x}.\lambda \mathtt{y}.\mathtt{yxx} : \{\mathtt{a}\} \multimap (\mathtt{a} \multimap \mathtt{a} \multimap \mathtt{a}) \multimap \mathtt{a}$
- the translated proof in STA is $\vdash \lambda \mathtt{x}.\lambda \mathtt{y}.\mathtt{yxx} : !\mathtt{a} \multimap (\mathtt{a} \multimap \mathtt{a} \multimap \mathtt{a}) \multimap \mathtt{a}$

So $(\mathtt{N})^* = \mathtt{N}$

Linearization

Example 1

Let $\mathtt{M} = \lambda \mathtt{x}.\mathtt{xx}$:

- in $\mathtt{ISTA}$, $\vdash \lambda \mathtt{x}.\mathtt{xx} : \{\{\mathtt{b} \multimap \mathtt{a}\}, \{\mathtt{b}\}\} \multimap \mathtt{a}$
- the translated proof in $\mathtt{STA}$ is $\vdash \lambda \mathtt{x}_1.\lambda \mathtt{x}_2.\mathtt{x}_1\mathtt{x}_2 : !!(\mathtt{b} \multimap \mathtt{a}) \multimap !!\mathtt{b} \multimap \mathtt{a}$

So $(\mathtt{M})^*$ is the term $\mathtt{M}$ in which the two occurrences of $\mathtt{x}$ are substituted by $\mathtt{x}_1$ and $\mathtt{x}_2$

Example 2

Let $\mathtt{N} = \lambda \mathtt{x}.\lambda \mathtt{y}.\mathtt{yxx}$:

- in $\mathtt{ISTA}$, $\vdash \lambda \mathtt{x}.\lambda \mathtt{y}.\mathtt{yxx} : \{\mathtt{a}\} \multimap (\mathtt{a} \multimap \mathtt{a} \multimap \mathtt{a}) \multimap \mathtt{a}$
- the translated proof in $\mathtt{STA}$ is $\vdash \lambda \mathtt{x}.\lambda \mathtt{y}.\mathtt{yxx} : !\mathtt{a} \multimap (\mathtt{a} \multimap \mathtt{a} \multimap \mathtt{a}) \multimap \mathtt{a}$

So $(\mathtt{N})^* = \mathtt{N}$

Linearization

Example 1

Let $\mathtt{M} = \lambda \mathtt{x}.\mathtt{xx}$:

- in $\mathtt{ISTA}$, $\vdash \lambda \mathtt{x}.\mathtt{xx} : \{\{\mathtt{b} \multimap \mathtt{a}\}, \{\mathtt{b}\}\} \multimap \mathtt{a}$
- the translated proof in $\mathtt{STA}$ is $\vdash \lambda \mathtt{x}_1.\lambda \mathtt{x}_2.\mathtt{x}_1\mathtt{x}_2 : !!(\mathtt{b} \multimap \mathtt{a}) \multimap !!\mathtt{b} \multimap \mathtt{a}$

So $(\mathtt{M})^*$ is the term $\mathtt{M}$ in which the two occurrences of $\mathtt{x}$ are substituted by $\mathtt{x}_1$ and $\mathtt{x}_2$

Example 2

Let $\mathtt{N} = \lambda \mathtt{x}.\lambda \mathtt{y}.\mathtt{yxx}$:

- in $\mathtt{ISTA}$, $\vdash \lambda \mathtt{x}.\lambda \mathtt{y}.\mathtt{yxx} : \{\mathtt{a}\} \multimap (\mathtt{a} \multimap \mathtt{a} \multimap \mathtt{a}) \multimap \mathtt{a}$
- the translated proof in $\mathtt{STA}$ is $\vdash \lambda \mathtt{x}.\lambda \mathtt{y}.\mathtt{yxx} : !\mathtt{a} \multimap (\mathtt{a} \multimap \mathtt{a} \multimap \mathtt{a}) \multimap \mathtt{a}$

So $(\mathtt{N})^* = \mathtt{N}$

Conservativity

Let $\mathtt{M}$ represent in $\mathtt{ISTA}$ a program computing a function $\phi : \mathcal{N} \longrightarrow \mathcal{N}$, such that

$$\pi :\vdash \mathtt{M} : \sigma \multimap \mathbf{NI}_j$$

where the elements of σ are $\mathbf{NI}_{j_1}, ..., \mathbf{NI}_{j_m}$, and let the corresponding derivation in $\mathtt{STA}$ be

$$(\pi)^* :\vdash_{\mathtt{STA}} (\mathtt{M})^* : !^{i_1}\mathbf{N}_{j_1} \multimap ... \multimap !^{i_m}\mathbf{N}_{j_m} \multimap \mathbf{N}_j$$

If $(\mathtt{M})^*$ represent the function ψ, then

$$\phi(n) = \psi(\underbrace{n...n}_{m})$$

Conservativity

Let $\mathtt{M}$ represent in $\mathtt{ISTA}$ a program computing a function $\phi : \mathcal{N} \longrightarrow \mathcal{N}$, such that

$$\pi :\vdash \mathtt{M} : \sigma \multimap \mathbf{NI}_j$$

where the elements of σ are $\mathbf{NI}_{j_1}, ..., \mathbf{NI}_{j_m}$, and let the corresponding derivation in $\mathtt{STA}$ be

$$(\pi)^* :\vdash_{\mathtt{STA}} (\mathtt{M})^* : !^{i_1}\mathbf{N}_{j_1} \multimap ... \multimap !^{i_m}\mathbf{N}_{j_m} \multimap \mathbf{N}_j$$

If $(\mathtt{M})^*$ represent the function ψ, then

$$\phi(n) = \psi(\underbrace{n...n}_{m})$$

Conservativity

Let $\mathtt{M}$ represent in $\mathtt{ISTA}$ a program computing a function $\phi : \mathcal{N} \longrightarrow \mathcal{N}$, such that

$$\pi :\vdash \mathtt{M} : \sigma \multimap \mathbf{NI}_j$$

where the elements of σ are $\mathbf{NI}_{j_1}, ..., \mathbf{NI}_{j_m}$, and let the corresponding derivation in $\mathtt{STA}$ be

$$(\pi)^* :\vdash_{\mathtt{STA}} (\mathtt{M})^* : !^{i_1}\mathbf{N}_{j_1} \multimap ... \multimap !^{i_m}\mathbf{N}_{j_m} \multimap \mathbf{N}_j$$

If $(\mathtt{M})^*$ represent the function ψ, then

$$\phi(n) = \psi(\underbrace{n...n}_{m})$$

Conservativity: proof

Moreover, there is a term in STA computing ϕ: its normal form is

$$\vdash_{\mathtt{STA}} \lambda \mathtt{x}.\mathtt{x}II(\lambda \mathtt{z}.\mathtt{z}\underbrace{(\lambda \mathtt{zw}.\mathtt{xzw})...(\lambda \mathtt{zw}.\mathtt{xzw})}_{m})(\mathtt{M})^* : !^k\mathbf{N} \multimap \mathbf{N}_j$$

for any $\mathbf{k} \geq \max\{\mathbf{i_1}, ..., \mathbf{i_m}\} + \mathbf{2}$

In the proof we exploit two properties of STA:

1. if $\mathtt{x}$ has type $!^i\mathtt{A}$ in the context, then $\mathtt{x} : !^{\geq i}\mathtt{A}$ by a sequence of applications of rule (m)
2. there is a coercion from any $\mathbf{N}_i$ to $\mathbf{N}_{\geq i}$, i.e. $\vdash_{\mathtt{STA}} \lambda \mathtt{xzw}.\mathtt{xzw} : \mathbf{N}_i \multimap \mathbf{N}_j$ for any $j \geq i$
   - In fact, $(\lambda \mathtt{xzw}.\mathtt{xzw})\underline{n} \xrightarrow[\beta]{*} \underline{n}$

Conservativity: proof

Moreover, there is a term in STA computing ϕ: its normal form is

$$\vdash_{\mathtt{STA}} \lambda \mathtt{x}.\mathtt{x}II(\lambda \mathtt{z}.\mathtt{z}\underbrace{(\lambda \mathtt{zw}.\mathtt{xzw})...(\lambda \mathtt{zw}.\mathtt{xzw})}_{m})(\mathtt{M})^* : !^k\mathbf{N} \multimap \mathbf{N}_j$$

for any $\mathbf{k} \geq \max\{\mathbf{i_1}, ..., \mathbf{i_m}\} + \mathbf{2}$

In the proof we exploit two properties of STA:

1. if $\mathtt{x}$ has type $!^i\mathtt{A}$ in the context, then $\mathtt{x} : !^{\geq i}\mathtt{A}$ by a sequence of applications of rule (m)
2. there is a coercion from any $\mathbf{N}_i$ to $\mathbf{N}_{\geq i}$, i.e. $\vdash_{\mathtt{STA}} \lambda \mathtt{xzw}.\mathtt{xzw} : \mathbf{N}_i \multimap \mathbf{N}_j$ for any $j \geq i$
   - In fact, $(\lambda \mathtt{xzw}.\mathtt{xzw})\underline{n} \xrightarrow[\beta]{*} \underline{n}$

Conservativity: proof

Moreover, there is a term in STA computing ϕ: its normal form is

$$\vdash_{\mathtt{STA}} \lambda \mathtt{x}.\mathtt{x}II(\lambda \mathtt{z}.\mathtt{z}\underbrace{(\lambda \mathtt{zw}.\mathtt{xzw})...(\lambda \mathtt{zw}.\mathtt{xzw})}_{m})(\mathtt{M})^* : !^k\mathbf{N} \multimap \mathbf{N}_j$$

for any $\mathbf{k} \geq \max\{\mathbf{i_1}, ..., \mathbf{i_m}\} + \mathbf{2}$

In the proof we exploit two properties of STA:

1. if $\mathtt{x}$ has type $!^i\mathtt{A}$ in the context, then $\mathtt{x} : !^{\geq i}\mathtt{A}$ by a sequence of applications of rule (m)
2. there is a coercion from any $\mathbf{N}_i$ to $\mathbf{N}_{\geq i}$, i.e. $\vdash_{\mathtt{STA}} \lambda \mathtt{xzw.xzw} : \mathbf{N}_i \multimap \mathbf{N}_j$ for any $j \geq i$
   - In fact, $(\lambda \mathtt{xzw.xzw})\underline{n} \xrightarrow[\beta]{*} \underline{n}$

FPTIME completeness

The term in normal form: $\lambda \mathtt{x}.\mathtt{x}II(\lambda \mathtt{z}.\mathtt{z}(\lambda \mathtt{zw}.\mathtt{xzw})_1 ... (\lambda \mathtt{zw}.\mathtt{xzw})_m)(\mathtt{M})^*$

represents the desired function:

$$(\lambda \mathtt{x}.\mathtt{x}II(\lambda \mathtt{z}.\mathtt{z}(\lambda \mathtt{zw}.\mathtt{xzw})_1 ... (\lambda \mathtt{zw}.\mathtt{xzw})_m)(\mathtt{M})^* \underline{n} \xrightarrow[\beta]{}$$

$$\underline{n}II(\lambda \mathtt{z}.\mathtt{z}(\lambda \mathtt{zw}.\underline{n}\mathtt{zw})_1 ... (\lambda \mathtt{zw}.\underline{n}\mathtt{zw})_m)(\mathtt{M})^* \xrightarrow[\beta]{*}$$

$$(\lambda \mathtt{z}.\mathtt{z}(\lambda \mathtt{zw}.\underline{n}\mathtt{zw})_1 ... (\lambda \mathtt{zw}.\underline{n}\mathtt{zw})_m)(\mathtt{M})^* \xrightarrow[\beta]{}$$

$$(\mathtt{M})^*(\lambda \mathtt{zw}.\underline{n}\mathtt{zw})_1 ... (\lambda \mathtt{zw}.\underline{n}\mathtt{zw})_m \xrightarrow[\beta]{*} (\mathtt{M})^* \underbrace{\underline{n} ... \underline{n}}_{m}$$

Theorem: FPTIME Completeness

ISTA is complete for FPTIME.

FPTIME completeness

The term in normal form: $\lambda \mathtt{x}.\mathtt{x}II(\lambda \mathtt{z}.\mathtt{z}(\lambda \mathtt{zw}.\mathtt{xzw})_1 ... (\lambda \mathtt{zw}.\mathtt{xzw})_m)(\mathtt{M})^*$ represents the desired function:

$$(\lambda \mathtt{x}.\mathtt{x}II(\lambda \mathtt{z}.\mathtt{z}(\lambda \mathtt{zw}.\mathtt{xzw})_1 ... (\lambda \mathtt{zw}.\mathtt{xzw})_m)(\mathtt{M})^* \underline{n} \xrightarrow[\beta]{}$$

$$\underline{n}II(\lambda \mathtt{z}.\mathtt{z}(\lambda \mathtt{zw}.\underline{n}\mathtt{zw})_1 ... (\lambda \mathtt{zw}.\underline{n}\mathtt{zw})_m)(\mathtt{M})^* \xrightarrow[\beta]{*}$$

$$(\lambda \mathtt{z}.\mathtt{z}(\lambda \mathtt{zw}.\underline{n}\mathtt{zw})_1 ... (\lambda \mathtt{zw}.\underline{n}\mathtt{zw})_m)(\mathtt{M})^* \xrightarrow[\beta]{}$$

$$(\mathtt{M})^*(\lambda \mathtt{zw}.\underline{n}\mathtt{zw})_1 ... (\lambda \mathtt{zw}.\underline{n}\mathtt{zw})_m \xrightarrow[\beta]{*} (\mathtt{M})^* \underbrace{\underline{n} ... \underline{n}}_{m}$$

Theorem: FPTIME Completeness

ISTA is complete for FPTIME.

FPTIME completeness

The term in normal form: $\lambda \mathtt{x}.\mathtt{x}II(\lambda \mathtt{z}.\mathtt{z}(\lambda \mathtt{zw}.\mathtt{xzw})_1 ... (\lambda \mathtt{zw}.\mathtt{xzw})_m)(\mathtt{M})^*$ represents the desired function:

$$(\lambda \mathtt{x}.\mathtt{x}II(\lambda \mathtt{z}.\mathtt{z}(\lambda \mathtt{zw}.\mathtt{xzw})_1 ... (\lambda \mathtt{zw}.\mathtt{xzw})_m)(\mathtt{M})^* \underline{n} \xrightarrow[\beta]{}$$

$$\underline{n}II(\lambda \mathtt{z}.\mathtt{z}(\lambda \mathtt{zw}.\underline{n}\mathtt{zw})_1 ... (\lambda \mathtt{zw}.\underline{n}\mathtt{zw})_m)(\mathtt{M})^* \xrightarrow[\beta]{*}$$

$$(\lambda \mathtt{z}.\mathtt{z}(\lambda \mathtt{zw}.\underline{n}\mathtt{zw})_1 ... (\lambda \mathtt{zw}.\underline{n}\mathtt{zw})_m)(\mathtt{M})^* \xrightarrow[\beta]{}$$

$$(\mathtt{M})^*(\lambda \mathtt{zw}.\underline{n}\mathtt{zw})_1 ... (\lambda \mathtt{zw}.\underline{n}\mathtt{zw})_m \xrightarrow[\beta]{*} (\mathtt{M})^* \underbrace{\underline{n} ... \underline{n}}_{m}$$

Theorem: FPTIME Completeness

ISTA is complete for FPTIME.

FPTIME completeness

The term in normal form: $\lambda \mathtt{x}.\mathtt{x}II(\lambda \mathtt{z}.\mathtt{z}(\lambda \mathtt{zw}.\mathtt{xzw})_1 ... (\lambda \mathtt{zw}.\mathtt{xzw})_m)(\mathtt{M})^*$ represents the desired function:

$$(\lambda \mathtt{x}.\mathtt{x}II(\lambda \mathtt{z}.\mathtt{z}(\lambda \mathtt{zw}.\mathtt{xzw})_1 ... (\lambda \mathtt{zw}.\mathtt{xzw})_m)(\mathtt{M})^* \underline{n} \xrightarrow[\beta]{}$$

$$\underline{n}II(\lambda \mathtt{z}.\mathtt{z}(\lambda \mathtt{zw}.\underline{n}\mathtt{zw})_1 ... (\lambda \mathtt{zw}.\underline{n}\mathtt{zw})_m)(\mathtt{M})^* \xrightarrow[\beta]{*}$$

$$(\lambda \mathtt{z}.\mathtt{z}(\lambda \mathtt{zw}.\underline{n}\mathtt{zw})_1 ... (\lambda \mathtt{zw}.\underline{n}\mathtt{zw})_m)(\mathtt{M})^* \xrightarrow[\beta]{}$$

$$(\mathtt{M})^*(\lambda \mathtt{zw}.\underline{n}\mathtt{zw})_1 ... (\lambda \mathtt{zw}.\underline{n}\mathtt{zw})_m \xrightarrow[\beta]{*} (\mathtt{M})^* \underbrace{\underline{n} ... \underline{n}}_{m}$$

Theorem: FPTIME Completeness

ISTA is complete for FPTIME.

Iteration of succ

- In ISTA it is possible to type higher order functionals such as

$$\mathtt{ITER}^3\ \underline{\mathtt{succ}}\ \underline{\mathtt{n}}$$

which is a more natural way of writing a program rather than

$$\underline{\mathtt{succ}}\ (\underline{\mathtt{succ}}\ (\underline{\mathtt{succ}}\ \underline{\mathtt{n}}))$$

- $\vdash \mathtt{ITER}^3 : \{\{\mathbf{NI}_i \multimap \mathbf{NI}_{i+1}\}, \{\mathbf{NI}_{i+1} \multimap \mathbf{NI}_{i+2}\}, \{\mathbf{NI}_{i+2} \multimap \mathbf{NI}_{i+3}\}\} \multimap \mathbf{NI}_i \multimap \mathbf{NI}_{i+3}$, so the term is not typed in $\mathtt{STA}_{¡¡¡¡¡¡¡}$ .mine
- The term can also be written in normal form as iter¡4¿λx.λy.$\mathtt{ITER}^3$xy
======= ¿¿¿¿¿¿¿ .r49

Iteration of succ

- In ISTA it is possible to type higher order functionals such as

$$\mathtt{ITER}^3\ \underline{\mathtt{succ}}\ \underline{\mathtt{n}}$$

which is a more natural way of writing a program rather than

$$\underline{\mathtt{succ}}\ (\underline{\mathtt{succ}}\ (\underline{\mathtt{succ}}\ \underline{\mathtt{n}}))$$

- $\vdash \mathtt{ITER}^3 : \{\{\mathbf{NI}_i \multimap \mathbf{NI}_{i+1}\}, \{\mathbf{NI}_{i+1} \multimap \mathbf{NI}_{i+2}\}, \{\mathbf{NI}_{i+2} \multimap \mathbf{NI}_{i+3}\}\} \multimap \mathbf{NI}_i \multimap \mathbf{NI}_{i+3}$, so the term is not typed in STA¡¡¡¡¡¡¡ .mine
- The term can also be written in normal form as iter¡4¿λx.λy.$\mathtt{ITER}^3$xy ======= ¿¿¿¿¿¿¿ .r49

Iteration of succ

- In ISTA it is possible to type higher order functionals such as

$$\mathtt{ITER}^3\ \underline{\mathtt{succ}}\ \underline{\mathtt{n}}$$

which is a more natural way of writing a program rather than

$$\underline{\mathtt{succ}}\ (\underline{\mathtt{succ}}\ (\underline{\mathtt{succ}}\ \underline{\mathtt{n}}))$$

- $\vdash \mathtt{ITER}^3 : \{\{\mathbf{NI}_i \multimap \mathbf{NI}_{i+1}\}, \{\mathbf{NI}_{i+1} \multimap \mathbf{NI}_{i+2}\}, \{\mathbf{NI}_{i+2} \multimap \mathbf{NI}_{i+3}\}\} \multimap \mathbf{NI}_i \multimap \mathbf{NI}_{i+3}$, so the term is not typed in $\mathtt{STA}_{¡¡¡¡¡¡¡}$ .mine
- The term can also be written in normal form as iter¡4¿λx.λy.$\mathtt{ITER}^3$xy ======= ¿¿¿¿¿¿¿ .r49

Iteration of succ

- In ISTA it is possible to type higher order functionals such as

$$\mathtt{ITER}^3\ \underline{\mathtt{succ}}\ \underline{\mathtt{n}}$$

which is a more natural way of writing a program rather than

$$\underline{\mathtt{succ}}\ (\underline{\mathtt{succ}}\ (\underline{\mathtt{succ}}\ \underline{\mathtt{n}}))$$

- $\vdash \mathtt{ITER}^3 : \{\{\mathbf{NI}_i \multimap \mathbf{NI}_{i+1}\}, \{\mathbf{NI}_{i+1} \multimap \mathbf{NI}_{i+2}\}, \{\mathbf{NI}_{i+2} \multimap \mathbf{NI}_{i+3}\}\} \multimap \mathbf{NI}_i \multimap \mathbf{NI}_{i+3}$, so the term is not typed in $\mathtt{STA}_{¡¡¡¡¡¡¡}$ .mine
- The term can also be written in normal form as iter¡4¿$\lambda\mathtt{x}.\lambda\mathtt{y}.\mathtt{ITER}^3\mathtt{xy}$ ======= ¿¿¿¿¿¿¿ .r49

Iteration of $\underline{\texttt{mult}}$

- On the other hand, in ISTA terms such as

$$\texttt{ITER}^3\ \underline{\texttt{mult}}\ \underline{n}\ \underline{m}$$

  cannot be typed

- In fact

$$\texttt{ITER}^3 : \forall a_1.\forall a_2.\forall a_3.\forall a_4.\{\{a_1 \multimap a_2\},\{a_2 \multimap a_3\},\{a_3 \multimap a_4\}\} \multimap a_1 \multimap a_4$$

- Since $\underline{\texttt{mult}} : \mathbf{NI}_i \multimap \{{}^i\mathbf{NI}_j\}^i \multimap \mathbf{NI}_{i+j}$, the presence of the stratified type prevents $\texttt{ITER}^3$ from being typed

Iteration of $\underline{\texttt{mult}}$

- On the other hand, in ISTA terms such as

$$\texttt{ITER}^3\ \underline{\texttt{mult}}\ \underline{n}\ \underline{m}$$

cannot be typed

- In fact

$$\texttt{ITER}^3 : \forall a_1.\forall a_2.\forall a_3.\forall a_4.\{\{a_1 \multimap a_2\}, \{a_2 \multimap a_3\}, \{a_3 \multimap a_4\}\} \multimap a_1 \multimap a_4$$

- Since $\underline{\texttt{mult}} : \mathbf{NI}_i \multimap \{{}^i\mathbf{NI}_j\}^i \multimap \mathbf{NI}_{i+j}$, the presence of the stratified type prevents $\texttt{ITER}^3$ from being typed

Iteration of $\underline{\texttt{mult}}$

- On the other hand, in ISTA terms such as

$$\texttt{ITER}^3\ \underline{\texttt{mult}}\ \underline{n}\ \underline{m}$$

  cannot be typed

- In fact

$$\texttt{ITER}^3 : \forall a_1.\forall a_2.\forall a_3.\forall a_4.\{\{a_1 \multimap a_2\},\{a_2 \multimap a_3\},\{a_3 \multimap a_4\}\} \multimap a_1 \multimap a_4$$

- Since $\underline{\texttt{mult}} : \mathbf{NI}_i \multimap \{{}^i\mathbf{NI}_j\}^i \multimap \mathbf{NI}_{i+j}$, the presence of the stratified type prevents $\texttt{ITER}^3$ from being typed

Conclusions

- ISTA is sound and complete w.r.t. FPTIME
- ISTA is sound and complete w.r.t. strong normalization
- In ISTA all numerical polynomial algorithms that can be expressed by a strongly normalizing term are typed
- Adding stratification types to STA types does not increase the definability of polynomial functions, but adds expressivity to the language

Conclusions

- ISTA is sound and complete w.r.t. FPTIME
- ISTA is sound and complete w.r.t. strong normalization
- In ISTA all numerical polynomial algorithms that can be expressed by a strongly normalizing term are typed
- Adding stratification types to STA types does not increase the definability of polynomial functions, but adds expressivity to the language

Conclusions

- ISTA is sound and complete w.r.t. FPTIME
- ISTA is sound and complete w.r.t. strong normalization
- In ISTA all numerical polynomial algorithms that can be expressed by a strongly normalizing term are typed
- Adding stratification types to STA types does not increase the definability of polynomial functions, but adds expressivity to the language

Conclusions

- ISTA is sound and complete w.r.t. FPTIME
- ISTA is sound and complete w.r.t. strong normalization
- In ISTA all numerical polynomial algorithms that can be expressed by a strongly normalizing term are typed
- Adding stratification types to STA types does not increase the definability of polynomial functions, but adds expressivity to the language