Free-cut elimination in linear logic and an application to a feasible arithmetic

Anupam Das   Patrick Baillot

LIP, Université de Lyon, CNRS, ENS de Lyon, INRIA, Université Claude-Bernard Lyon 1, Milyon

6th October, 2016

Bologne
ELICA meeting

Outline

Introduction

Normal forms in first-order linear logic

An arithmetic in linear logic

Bellantoni-Cook programs and the WFM for $I\Sigma_1^{N^+}$

Conclusions

Implicit computational complexity (ICC)

Implicit computational complexity (ICC)

In a nutshell:

ICC studies correspondences between features of logic and complexity classes

In a nutshell:

ICC studies correspondences between features of logic and complexity classes

Proof-theoretic approach

For a logic or theory,

'representable' functions = given complexity class

where representability can mean definability, typability etc.

Implicit computational complexity (ICC)

In a nutshell:

ICC studies correspondences between features of logic and complexity classes

Proof-theoretic approach

For a logic or theory,

'representable' functions = given complexity class

where representability can mean definability, typability etc.

We distinguish the following two methodologies:

1. Theories whose definable functions = given complexity class.
2. Logics that type terms with normalisation complexity of a given class.

Implicit computational complexity (ICC)

In a nutshell:

ICC studies correspondences between features of logic and complexity classes

Proof-theoretic approach

For a logic or theory,

'representable' functions = given complexity class

where representability can mean definability, typability etc.

We distinguish the following two methodologies:

1. Theories whose definable functions = given complexity class.
2. Logics that type terms with normalisation complexity of a given class.

This work is about the first methodology.

Provably convergent functions

Correspondence between a theory $\mathcal{T}$ and a class $\mathcal{C}$:

$$\mathcal{T} \vdash \forall x.\exists y.A(x, y) \quad \Leftrightarrow \quad \mathbb{N} \models \forall x.A(x, \mathfrak{f}(x)) \text{ for some } \mathfrak{f} \in \mathcal{C}$$

Correspondence between a theory $\mathcal{T}$ and a class $\mathcal{C}$:

$$\mathcal{T} \vdash \forall x.\exists y.A(x,y) \quad \Leftrightarrow \quad \mathbb{N} \models \forall x.A(x,\mathfrak{f}(x)) \text{ for some } \mathfrak{f} \in \mathcal{C}$$

For example:

Theorem (Parsons '68, Mints '73, Buss '95)

$I\Sigma_1$ proves the totality of precisely the primitive recursive functions.

Provably convergent functions

Correspondence between a theory $\mathcal{T}$ and a class $\mathcal{C}$:

$$\mathcal{T} \vdash \forall x.\exists y.A(x,y) \quad \Leftrightarrow \quad \mathbb{N} \models \forall x.A(x,\mathfrak{f}(x)) \text{ for some } \mathfrak{f} \in \mathcal{C}$$

For example:

Theorem (Parsons '68, Mints '73, Buss '95)

$I\Sigma_1$ proves the totality of precisely the primitive recursive functions.

Parsons' proof.

- Via a Dialectica-style functional interpretation.
- **Extracted programs:** higher-order variant of primitive recursive functions.

□

Provably convergent functions

Correspondence between a theory $\mathcal{T}$ and a class $\mathcal{C}$:

$$\mathcal{T} \vdash \forall x.\exists y.A(x,y) \quad \Leftrightarrow \quad \mathbb{N} \models \forall x.A(x,\mathfrak{f}(x)) \text{ for some } \mathfrak{f} \in \mathcal{C}$$

For example:

Theorem (Parsons '68, Mints '73, Buss '95)

$I\Sigma_1$ proves the totality of precisely the primitive recursive functions.

Parsons' proof.

- Via a Dialectica-style functional interpretation.
- **Extracted programs:** higher-order variant of primitive recursive functions.

□

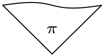
Buss' and Mints' proof.

- Via the witness function method.
- **Extracted programs:** regular primitive recursive functions of ground type.

□

The idea

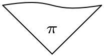
- A formal witness predicate over $\mathbb{N}$ for each 'tame' formula.
- Arithmetic proofs $\rightsquigarrow$ functions from witnesses to witnesses:



$$\Gamma \vdash \Delta \quad \rightsquigarrow \quad f^{\pi} : \left\{ \begin{array}{c} \text{witnesses} \\ \text{of } \bigwedge \Gamma \end{array} \right\} \to \left\{ \begin{array}{c} \text{witnesses} \\ \text{of } \bigvee \Delta \end{array} \right\}$$

The idea

- A formal witness predicate over $\mathbb{N}$ for each 'tame' formula.
- Arithmetic proofs $\rightsquigarrow$ functions from witnesses to witnesses:



$$\Gamma \vdash \Delta \quad \rightsquigarrow \quad f^{\pi} : \left\{ \begin{array}{c} \text{witnesses} \\ \text{of } \bigwedge \Gamma \end{array} \right\} \to \left\{ \begin{array}{c} \text{witnesses} \\ \text{of } \bigvee \Delta \end{array} \right\}$$

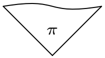
Crucial points

- π free-cut free: tames the complexity of formulae; no bad $\forall$.

The witness function method (WFM)

The idea

- A formal witness predicate over $\mathbb{N}$ for each 'tame' formula.
- Arithmetic proofs $\rightsquigarrow$ functions from witnesses to witnesses:



$\Gamma \vdash \Delta$

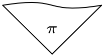
$$\rightsquigarrow \quad f^\pi : \left\{ \begin{array}{c} \text{witnesses} \\ \text{of } \bigwedge \Gamma \end{array} \right\} \to \left\{ \begin{array}{c} \text{witnesses} \\ \text{of } \bigvee \Delta \end{array} \right\}$$

Crucial points

- π free-cut free: tames the complexity of formulae; no bad $\forall$.
- De Morgan normal form: only functions at ground type, i.e. $\mathbb{N}^k \to \mathbb{N}$.

The idea

- A formal witness predicate over $\mathbb{N}$ for each 'tame' formula.
- Arithmetic proofs $\rightsquigarrow$ functions from witnesses to witnesses:



$$\Gamma \vdash \Delta \quad \rightsquigarrow \quad f^{\pi} : \left\{ \begin{array}{c} \text{witnesses} \\ \text{of } \bigwedge \Gamma \end{array} \right\} \to \left\{ \begin{array}{c} \text{witnesses} \\ \text{of } \bigvee \Delta \end{array} \right\}$$

Crucial points

- π free-cut free: tames the complexity of formulae; no bad $\forall$.
- De Morgan normal form: only functions at ground type, i.e. $\mathbb{N}^k \to \mathbb{N}$.
- Right-contraction: tests the witness predicate (should be decidable).

Free-cut elimination

- Used in various forms by Gentzen, Parikh, Paris & Wilkie, Cook, Krajícek,...
- First presented for general fragments of *PA* by Takeuti.
- Further generalised by Buss and others.

Free-cut elimination

- Used in various forms by Gentzen, Parikh, Paris & Wilkie, Cook, Krajícek,...
- First presented for general fragments of *PA* by Takeuti.
- Further generalised by Buss and others.

Witness function method

- Due to Buss and Mints.
- $\rightsquigarrow$ bounded arithmetic. Theories for $\mathbf{NC}_i, \mathbf{AC}_i, \mathbf{P}, \mathbf{PH}$,...
- The best method available to delineate hierarchies of classical theories.

Context and motivation

Free-cut elimination

- Used in various forms by Gentzen, Parikh, Paris & Wilkie, Cook, Krajícek,...
- First presented for general fragments of *PA* by Takeuti.
- Further generalised by Buss and others.

Witness function method

- Due to Buss and Mints.
- $\rightsquigarrow$ bounded arithmetic. Theories for $\mathbf{NC}_i, \mathbf{AC}_i, \mathbf{P}, \mathbf{PH}$,...
- The best method available to delineate hierarchies of classical theories.

Question

Can WFM be useful for characterising complexity classes via linear logic?

Outline

Linear logic (LL)

- LL is a substructural logic:

$$A \parr A \nvdash A \qquad A \nvdash A \parr B$$

Linear logic (LL)

- LL is a substructural logic:

$$A \parr A \nvdash A \qquad A \nvdash A \parr B$$

- It distinguishes multiplicative and additive rules by separate connectives:

$$\frac{\Gamma, A \quad \Delta, B}{\Gamma, \Delta, A \otimes B} \qquad \frac{\Gamma, A \quad \Gamma, B}{\Gamma, A \& B}$$

Linear logic (LL)

- LL is a substructural logic:

$$A \parr A \nvdash A \qquad A \nvdash A \parr B$$

- It distinguishes multiplicative and additive rules by separate connectives:

$$\frac{\Gamma, A \quad \Delta, B}{\Gamma, \Delta, A \otimes B} \qquad \frac{\Gamma, A \quad \Gamma, B}{\Gamma, A \& B}$$

- Controlled access to structural rules via modalities:

$$!A \ \vdash\ !A \otimes !A$$

Linear logic (LL)

- LL is a substructural logic:

$$A \parr A \nvdash A \qquad A \nvdash A \parr B$$

- It distinguishes multiplicative and additive rules by separate connectives:

$$\frac{\Gamma, A \quad \Delta, B}{\Gamma, \Delta, A \otimes B} \qquad \frac{\Gamma, A \quad \Gamma, B}{\Gamma, A \& B}$$

- Controlled access to structural rules via modalities:

$$!A \vdash !A \otimes !A$$

(otherwise ! behaves just like $\Box$ in *S4*)

Linear logic (LL)

- LL is a substructural logic:

$$A \parr A \nvdash A \qquad A \nvdash A \parr B$$

- It distinguishes multiplicative and additive rules by separate connectives:

$$\frac{\Gamma, A \quad \Delta, B}{\Gamma, \Delta, A \otimes B} \qquad \frac{\Gamma, A \quad \Gamma, B}{\Gamma, A \& B}$$

- Controlled access to structural rules via modalities:

$$!A \ \vdash \ !A \otimes !A$$

(otherwise ! behaves just like $\Box$ in *S4*)

- De Morgan duality is everywhere!

Free-cut elimination in linear logic

A nonlogical rule has the following format:

$$\frac{\{!\Gamma, \Sigma_i \vdash \Delta_i, ?\Pi\}_{i \in \mathcal{I}}}{!\Gamma, \Sigma \vdash \Delta, ?\Pi}$$

The formulae in Σ and Δ are considered principal.

Free-cut elimination in linear logic

A nonlogical rule has the following format:

$$\frac{\{!\Gamma, \Sigma_i \vdash \Delta_i, ?\Pi\}_{i \in \mathcal{I}}}{!\Gamma, \Sigma \vdash \Delta, ?\Pi}$$

The formulae in Σ and Δ are considered principal.

A cut step is anchored if:

- its cut-formulae are (almost) principal on both sides.
- on at least one side it is (almost) principal for a nonlogical step.

Free-cut elimination in linear logic

A nonlogical rule has the following format:

$$\frac{\{!\Gamma, \Sigma_i \vdash \Delta_i, ?\Pi\}_{i \in \mathcal{I}}}{!\Gamma, \Sigma \vdash \Delta, ?\Pi}$$

The formulae in Σ and Δ are considered principal.

A cut step is anchored if:

- its cut-formulae are (almost) principal on both sides.
- on at least one side it is (almost) principal for a nonlogical step.

Theorem

Any linear logic proof can be transformed into one where all cuts are anchored.

A nonlogical rule has the following format:

$$\frac{\{!\Gamma, \Sigma_i \vdash \Delta_i, ?\Pi\}_{i\in\mathcal{I}}}{!\Gamma, \Sigma \vdash \Delta, ?\Pi}$$

The formulae in Σ and Δ are considered principal.

A cut step is anchored if:

- its cut-formulae are (almost) principal on both sides.
- on at least one side it is (almost) principal for a nonlogical step.

Theorem

Any linear logic proof can be transformed into one where all cuts are anchored.

- Proof similar to usual cut-elimination arguments.
- Special cases due to Lincoln et al., Baelde & Miller,...

A nonlogical rule has the following format:

$$\frac{\{!\Gamma, \Sigma_i \vdash \Delta_i, ?\Pi\}_{i \in \mathcal{I}}}{!\Gamma, \Sigma \vdash \Delta, ?\Pi}$$

The formulae in Σ and Δ are considered principal.

A cut step is anchored if:

- its cut-formulae are (almost) principal on both sides.
- on at least one side it is (almost) principal for a nonlogical step.

Theorem

Any linear logic proof can be transformed into one where all cuts are anchored.

- Proof similar to usual cut-elimination arguments.
- Special cases due to Lincoln et al., Baelde & Miller,...

Corollary

Every theorem has a proof where all formulae are subformulae of the conclusion or a principal formula of a nonlogical step.

Outline

We consider an axiomatisation inspired by Bellantoni & Hofmann:

$$
\begin{array}{rl}
N_{cntr} & \forall x^N.(N(x) \otimes N(x)) \\
\\
N_\varepsilon & N(\varepsilon) \\
N_0 & \forall x^N.N(s_0 x) \\
N_1 & \forall x^N.N(s_1 x) \\
\varepsilon & \forall x^N.(\varepsilon \neq s_0 x \otimes \varepsilon \neq s_1 x) \\
inj_0 & \forall x^N, y^N.(s_0 x = s_0 y \multimap x = y) \\
inj_1 & \forall x^N, y^N.(s_1 x = s_1 y \multimap x = y) \\
tree & \forall x^N.s_0 x \neq s_1 x \\
surj & \forall x^N.(x = \varepsilon \oplus \exists y^N.x = s_0 y \oplus \exists y^N.x = s_1 y)
\end{array}
$$

$$
PIND \quad \begin{array}{rl}
 & A(\varepsilon) \\
\multimap & !(\forall x^{!N}.(A(x) \multimap A(s_0 x))) \\
\multimap & !(\forall x^{!N}.(A(x) \multimap A(s_1 x))) \\
\multimap & \forall x^{!N}.A(x)
\end{array}
$$

Peano's N predicate: $N(t) :=$ "t is a natural number".

Functions are specified by equational programs. E.g.:

$$\Phi \begin{cases} \texttt{add}(0, x) & = & x \\ \texttt{add}(su, x) & = & s(\texttt{add}(u, y)) \\ & & \\ \texttt{mult}(0, x) & = & 0 \\ \texttt{mult}(su, x) & = & \texttt{add}(x, \texttt{mult}(u, x)) \end{cases}$$

Convergence

Functions are specified by equational programs. E.g.:

$$\Phi \begin{cases} \texttt{add}(0, x) &= x \\ \texttt{add}(su, x) &= s(\texttt{add}(u, y)) \\ \\ \texttt{mult}(0, x) &= 0 \\ \texttt{mult}(su, x) &= \texttt{add}(x, \texttt{mult}(u, x)) \end{cases}$$

Convergence statement:

$$\forall \mathbf{x}^{!N}.\Phi(\mathbf{x}) \multimap \forall x^N, y^N.N(\texttt{mult}(x, y))$$

Functions are specified by equational programs. E.g.:

$$\Phi \left\{ \begin{array}{rcl} \mathtt{add}(0,x) & = & x \\ \mathtt{add}(su,x) & = & s(\mathtt{add}(u,y)) \\ & & \\ \mathtt{mult}(0,x) & = & 0 \\ \mathtt{mult}(su,x) & = & \mathtt{add}(x,\mathtt{mult}(u,x)) \end{array} \right.$$

Convergence statement:

$$\forall \mathbf{x}^{!N}.\Phi(\mathbf{x}) \multimap \forall x^N, y^N.N(\mathtt{mult}(x,y))$$

We will consider the theory $I\Sigma_1^{N^+}$, admitting *PIND* only over:

$$E \quad ::= \quad N(t) \mid s = t \mid s \neq t \mid E \parr E \mid E \otimes E \mid \exists x.E$$

Outline

Introduction

Normal forms in first-order linear logic

An arithmetic in linear logic

Bellantoni-Cook programs and the WFM for $I\Sigma_1^{N^+}$

Conclusions

Bellantoni-Cook characterisation of polytime functions

Arguments of a function are separated into normal and safe inputs:

$$f(\mathbf{u}; \mathbf{x})$$

Normal: left of ; so $\mathbf{u}$ above. Safe: right of ; so $\mathbf{x}$ above.

Bellantoni-Cook characterisation of polytime functions

Arguments of a function are separated into normal and safe inputs:

$$f(\mathbf{u}; \mathbf{x})$$

Normal: left of ; so **u** above. Safe: right of ; so **x** above.

Predicative recursion on notation
If g, h_0, h_1 are BC then so is f defined by:

$$\begin{aligned} f(\varepsilon, \mathbf{v}; \mathbf{x}) &= g(\mathbf{v}; \mathbf{x}) \\ f(s_0 u, \mathbf{v}; \mathbf{x}) &= h_0(u, \mathbf{v}; \mathbf{x}, f(u, \mathbf{v}; \mathbf{x})) \\ f(s_1 u, \mathbf{v}; \mathbf{x}) &= h_1(u, \mathbf{v}; \mathbf{x}, f(u, \mathbf{v}; \mathbf{x})) \end{aligned}$$

Arguments of a function are separated into normal and safe inputs:

$$f(\mathbf{u}; \mathbf{x})$$

Normal: left of ; so $\mathbf{u}$ above. Safe: right of ; so $\mathbf{x}$ above.

Predicative recursion on notation
If g, h_0, h_1 are BC then so is f defined by:

$$\begin{array}{rcl} f(\varepsilon, \mathbf{v}; \mathbf{x}) & = & g(\mathbf{v}; \mathbf{x}) \\ f(s_0 u, \mathbf{v}; \mathbf{x}) & = & h_0(u, \mathbf{v}; \mathbf{x}, f(u, \mathbf{v}; \mathbf{x})) \\ f(s_1 u, \mathbf{v}; \mathbf{x}) & = & h_1(u, \mathbf{v}; \mathbf{x}, f(u, \mathbf{v}; \mathbf{x})) \end{array}$$

Safe composition
Can compose functions as long as safe inputs are hereditarily safe.

Arguments of a function are separated into normal and safe inputs:

$$f(\mathbf{u}; \mathbf{x})$$

Normal: left of ; so $\mathbf{u}$ above. Safe: right of ; so $\mathbf{x}$ above.

Predicative recursion on notation
If g, h_0, h_1 are BC then so is f defined by:

$$\begin{aligned} f(\varepsilon, \mathbf{v}; \mathbf{x}) &= g(\mathbf{v}; \mathbf{x}) \\ f(s_0 u, \mathbf{v}; \mathbf{x}) &= h_0(u, \mathbf{v}; \mathbf{x}, f(u, \mathbf{v}; \mathbf{x})) \\ f(s_1 u, \mathbf{v}; \mathbf{x}) &= h_1(u, \mathbf{v}; \mathbf{x}, f(u, \mathbf{v}; \mathbf{x})) \end{aligned}$$

Safe composition
Can compose functions as long as safe inputs are hereditarily safe.

(Also an adequate stock of initial functions.)

Bellantoni-Cook characterisation of polytime functions

Arguments of a function are separated into normal and safe inputs:

$$f(\mathbf{u};\mathbf{x})$$

Normal: left of ; so $\mathbf{u}$ above. Safe: right of ; so $\mathbf{x}$ above.

Predicative recursion on notation
If g, h_0, h_1 are BC then so is f defined by:

$$\begin{array}{rcl} f(\varepsilon,\mathbf{v};\mathbf{x}) & = & g(\mathbf{v};\mathbf{x}) \\ f(s_0 u,\mathbf{v};\mathbf{x}) & = & h_0(u,\mathbf{v};\mathbf{x}, f(u,\mathbf{v};\mathbf{x})) \\ f(s_1 u,\mathbf{v};\mathbf{x}) & = & h_1(u,\mathbf{v};\mathbf{x}, f(u,\mathbf{v};\mathbf{x})) \end{array}$$

Safe composition
Can compose functions as long as safe inputs are hereditarily safe.

(Also an adequate stock of initial functions.)

Theorem (Bellantoni & Cook '92)

BC programs compute just the polynomial-time functions.

Multiplication again...

$$
\begin{aligned}
\texttt{add}(0; x) &= x \\
\texttt{add}(su; x) &= s(\texttt{add}(u; y))
\end{aligned}
$$

$$
\begin{aligned}
\texttt{mult}(0, x;) &= 0 \\
\texttt{mult}(su, x;) &= \texttt{add}(x; \texttt{mult}(u, x))
\end{aligned}
$$

Main results

Main results

Theorem
Every BC program is provably convergent in $I\Sigma_1^{N^+}$.

Main results

Theorem

Every BC program is provably convergent in $I\Sigma_1^{N^+}$.

Proof.

Straightforward. Similar to previous arguments by Leivant, Bellantoni & Hofmann, Cantini, taking care to respect linear considerations. □

Main results

Theorem

Every BC program is provably convergent in $I\Sigma_1^{N^+}$.

Proof.

Straightforward. Similar to previous arguments by Leivant, Bellantoni & Hofmann, Cantini, taking care to respect linear considerations. □

Theorem

Every function provably convergent in $I\Sigma_1^{N^+}$ is polynomial-time computable.

Main results

Theorem

Every BC program is provably convergent in $I\Sigma_1^{N^+}$.

Proof.

Straightforward. Similar to previous arguments by Leivant, Bellantoni & Hofmann, Cantini, taking care to respect linear considerations. □

Theorem

Every function provably convergent in $I\Sigma_1^{N^+}$ is polynomial-time computable.

Main proof intuitions.

Via the WFM.

- **Free-cut elimination:**
  - No $\forall$-formulae $\rightsquigarrow$ witness predicates of ground type, no $\forall$-right.
  - No ?-formulae $\rightsquigarrow$ no contraction-right.

Main results

Theorem

Every BC program is provably convergent in $I\Sigma_1^{N^+}$.

Proof.

Straightforward. Similar to previous arguments by Leivant, Bellantoni & Hofmann, Cantini, taking care to respect linear considerations. □

Theorem

Every function provably convergent in $I\Sigma_1^{N^+}$ is polynomial-time computable.

Main proof intuitions.

Via the WFM.

- **Free-cut elimination:**
  - No $\forall$-formulae $\rightsquigarrow$ witness predicates of ground type, no $\forall$-right.
  - No ?-formulae $\rightsquigarrow$ no contraction-right.
- !-**formulae:** normal inputs for the witness functions.

Main results

Theorem

Every BC program is provably convergent in $I\Sigma_1^{N^+}$.

Proof.

Straightforward. Similar to previous arguments by Leivant, Bellantoni & Hofmann, Cantini, taking care to respect linear considerations. □

Theorem

Every function provably convergent in $I\Sigma_1^{N^+}$ is polynomial-time computable.

Main proof intuitions.

Via the WFM.

- **Free-cut elimination:**
  - No $\forall$-formulae $\rightsquigarrow$ witness predicates of ground type, no $\forall$-right.
  - No ?-formulae $\rightsquigarrow$ no contraction-right.
- **!-formulae:** normal inputs for the witness functions.
- **!, ?-free** *PIND***:** predicative recursion.

Main results

Theorem
Every BC program is provably convergent in $I\Sigma_1^{N^+}$.

Proof.
Straightforward. Similar to previous arguments by Leivant, Bellantoni & Hofmann, Cantini, taking care to respect linear considerations. □

Theorem
Every function provably convergent in $I\Sigma_1^{N^+}$ is polynomial-time computable.

Main proof intuitions.

Via the WFM.

- **Free-cut elimination:**
  - No $\forall$-formulae $\leadsto$ witness predicates of ground type, no $\forall$-right.
  - No ?-formulae $\leadsto$ no contraction-right.
- **!-formulae:** normal inputs for the witness functions.
- **!, ?-free** *PIND*: predicative recursion.
- **Anchored cuts:** safe composition of functions.

□

Example case: induction

Example case: induction

- By deduction and invertibility, we can assume *PIND* occurs as:

$$\frac{!\Gamma \vdash A(\varepsilon), ?\Delta \quad !N(a), !\Gamma, A(a) \vdash A(s_0 a), ?\Delta \quad !N(a), !\Gamma, A(a) \vdash A(s_0 a), ?\Delta}{!N(t), !\Gamma \vdash A(t), ?\Delta}$$

Example case: induction

- By deduction and invertibility, we can assume *PIND* occurs as:

$$\frac{!\Gamma \vdash A(\varepsilon), ?\Delta \quad !N(a), !\Gamma, A(a) \vdash A(s_0 a), ?\Delta \quad !N(a), !\Gamma, A(a) \vdash A(s_0 a), ?\Delta}{!N(t), !\Gamma \vdash A(t), ?\Delta}$$

- By free-cut elimination we can assume Δ is empty.

Example case: induction

- By deduction and invertibility, we can assume *PIND* occurs as:

$$\frac{!\Gamma \vdash A(\varepsilon), ?\Delta \quad !N(a), !\Gamma, A(a) \vdash A(s_0 a), ?\Delta \quad !N(a), !\Gamma, A(a) \vdash A(s_0 a), ?\Delta}{!N(t), !\Gamma \vdash A(t), ?\Delta}$$

- By free-cut elimination we can assume Δ is empty.
- By inductive hypothesis suppose we have functions:

$$g\left(u^{\Gamma};\right)$$
$$h_i\left(u^{N(a)}, u^{\Gamma}; x^{A(a)}\right)$$

Example case: induction

- By deduction and invertibility, we can assume $PIND$ occurs as:

$$\frac{!\Gamma \vdash A(\varepsilon), ?\Delta \quad !N(a), !\Gamma, A(a) \vdash A(s_0 a), ?\Delta \quad !N(a), !\Gamma, A(a) \vdash A(s_0 a), ?\Delta}{!N(t), !\Gamma \vdash A(t), ?\Delta}$$

- By free-cut elimination we can assume Δ is empty.
- By inductive hypothesis suppose we have functions:

$$g\left(u^{\Gamma};\right)$$
$$h_i\left(u^{N(a)}, u^{\Gamma}; x^{A(a)}\right)$$

- Define f by PRN:

$$\begin{aligned} f\left(0, u^{\Gamma};\right) &:= g\left(u^{\Gamma};\right) \\ f\left(s_i u^{N(t)}, u^{\Gamma};\right) &:= h_i\left(u^{N(t)}, u^{\Gamma}; f\left(u^{N(t)}, u^{\Gamma};\right)\right) \end{aligned}$$

Outline

Introduction

Normal forms in first-order linear logic

An arithmetic in linear logic

Bellantoni-Cook programs and the WFM for $I\Sigma_1^{N^+}$

Conclusions

Conclusions

Summary

- General form of free-cut elimination for first-order linear logic.
- Induces useful normal forms for arithmetic proofs.
- Soundness and completeness of an arithmetic for BC-programs.

Conclusions

Summary

- General form of free-cut elimination for first-order linear logic.
- Induces useful normal forms for arithmetic proofs.
- Soundness and completeness of an arithmetic for BC-programs.

Further work

- Bounded arithmetic style approach.
  - Finer use of the witness predicate: evaluation in polynomial-time.
  - Relationships to BC-versions of equational theory PV?

Conclusions

Summary

- General form of free-cut elimination for first-order linear logic.
- Induces useful normal forms for arithmetic proofs.
- Soundness and completeness of an arithmetic for BC-programs.

Further work

- Bounded arithmetic style approach.
  - Finer use of the witness predicate: evaluation in polynomial-time.
  - Relationships to BC-versions of equational theory PV?
- Characterise polynomial hierarchy via minimisation principles.
  - Functions conditional on Σ^p_i tests.
  - Relies on evaluation of witness predicate in Δ^p_i.

Thank you.