Transcendental Syntax

A toolbox for the interface logic-computation

LIPN – Université Sorbonne Paris Nord

Boris Eng

Realisability theory

Realisability/logical relations. [Riba, LICS 2007].

Realisability theory

Realisability/logical relations. [Riba, LICS 2007].

- programs : pure λ-calculus $\qquad t, u ::= x \mid \lambda x.t \mid tu.$

Realisability theory

Realisability/logical relations. [Riba, LICS 2007].

- programs : pure λ-calculus $\qquad t, u ::= x \mid \lambda x.t \mid tu.$
- types : simple types as set of terms $t : \mathbf{A} \iff t \in \mathbf{A}.$

Realisability theory

Realisability/logical relations. [Riba, LICS 2007].

- programs : pure λ-calculus $\qquad t, u ::= x \mid \lambda x.t \mid tu.$
- types : simple types as set of terms $t : \mathbf{A} \iff t \in \mathbf{A}.$
  - base type $\mathbf{o} := \mathcal{SN}$ (set of terminating programs).

Realisability theory

Realisability/logical relations. [Riba, LICS 2007].

- programs : pure λ-calculus $\quad t, u ::= x \mid \lambda x.t \mid tu.$
- types : simple types as set of terms $t : \mathbf{A} \iff t \in \mathbf{A}$.
  - base type $\mathbf{o} := \mathcal{SN}$ (set of terminating programs).
  - $\mathbf{A} \Rightarrow \mathbf{B} = \{t \mid \forall u \in \mathbf{A}, tu \in \mathbf{B})\}$

From Geometry of Interaction to Transcendental Syntax

Geometry of Interaction and Ludics (Girard). Reconstruction of logic from computation.

From Geometry of Interaction to Transcendental Syntax

Geometry of Interaction and Ludics (Girard). Reconstruction of logic from computation.

- programs : some mathematical representation of proofs(-nets).

From Geometry of Interaction to Transcendental Syntax

Geometry of Interaction and Ludics (Girard). Reconstruction of logic from computation.

- programs : some mathematical representation of proofs(-nets).
- types : formulas of linear logic.

From Geometry of Interaction to Transcendental Syntax

Geometry of Interaction and Ludics (Girard). Reconstruction of logic from computation.

- programs : some mathematical representation of proofs(-nets).
- types : formulas of linear logic.
  - $\mathbf{A} \Rightarrow \mathbf{B} := \{\pi \mid \forall \pi' \in \mathbf{A}, \mathtt{cut}(\pi, \pi') \in \mathbf{B})\}$ (linear implication).

From Geometry of Interaction to Transcendental Syntax

Geometry of Interaction and Ludics (Girard). Reconstruction of logic from computation.

- programs : some mathematical representation of proofs(-nets).
- types : formulas of linear logic.
  - $\mathbf{A} \Rightarrow \mathbf{B} := \{\pi \mid \forall \pi' \in \mathbf{A}, \mathtt{cut}(\pi, \pi') \in \mathbf{B})\}$ (linear implication).
  - $\pi \perp \pi' \iff \mathtt{cut}(\pi, \pi')$ satisfies some P.

From Geometry of Interaction to Transcendental Syntax

Geometry of Interaction and Ludics (Girard). Reconstruction of logic from computation.

- programs : some mathematical representation of proofs(-nets).
- types : formulas of linear logic.
  - $\mathbf{A} \Rightarrow \mathbf{B} := \{\pi \mid \forall \pi' \in \mathbf{A}, \mathtt{cut}(\pi, \pi') \in \mathbf{B})\}$ (linear implication).
  - $\pi \perp \pi' \iff \mathtt{cut}(\pi, \pi')$ satisfies some P.
  - $\mathbf{A}^{\perp} := \{\pi \mid \forall \pi' \in \mathbf{A}, \pi \perp \pi'\}$ (linear negation).

From Geometry of Interaction to Transcendental Syntax

Geometry of Interaction and Ludics (Girard). Reconstruction of logic from computation.

- programs : some mathematical representation of proofs(-nets).
- types : formulas of linear logic.
  - $\mathbf{A} \Rightarrow \mathbf{B} := \{\pi \mid \forall \pi' \in \mathbf{A}, \mathtt{cut}(\pi, \pi') \in \mathbf{B})\}$ (linear implication).
  - $\pi \perp \pi' \iff \mathtt{cut}(\pi, \pi')$ satisfies some P.
  - $\mathbf{A}^{\perp} := \{\pi \mid \forall \pi' \in \mathbf{A}, \pi \perp \pi'\}$ (linear negation).
  - Linear logic formulas satisfy $\mathbf{A} = \mathbf{A}^{\perp\perp}$.

From Geometry of Interaction to Transcendental Syntax

Geometry of Interaction and Ludics (Girard). Reconstruction of logic from computation.

- programs : some mathematical representation of proofs(-nets).
- types : formulas of linear logic.
  - $\mathbf{A} \Rightarrow \mathbf{B} := \{\pi \mid \forall \pi' \in \mathbf{A}, \mathtt{cut}(\pi, \pi') \in \mathbf{B})\}$ (linear implication).
  - $\pi \perp \pi' \iff \mathtt{cut}(\pi, \pi')$ satisfies some P.
  - $\mathbf{A}^{\perp} := \{\pi \mid \forall \pi' \in \mathbf{A}, \pi \perp \pi'\}$ (linear negation).
  - Linear logic formulas satisfy $\mathbf{A} = \mathbf{A}^{\perp\perp}$.

Transcendental Syntax (Girard, 2013). Improvements on GoI.

From Geometry of Interaction to Transcendental Syntax

Geometry of Interaction and Ludics (Girard). Reconstruction of logic from computation.

- programs : some mathematical representation of proofs(-nets).
- types : formulas of linear logic.
  - $\mathbf{A} \Rightarrow \mathbf{B} := \{\pi \mid \forall \pi' \in \mathbf{A}, \mathtt{cut}(\pi, \pi') \in \mathbf{B})\}$ (linear implication).
  - $\pi \perp \pi' \iff \mathtt{cut}(\pi, \pi')$ satisfies some P.
  - $\mathbf{A}^{\perp} := \{\pi \mid \forall \pi' \in \mathbf{A}, \pi \perp \pi'\}$ (linear negation).
  - Linear logic formulas satisfy $\mathbf{A} = \mathbf{A}^{\perp\perp}$.

Transcendental Syntax (Girard, 2013). Improvements on GoI.

- programs : "Stellar Resolution" (Turing-complete).

From Geometry of Interaction to Transcendental Syntax

Geometry of Interaction and Ludics (Girard). Reconstruction of logic from computation.

- programs : some mathematical representation of proofs(-nets).
- types : formulas of linear logic.
  - $\mathbf{A} \Rightarrow \mathbf{B} := \{\pi \mid \forall \pi' \in \mathbf{A}, \mathtt{cut}(\pi, \pi') \in \mathbf{B})\}$ (linear implication).
  - $\pi \perp \pi' \iff \mathtt{cut}(\pi, \pi')$ satisfies some P.
  - $\mathbf{A}^{\perp} := \{\pi \mid \forall \pi' \in \mathbf{A}, \pi \perp \pi'\}$ (linear negation).
  - Linear logic formulas satisfy $\mathbf{A} = \mathbf{A}^{\perp\perp}$.

Transcendental Syntax (Girard, 2013). Improvements on GoI.

- programs : "Stellar Resolution" (Turing-complete).
- types : formulas of linear logic and more.

From Geometry of Interaction to Transcendental Syntax

Geometry of Interaction and Ludics (Girard). Reconstruction of logic from computation.

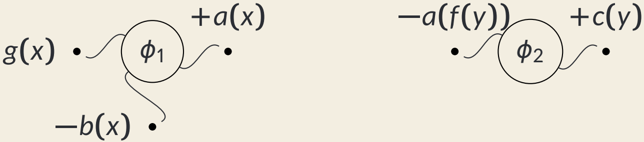
- programs : some mathematical representation of proofs(-nets).
- types : formulas of linear logic.
  - $\mathbf{A} \Rightarrow \mathbf{B} := \{\pi \mid \forall \pi' \in \mathbf{A}, \texttt{cut}(\pi, \pi') \in \mathbf{B})\}$ (linear implication).
  - $\pi \perp \pi' \iff \texttt{cut}(\pi, \pi')$ satisfies some P.
  - $\mathbf{A}^{\perp} := \{\pi \mid \forall \pi' \in \mathbf{A}, \pi \perp \pi'\}$ (linear negation).
  - Linear logic formulas satisfy $\mathbf{A} = \mathbf{A}^{\perp\perp}$.

Transcendental Syntax (Girard, 2013). Improvements on GoI.

- programs : "Stellar Resolution" (Turing-complete).
- types : formulas of linear logic and more.
- Speaks about the "logic" of a computational model.

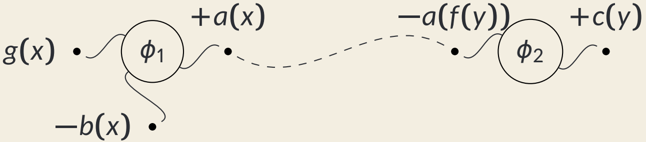
Stellar Resolution

Girard's stars and constellations



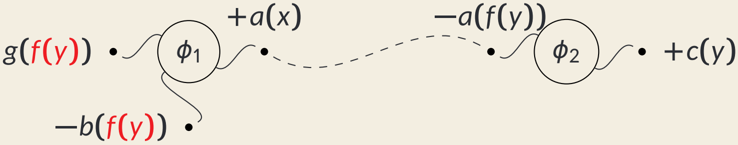
Stellar Resolution

Girard's stars and constellations



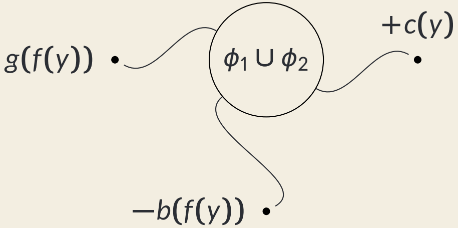
Stellar Resolution

Girard's stars and constellations



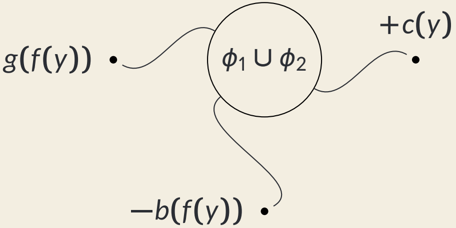
Stellar Resolution

Girard's stars and constellations



Stellar Resolution

Girard's stars and constellations



Constellation Φ (n stars)

= program

↓

Diagrams (maximal tilings)

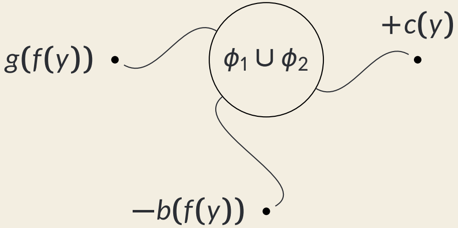
↓

Constellation $\text{Ex}(\Phi)$

= normal form

Stellar Resolution

Girard's stars and constellations



Constellation Φ (n stars)

= program

$\downarrow$

Diagrams (maximal tilings)

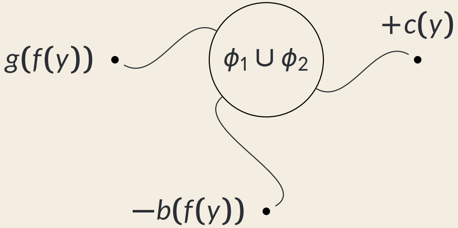
$\downarrow$

Constellation $\mathrm{Ex}(\Phi)$

= normal form

Stellar Resolution

Girard's stars and constellations



Constellation Φ (n stars)

= program

$\downarrow$

Diagrams (maximal tilings)

$\downarrow$

Constellation $\text{Ex}(\Phi)$

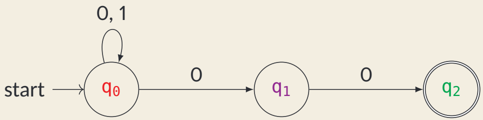
= normal form

A reformulation of Robinson's first-order resolution / Query-free logic programming.

Stellar Resolution

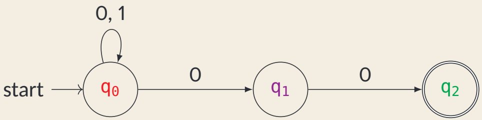
Automata and circuits unified

Generalised automata.



Stellar Resolution

Automata and circuits unified



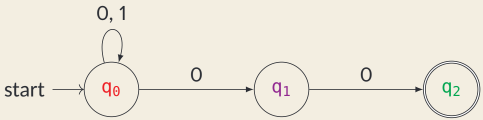
Generalised automata.

Transitions $\longleftrightarrow$ binary stars $[-a(c \cdot w, q), +a(w, q')]$.

Run on a word $\longleftrightarrow$ tiling/diagram.

Stellar Resolution

Automata and circuits unified

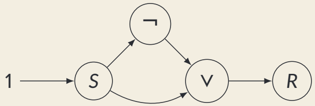


Generalised automata.

Transitions $\longleftrightarrow$ binary stars $[-a(c \cdot w, q), +a(w, q')]$.

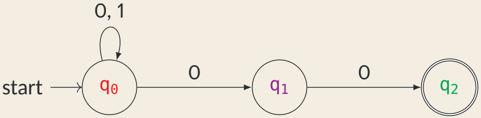
Run on a word $\longleftrightarrow$ tiling/diagram.

Generalised circuits.



Stellar Resolution

Automata and circuits unified

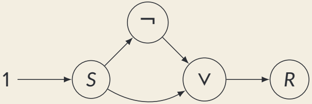


Generalised automata.

Transitions $\longleftrightarrow$ binary stars $[-a(c \cdot w, q), +a(w, q')]$.

Run on a word $\longleftrightarrow$ tiling/diagram.

Generalised circuits.

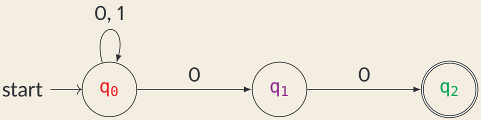


Gates (not) $\longleftrightarrow$ star $[-c_i(x), -not(x, r), +c_j(r)]$.

Circuit evaluation $\longleftrightarrow$ execution of constellation.

Stellar Resolution

Automata and circuits unified

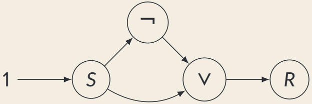


Generalised automata.

Transitions $\longleftrightarrow$ binary stars $[-a(c \cdot w, q), +a(w, q')]$.

Run on a word $\longleftrightarrow$ tiling/diagram.

Generalised circuits.



Gates (not) $\longleftrightarrow$ star $[-c_i(x), -not(x, r), +c_j(r)]$.

Circuit evaluation $\longleftrightarrow$ execution of constellation.

Information flow inside a structure : pushdown/tree/alternating automata, Turing machines, tile systems, ...

Realisability and interactive typing

We have a new model of computation. What can we do ?

Realisability and interactive typing

We have a new model of computation. What can we do ?

Reconstructing linear logic (Transcendental Syntax).

Realisability and interactive typing

We have a new model of computation. What can we do ?

Reconstructing linear logic (Transcendental Syntax).

- Pre-types **A** a set of constellations (programs).

Realisability and interactive typing

We have a new model of computation. What can we do ?

Reconstructing linear logic (Transcendental Syntax).

- Pre-types **A** a set of constellations (programs).
- Choose a binary orthogonality $\perp$ for "correct interaction".

Realisability and interactive typing

We have a new model of computation. What can we do ?

Reconstructing linear logic (Transcendental Syntax).

- Pre-types $\mathbf{A}$ a set of constellations (programs).
- Choose a binary orthogonality $\perp$ for "correct interaction".
- Define $\mathbf{A}^{\perp} = \{\Phi \mid \forall \Phi' \in \mathbf{A}, \Phi \perp \Phi'\}$ (linear negation / duality).

Realisability and interactive typing

We have a new model of computation. What can we do ?

Reconstructing linear logic (Transcendental Syntax).

- Pre-types $\mathbf{A}$ a set of constellations (programs).
- Choose a binary orthogonality $\perp$ for "correct interaction".
- Define $\mathbf{A}^{\perp} = \{\Phi \mid \forall \Phi' \in \mathbf{A}, \Phi \perp \Phi'\}$ (linear negation / duality).
- Formulas/types : $\mathbf{A}$ such that $\mathbf{A} = \mathbf{A}^{\perp\perp}$.

Realisability and interactive typing

We have a new model of computation. What can we do ?

Reconstructing linear logic (Transcendental Syntax).

- Pre-types $\mathbf{A}$ a set of constellations (programs).
- Choose a binary orthogonality $\perp$ for "correct interaction".
- Define $\mathbf{A}^{\perp} = \{\Phi \mid \forall \Phi' \in \mathbf{A}, \Phi \perp \Phi'\}$ (linear negation / duality).
- Formulas/types : $\mathbf{A}$ such that $\mathbf{A} = \mathbf{A}^{\perp\perp}$.
- Assembling types : $\mathbf{A} \otimes \mathbf{B} = \{\Phi_A \uplus \Phi_B \mid \Phi_A \in \mathbf{A}, \Phi_B \in \mathbf{B}\}^{\perp\perp}$.

Realisability and interactive typing

We have a new model of computation. What can we do ?

Reconstructing linear logic (Transcendental Syntax).

- Pre-types $\mathbf{A}$ a set of constellations (programs).
- Choose a binary orthogonality $\perp$ for "correct interaction".
- Define $\mathbf{A}^{\perp} = \{\Phi \mid \forall \Phi' \in \mathbf{A}, \Phi \perp \Phi'\}$ (linear negation / duality).
- Formulas/types : $\mathbf{A}$ such that $\mathbf{A} = \mathbf{A}^{\perp\perp}$.
- Assembling types : $\mathbf{A} \otimes \mathbf{B} = \{\Phi_A \uplus \Phi_B \mid \Phi_A \in \mathbf{A}, \Phi_B \in \mathbf{B}\}^{\perp\perp}$.
- Deriving other connectives : $\mathbf{A} \parr \mathbf{B} = (\mathbf{A}^{\perp} \otimes \mathbf{B}^{\perp})^{\perp}$ and $\mathbf{A} \multimap \mathbf{B} = \mathbf{A}^{\perp} \parr \mathbf{B}$.

Realisability and interactive typing

We have a new model of computation. What can we do ?

Reconstructing linear logic (Transcendental Syntax).

- Pre-types $\mathbf{A}$ a set of constellations (programs).
- Choose a binary orthogonality $\perp$ for "correct interaction".
- Define $\mathbf{A}^{\perp} = \{\Phi \mid \forall \Phi' \in \mathbf{A}, \Phi \perp \Phi'\}$ (linear negation / duality).
- Formulas/types : $\mathbf{A}$ such that $\mathbf{A} = \mathbf{A}^{\perp\perp}$.
- Assembling types : $\mathbf{A} \otimes \mathbf{B} = \{\Phi_A \uplus \Phi_B \mid \Phi_A \in \mathbf{A}, \Phi_B \in \mathbf{B}\}^{\perp\perp}$.
- Deriving other connectives : $\mathbf{A} \parr \mathbf{B} = (\mathbf{A}^{\perp} \otimes \mathbf{B}^{\perp})^{\perp}$ and $\mathbf{A} \multimap \mathbf{B} = \mathbf{A}^{\perp} \parr \mathbf{B}$.

Various models of linear logic + a logical description of a model of computation.

Vague ideas of applications

(Unit) testing in logic

Generalising the correctness criterion

Transcendental Syntax. A constellation Φ is a proof of A when :

(Unit) testing in logic

Generalising the correctness criterion

Transcendental Syntax. A constellation Φ is a proof of A when :

(Danos-Regnier criterion) Φ_t^1 Φ_t^2 Φ_t^n

(Unit) testing in logic

Generalising the correctness criterion

Transcendental Syntax. A constellation Φ is a proof of A when :

(Tested constellation) Φ Φ Φ

(Danos-Regnier criterion) Φ_t^1 Φ_t^2 Φ_t^n

(Unit) testing in logic

Generalising the correctness criterion

Transcendental Syntax. A constellation Φ is a proof of A when :

$$\begin{array}{lcccc} \text{(Tested constellation)} & \Phi & \Phi & & \Phi \\ & \perp_{DR} & \perp_{DR} & \dots & \perp_{DR} \\ \text{(Danos-Regnier criterion)} & \Phi_t^1 & \Phi_t^2 & & \Phi_t^n \end{array}$$

(Unit) testing in logic

Generalising the correctness criterion

Transcendental Syntax. A constellation Φ is a proof of A when :

$$\begin{array}{lcccc} \text{(Tested constellation)} & \Phi & \Phi & & \Phi \\ & \perp_{DR} & \perp_{DR} & \dots & \perp_{DR} \\ \text{(Danos-Regnier criterion)} & \Phi_t^1 & \Phi_t^2 & & \Phi_t^n \end{array}$$

Unit testing and specifications.

(Unit) testing in logic

Generalising the correctness criterion

Transcendental Syntax. A constellation Φ is a proof of A when :

$$\begin{array}{lcccc} \text{(Tested constellation)} & \Phi & \Phi & & \Phi \\ & \perp_{DR} & \perp_{DR} & \cdots & \perp_{DR} \\ \text{(Danos-Regnier criterion)} & \Phi_t^1 & \Phi_t^2 & & \Phi_t^n \end{array}$$

Unit testing and specifications.

- Unit testing : a function f is "correct" when $f(a_i) = b_i$ for some (a_i, b_i).

(Unit) testing in logic

Generalising the correctness criterion

Transcendental Syntax. A constellation Φ is a proof of A when :

$$
\begin{array}{lcccc}
\text{(Tested constellation)} & \Phi & \Phi & & \Phi \\
 & \perp_{DR} & \perp_{DR} & \dots & \perp_{DR} \\
\text{(Danos-Regnier criterion)} & \Phi_t^1 & \Phi_t^2 & & \Phi_t^n
\end{array}
$$

Unit testing and specifications.

- Unit testing : a function f is "correct" when $f(a_i) = b_i$ for some (a_i, b_i).
- Specifications : a function f is labelled by A when it has some behaviour $BH(A)$.

(Unit) testing in logic

Generalising the correctness criterion

Transcendental Syntax. A constellation Φ is a proof of A when :

$$\begin{array}{lcccc} \text{(Tested constellation)} & \Phi & \Phi & & \Phi \\ & \perp_{DR} & \perp_{DR} & \cdots & \perp_{DR} \\ \text{(Danos-Regnier criterion)} & \Phi_t^1 & \Phi_t^2 & & \Phi_t^n \end{array}$$

Unit testing and specifications.

- Unit testing : a function f is "correct" when $f(a_i) = b_i$ for some (a_i, b_i).
- Specifications : a function f is labelled by A when it has some behaviour $BH(A)$.

Transcendental Syntax.

(Unit) testing in logic

Generalising the correctness criterion

Transcendental Syntax. A constellation Φ is a proof of A when :

$$\begin{array}{lcccc} \text{(Tested constellation)} & \Phi & \Phi & & \Phi \\ & \perp_{DR} & \perp_{DR} & \dots & \perp_{DR} \\ \text{(Danos-Regnier criterion)} & \Phi_t^1 & \Phi_t^2 & & \Phi_t^n \end{array}$$

Unit testing and specifications.

- Unit testing : a function f is "correct" when $f(a_i) = b_i$ for some (a_i, b_i).
- Specifications : a function f is labelled by A when it has some behaviour $BH(A)$.

Transcendental Syntax.

- A constellation Φ is correct w.r.t. A when it passes some tests in $\texttt{Tests}(A)$.

(Unit) testing in logic

Generalising the correctness criterion

Transcendental Syntax. A constellation Φ is a proof of A when :

$$\begin{array}{lcccc} \text{(Tested constellation)} & \Phi & \Phi & & \Phi \\ & \perp_{DR} & \perp_{DR} & \dots & \perp_{DR} \\ \text{(Danos-Regnier criterion)} & \Phi_t^1 & \Phi_t^2 & & \Phi_t^n \end{array}$$

Unit testing and specifications.

- Unit testing : a function f is "correct" when $f(a_i) = b_i$ for some (a_i, b_i).
- Specifications : a function f is labelled by A when it has some behaviour $BH(A)$.

Transcendental Syntax.

- A constellation Φ is correct w.r.t. A when it passes some tests in $\mathtt{Tests}(A)$.
- Adequation : Φ is correct w.r.t. $A \Longrightarrow \Phi \in BH(A)$ with $BH(A) = BH(A)^{\perp\perp}$.

Atypic typing and complexity

Typing outside λ-calculus. Automata, logic programs, circuits, tile systems, ...

Atypic typing and complexity

Typing outside λ-calculus. Automata, logic programs, circuits, tile systems, ...

↳ basically information flow in a structure.

Atypic typing and complexity

Typing outside λ-calculus. Automata, logic programs, circuits, tile systems, ...
↳ basically information flow in a structure.

Implicit Computational Complexity (ICC). Capture classes with restrictions on constellations.

Atypic typing and complexity

Typing outside λ-calculus. Automata, logic programs, circuits, tile systems, ...
↳ basically information flow in a structure.

Implicit Computational Complexity (ICC). Capture classes with restrictions on constellations.

- Previous works of Aubert & Bagnol.

Atypic typing and complexity

Typing outside λ-calculus. Automata, logic programs, circuits, tile systems, ...
↳ basically information flow in a structure.

Implicit Computational Complexity (ICC). Capture classes with restrictions on constellations.

- Previous works of Aubert & Bagnol.
  ↳ Capture of classes **P** and **(N)L** (with pointer machines).

Atypic typing and complexity

Typing outside λ-calculus. Automata, logic programs, circuits, tile systems, ...
↳ basically information flow in a structure.

Implicit Computational Complexity (ICC). Capture classes with restrictions on constellations.

- Previous works of Aubert & Bagnol.
  ↳ Capture of classes **P** and **(N)L** (with pointer machines).

Descriptive complexity. Capture classes with formulas.

Atypic typing and complexity

Typing outside λ-calculus. Automata, logic programs, circuits, tile systems, ...
↳ basically information flow in a structure.

Implicit Computational Complexity (ICC). Capture classes with restrictions on constellations.

- Previous works of Aubert & Bagnol.
  ↳ Capture of classes **P** and **(N)L** (with pointer machines).

Descriptive complexity. Capture classes with formulas.

- **P** and **NP** as classes of formulas (Immerman, Fagin).

Conclusion

A new model of computation : Stellar Resolution.

Conclusion

A new model of computation : Stellar Resolution.

↳ Turing-complete, generalised circuit-automata-logic programs.

Conclusion

A new model of computation : Stellar Resolution.

↳ Turing-complete, generalised circuit-automata-logic programs.

↳ Speaks about (unit) testing with orthogonality.

Conclusion

A new model of computation : Stellar Resolution.

↳ Turing-complete, generalised circuit-automata-logic programs.

↳ Speaks about (unit) testing with orthogonality.

↳ Speaks about the behaviour/specification of programs with realisability types.

Conclusion

A new model of computation : Stellar Resolution.

↳ Turing-complete, generalised circuit-automata-logic programs.

↳ Speaks about (unit) testing with orthogonality.

↳ Speaks about the behaviour/specification of programs with realisability types.

Thank you for listening to my talk.