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 \( t, u ::= x \mid \lambda x.t \mid tu. \)
Realisability theory

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

- **programs**: pure $\lambda$-calculus $t, u ::= x | \lambda x. t | tu$.
- **types**: simple types as set of terms $t : A \iff t \in A$. 
Realisability theory

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

- **Programs**: pure λ-calculus \( t, u ::= x | \lambda x. t | tu \).
- **Types**: simple types as set of terms \( t : A \iff t \in A \).
  - Base type \( o ::= SN \) (set of terminating programs).
Realisability theory

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

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

  – $A \Rightarrow B = \{ \pi | \forall \pi' \in A, \text{cut}(\pi, \pi') \in B \}$ (linear implication).

  – $\pi \perp \pi' \iff \text{cut}(\pi, \pi')$ satisfies some $P$.

  – $A \perp = \{ \pi | \forall \pi' \in A, \pi \perp \pi' \}$ (linear negation).

  – Linear logic formulas satisfy $A = A \perp \perp$.

Transcendental Syntax (Girard, /two.pnum/zero.pnum/one.pnum/three.pnum).

Improvements on GoI.

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

• types: formulas of linear logic and more.

• Speaks about the "logic" of a computational model.
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.
  - $A \Rightarrow B = \{ \pi | \forall \pi' \in A, \text{cut}(\pi, \pi') \in B \}$ (linear implication).
  - $\pi \perp \pi' \iff \text{cut}(\pi, \pi')$ satisfies some $P$.
  - $A \perp = \{ \pi | \forall \pi' \in A, \pi \perp \pi' \}$ (linear negation).
- Linear logic formulas satisfy $A = A \perp \perp$.

Transcendental Syntax (Girard, /two.pnum/zero.pnum/one.pnum/three.pnum).
Improvements on GoI.

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

Transcendental Syntax (Girard, two.pnum/zero.pnum/one.pnum/three.pnum).
Improvements on GoI.

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

Linear logic formulas satisfy $A = A \perp \perp$. 

P205
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.

\[ A \Rightarrow B := \{ \pi \mid \forall \pi' \in A, \text{cut}(\pi, \pi') \in B \} \] (linear implication).

Transcendental Syntax (Girard, /two.pnum//eight.pnum). Improvements on GoI.

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

Speaks about the "logic" of a computational model.
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.
  - $A \Rightarrow B := \{\pi \mid \forall \pi' \in A, \text{cut}(\pi, \pi') \in B\}$ (linear implication).
  - $\pi \perp \pi' \iff \text{cut}(\pi, \pi')$ satisfies some $P$.

Transcendental Syntax (Girard, /two.pnum//eight.pnum). Improvements on GoI.

- programs: "Stellar Resolution" (Turing-complete).
- types: formulas of linear logic and more.
- Speaks about the "logic" of a computational model.
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.
  - \( A \Rightarrow B := \{ \pi | \forall \pi' \in A, \text{cut}(\pi, \pi') \in B \} \) (linear implication).
  - \( \pi \perp \pi' \iff \text{cut}(\pi, \pi') \) satisfies some \( P \).
  - \( A \perp := \{ \pi | \forall \pi' \in A, \pi \perp \pi' \} \) (linear negation).

Transcendental Syntax (Girard, two.pnum/zero.pnum/one.pnum/three.pnum).

Improvements on GoI.

- **programs**: “Stellar Resolution” (Turing-complete).
- **types**: formulas of linear logic and more.
- Speaks about the “logic” of a computational model.
Geometry of Interaction and Ludics (Girard). Reconstruction of logic from computation.

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

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.
  - \( A \Rightarrow B \) := \{ \pi \mid \forall \pi’ \in A, \text{cut}(\pi, \pi’) \in B \} (linear implication).
  - \( \pi \perp \pi’ \iff \text{cut}(\pi, \pi’) \) satisfies some \( P \).
  - \( A^\perp \) := \{ \pi \mid \forall \pi’ \in A, \pi \perp \pi’ \} (linear negation).
  - Linear logic formulas satisfy \( A = A^\perp \perp \).

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.
  - \[ A \Rightarrow B := \{ \pi \mid \forall \pi' \in A, \text{cut}(\pi, \pi') \in B \} \] (linear implication).
  - \[ \pi \perp \pi' \iff \text{cut}(\pi, \pi') \text{ satisfies some } P. \]
  - \[ A^\perp := \{ \pi \mid \forall \pi' \in A, \pi \perp \pi' \} \] (linear negation).
  - Linear logic formulas satisfy \[ A \equiv A^\perp \perp. \]

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.
  - $A \Rightarrow B := \{ \pi \mid \forall \pi' \in A, \text{cut}(\pi, \pi') \in B \}$ (linear implication).
  - $\pi \perp \pi' \iff \text{cut}(\pi, \pi')$ satisfies some $P$.
  - $A^{\perp} := \{ \pi \mid \forall \pi' \in A, \pi \perp \pi' \}$ (linear negation).
  - Linear logic formulas satisfy $A = A^{\perp\perp}$.

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

\[
g(x) \cdot \phi_1 \cdot \phi_2 \cdot \phi_3 \cdot \phi_4
\]

\[
-a(f(y)) \cdot +c(y)
\]

\[
-g(x) \cdot +a(x)
\]
Stellar Resolution

Girard’s stars and constellations

\[ \phi \]

\[ g(x) \cdot \phi_1 \cdot +a(x) \cdot -a(f(y)) \cdot +c(y) \cdot -b(x) \cdot \]

Constellation /uniEF26 (n stars) = program ↓

Diagrams (maximal tilings) ↓

Constellation Ex (/uniEF26) = normal form

A reformulation of Robinson’s /uniFB01rst-order resolution / Query-free logic programming.
Stellar Resolution

Girard’s stars and constellations

\[
g(f(y)) \cdot \phi_1 \cdot +a(x) \cdot -a(f(y)) \cdot \phi_2 \cdot +c(y)
\]

\[
-b(f(y)) \cdot
\]

A reformulation of Robinson's first-order resolution in query-free logic programming.
Stellar Resolution

Girard’s stars and constellations

\[
g(f(y)) \quad \phi_1 \cup \phi_2 \quad +c(y) \\
-b(f(y))
\]
Stellar Resolution
Girard’s stars and constellations

\[
g(f(y)) \cdot (\phi_1 \cup \phi_2) \cdot +c(y) \cdot -b(f(y)) \cdot
\]

Constellation \(\Phi\) (n stars) = program

\[\text{Diagrams (maximal tilings)}\]

\[\text{Constellation } \text{Ex}(\Phi) \]

= normal form
Stellar Resolution
Girard’s stars and constellations

\[ g(f(y)) \cdot \phi_1 \cup \phi_2 \cdot +c(y) \cdot -b(f(y)) \cdot \]

Constellation \( \Phi \) (\( n \) stars)
= program
\[ \downarrow \]
Diagrams (maximal tilings)
\[ \downarrow \]
Constellation \( \text{Ex}(\Phi) \)
= normal form
Stellar Resolution

Girard’s stars and constellations

\[ \phi_1 \cup \phi_2 \]

\[ g(f(y)) \cdot +c(y) \]

\[ \neg b(f(y)) \cdot \]

A reformulation of Robinson’s first-order resolution / Query-free logic programming.
Stellar Resolution
Automata and circuits unified

Generalised automata.
Generalised automata.

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

Run on a word ↔ tiling/diagram.
Stellar Resolution
Automata and circuits unified

Generalised automata.
Transitions \( \leftrightarrow \) binary stars \([ -a(c \cdot w, q), +a(w, q') ]\).
Run on a word \( \leftrightarrow \) tiling/diagram.

Generalised circuits.
Stellar Resolution
Automata and circuits unified

**Generalised automata.**
Transitions $\leftrightarrow$ binary stars $[-a(c \cdot w, q), +a(w, q')]$.
Run on a word $\leftrightarrow$ tiling/diagram.

**Generalised circuits.**
Gates (not) $\leftrightarrow$ star $[-c_i(x), -\text{not}(x, r), +c_j(r)]$.
Circuit evaluation $\leftrightarrow$ execution of constellation.
Stellar Resolution

Automata and circuits unified

Generalised automata.
Transitions $\leftrightarrow$ binary stars $[-a(c \cdot w, q), +a(w, q')]$.
Run on a word $\leftrightarrow$ tiling/diagram.

Generalised circuits.

Gates (not) $\leftrightarrow$ star $[-c_i(x), -\text{not}(x, r), +c_j(r)]$.
Circuit evaluation $\leftrightarrow$ 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 $A$ a set of constellations (programs).
- Choose a binary orthogonality $\perp$ for "correct interaction".
- Define $A^\perp = \{ \Phi | \forall \Phi' \in A, \Phi \perp \Phi' \}$ (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 $A$ a set of constellations (programs).
- Choose a binary orthogonality $\perp$ for "correct interaction".
- Define $A^{\perp} = \{ \Phi \mid \forall \Phi' \in A, \Phi \perp \Phi' \}$ (linear negation / duality).
- Formulas/types : $A$ such that $A = A^{\perp^{\perp}}$. 
Realisability and interactive typing

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

Reconstructing linear logic (Transcendental Syntax).

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

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

Reconstructing linear logic (Transcendental Syntax).

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

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 $\Phi$ is a proof of $A$ when:

$$
\text{Danos-Regnier criterion}
\begin{align*}
\text{Unit testing and specifications.} \\
\text{• Unit testing: a function } f \text{ is "correct" when } f(a_i) = b_i \text{ for some } (a_i, b_i). \\
\text{• Specifications: a function } f \text{ is labelled by } A \text{ when it has some behaviour } BH(A).
\end{align*}
$$
(Unit) testing in logic

*Generalising the correctness criterion*

**Transcendental Syntax.** A constellation $\Phi$ is a proof of $A$ when:

\[(\text{Danos-Regnier criterion}) \quad \Phi_1^t \quad \Phi_2^t \quad \ldots \quad \Phi_n^t\]
(Unit) testing in logic

*Generalising the correctness criterion*

**Transcendental Syntax.** A constellation $\Phi$ is a proof of $A$ when:

- (Tested constellation) $\Phi \quad \Phi \quad \Phi$

- (Danos-Regnier criterion) $\Phi^1_t \quad \Phi^2_t \quad \Phi^n_t$
(Unit) testing in logic

Generalising the correctness criterion

Transcendental Syntax. A constellation $\Phi$ is a proof of $A$ when:

- **(Tested constellation)**
  
  $\Phi \quad \Phi \quad \Phi$

  $\bot_{DR} \quad \bot_{DR} \quad \ldots \quad \bot_{DR}$

- **(Danos-Regnier criterion)**
  
  $\Phi_t^1 \quad \Phi_t^2 \quad \ldots \quad \Phi_t^n$
(Unit) testing in logic

Generalising the correctness criterion

**Transcendental Syntax.** A constellation $\Phi$ is a proof of $A$ when:

( Tested constellation ) $\Phi$ $\Phi$ $\Phi$

$\perp_{DR}$ $\perp_{DR}$ $\ldots$ $\perp_{DR}$

(Danos-Regnier criterion) $\Phi_1^t$ $\Phi_2^t$ $\ldots$ $\Phi_n^t$

Unit testing and specifications.
(Unit) testing in logic

Generalising the correctness criterion

Transcendental Syntax. A constellation $\Phi$ is a proof of $A$ when:

- (Tested constellation) $\Phi \quad \Phi \quad \Phi \quad \Phi \\
  \Downarrow_{DR} \quad \Downarrow_{DR} \quad \cdots \quad \Downarrow_{DR}$

- (Danos-Regnier criterion) $\Phi_t^{1} \quad \Phi_t^{2} \quad \cdots \quad \Phi_t^{n}$

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 $\Phi$ is a proof of $A$ when:

- (Tested constellation) $\Phi \Downarrow_1 DR \Phi \Downarrow_2 DR \cdots \Phi \Downarrow_n DR$

- (Danos-Regnier criterion) $\Phi_t^1 \Phi_t^2 \cdots \Phi_t^n$

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 $\Phi$ is a proof of $A$ when:

(Tested constellation) $\Phi$ $\Phi$ $\Phi$

$\perp_{DR}$ $\perp_{DR}$ $\ldots$ $\perp_{DR}$

(Danos-Regnier criterion) $\Phi^1_t$ $\Phi^2_t$ $\Phi^n_t$

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 $\Phi$ is a proof of $A$ when:

(Tested constellation) $\Phi \quad \Phi \quad \Phi$

$\perp_{DR} \quad \perp_{DR} \quad \ldots \quad \perp_{DR}$

(Danos-Regnier criterion) $\Phi^1_t \quad \Phi^2_t \quad \Phi^n_t$

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 $\Phi$ is correct w.r.t. $A$ when it passes some tests in $\text{Tests}(A)$. 
(Unit) testing in logic

*Generalising the correctness criterion*

**Transcendental Syntax.** A constellation $\Phi$ is a proof of $A$ when:

- *(Tested constellation)* $\Phi$ $\Phi$ $\Phi$

| $\perp_{DR}$ | $\perp_{DR}$ | $\ldots$ | $\perp_{DR}$ |

- *(Danos-Regnier criterion)* $\Phi_t^1$ $\Phi_t^2$ $\ldots$ $\Phi_t^n$

**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 $\Phi$ is correct w.r.t. $A$ when it passes some tests in $Tests(A)$.
- **Adequation**: $\Phi$ is correct w.r.t. $A \implies \Phi \in BH(A)$ with $BH(A) = BH(A) \perp \perp$. 
Atypic typing and complexity

Typing outside $\lambda$-calculus. Automata, logic programs, circuits, tile systems, ...
Atypic typing and complexity

**Typing outside** $\lambda$-calculus. Automata, logic programs, circuits, tile systems, ...

 MacOS basically information flow in a structure.
Atypic typing and complexity

**Typing outside** $\lambda$-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 $\lambda$-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.
Atypical typing and complexity

Typing outside \( \lambda \)-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 $\lambda$-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** $\lambda$-calculus. Automata, logic programs, circuits, tile systems, ...

\[ \downarrow \text{basically information flow in a structure.} \]

**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 realizability types.

Thank you for listening to my talk.