A technical reading of the Transcendental Syntax

LIPN – Université Sorbonne Paris Nord

Boris Eng

The essence of proofs

Sequent calculus.

$$\frac{}{\vdash \neg A, A} \qquad \frac{\vdash \Gamma, A \qquad \vdash \Delta, B}{\vdash \Gamma, \Delta, A \wedge B} \qquad \frac{\vdash \Gamma, A, B}{\vdash \Gamma, A \vee B}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\vdash \neg B, B} \qquad \dfrac{}{\vdash \neg A, A}}{\vdash \neg B, \neg A, B \wedge A}}{\vdash \neg A, \neg B, B \wedge A}}{\vdash \neg A \vee \neg B, B \wedge A}$$

The essence of proofs

Sequent calculus.

$$\frac{}{\vdash \neg A, A} \qquad \frac{\vdash \Gamma, A \qquad \vdash \Delta, B}{\vdash \Gamma, \Delta, A \wedge B} \qquad \frac{\vdash \Gamma, A, B}{\vdash \Gamma, A \vee B} \qquad \dfrac{\dfrac{\dfrac{\dfrac{}{\vdash \neg B, B} \qquad \dfrac{}{\vdash \neg A, A}}{\vdash \neg B, \neg A, B \wedge A}}{\vdash \neg A, \neg B, B \wedge A}}{\vdash \neg A \vee \neg B, B \wedge A}$$

Linear logic (Girard). $A \Rightarrow B = !A \multimap B$ $\qquad A, B := X_i \mid X_i^{\perp} \mid A \otimes B \mid A \parr B$ (MLL).

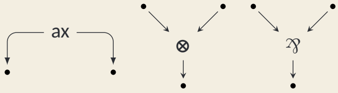
The essence of proofs

Sequent calculus.

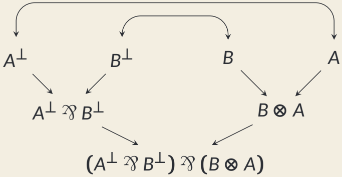
$$\frac{}{\vdash \neg A, A} \qquad \frac{\vdash \Gamma, A \qquad \vdash \Delta, B}{\vdash \Gamma, \Delta, A \wedge B} \qquad \frac{\vdash \Gamma, A, B}{\vdash \Gamma, A \vee B} \qquad \dfrac{\dfrac{\dfrac{\dfrac{}{\vdash \neg B, B} \qquad \dfrac{}{\vdash \neg A, A}}{\vdash \neg B, \neg A, B \wedge A}}{\vdash \neg A, \neg B, B \wedge A}}{\vdash \neg A \vee \neg B, B \wedge A}$$

Linear logic (Girard). $A \Rightarrow B = !A \multimap B$ $\qquad A, B := X_i \mid X_i^{\perp} \mid A \otimes B \mid A \parr B$ (MLL).

MLL proof-structures (Girard).



Axiom Tensor Par



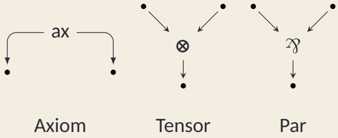
The essence of proofs

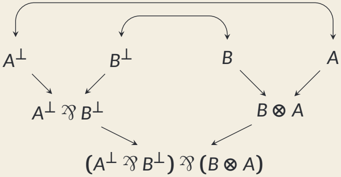
Sequent calculus.

$$\frac{}{\vdash \neg A, A} \qquad \frac{\vdash \Gamma, A \qquad \vdash \Delta, B}{\vdash \Gamma, \Delta, A \wedge B} \qquad \frac{\vdash \Gamma, A, B}{\vdash \Gamma, A \vee B} \qquad \dfrac{\dfrac{\dfrac{\dfrac{}{\vdash \neg B, B} \qquad \dfrac{}{\vdash \neg A, A}}{\vdash \neg B, \neg A, B \wedge A}}{\vdash \neg A, \neg B, B \wedge A}}{\vdash \neg A \vee \neg B, B \wedge A}$$

Linear logic (Girard). $A \Rightarrow B = !A \multimap B \qquad A, B := X_i \mid X_i^\perp \mid A \otimes B \mid A \parr B$ (MLL).

MLL proof-structures (Girard).





Other approaches. Miller's expansion trees, deep inference, ...

The computational and logical side of proofs

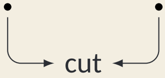
The cut rule.

$$\frac{\Gamma \vdash A \qquad \Delta, A \vdash C}{\Gamma, \Delta \vdash C}$$

The computational and logical side of proofs

The cut rule.

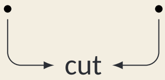
$$\frac{\Gamma \vdash A \qquad \Delta, A \vdash C}{\Gamma, \Delta \vdash C}$$



The computational and logical side of proofs

The cut rule.

$$\frac{\Gamma \vdash A \qquad \Delta, A \vdash C}{\Gamma, \Delta \vdash C}$$



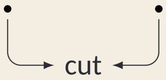
Cut-elimination (Gentzen) :

Procedure of elimination of cuts

The computational and logical side of proofs

The cut rule.

$$\frac{\Gamma \vdash A \qquad \Delta, A \vdash C}{\Gamma, \Delta \vdash C}$$



Cut-elimination (Gentzen) :

Procedure of elimination of cuts

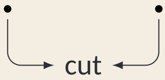
Proof-program correspondence (Curry, Howard) :

Cut-elimination ≃ program execution

The computational and logical side of proofs

The cut rule.

$$\frac{\Gamma \vdash A \qquad \Delta, A \vdash C}{\Gamma, \Delta \vdash C}$$



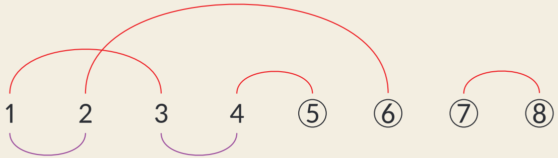
Cut-elimination (Gentzen) :

Procedure of elimination of cuts

Proof-program correspondence (Curry, Howard) :

Cut-elimination $\simeq$ program execution

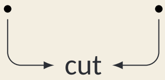
Geometry of Interaction (Girard). Dynamics of proofs with more general objects.



The computational and logical side of proofs

The cut rule.

$$\frac{\Gamma \vdash A \qquad \Delta, A \vdash C}{\Gamma, \Delta \vdash C}$$



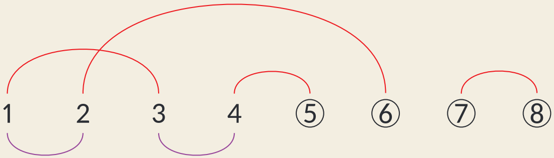
Cut-elimination (Gentzen) :

Procedure of elimination of cuts

Proof-program correspondence (Curry, Howard) :

Cut-elimination ≃ program execution

Geometry of Interaction (Girard). Dynamics of proofs with more general objects.

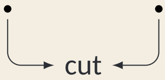


Logical correctness. How to tell if a proof-structure is "logically correct" ?

The computational and logical side of proofs

The cut rule.

$$\frac{\Gamma \vdash A \qquad \Delta, A \vdash C}{\Gamma, \Delta \vdash C}$$



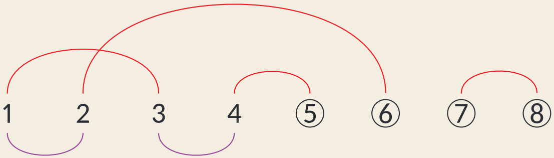
Cut-elimination (Gentzen) :

Procedure of elimination of cuts

Proof-program correspondence (Curry, Howard) :

Cut-elimination $\simeq$ program execution

Geometry of Interaction (Girard). Dynamics of proofs with more general objects.



Logical correctness. How to tell if a proof-structure is "logically correct" ?

$\rightarrow$ Danos-Regnier criterion : use some tests $\varphi_1, \ldots, \varphi_n$

Transcendental Syntax

Quest for the essence of proofs. Given definition of proof $\longmapsto$ Refinement.

Transcendental Syntax

Quest for the essence of proofs. Given definition of proof $\longmapsto$ Refinement.

Reverse engineering (GoI). Start from computation, derives linear logic.

Transcendental Syntax

Quest for the essence of proofs. Given definition of proof $\longmapsto$ Refinement.

Reverse engineering (GoI). Start from computation, derives linear logic.

Transcenscendental Syntax (Girard). Logic emerging from computation.

Transcendental Syntax

Quest for the essence of proofs. Given definition of proof $\longmapsto$ Refinement.

Reverse engineering (GoI). Start from computation, derives linear logic.

Transcenscendental Syntax (Girard). Logic emerging from computation.

- 4 vague informal papers $\longmapsto$ formalised (Eng, Seiller) ;

Transcendental Syntax

Quest for the essence of proofs. Given definition of proof $\longmapsto$ Refinement.

Reverse engineering (GoI). Start from computation, derives linear logic.

Transcenscendental Syntax (Girard). Logic emerging from computation.

- 4 vague informal papers $\longmapsto$ formalised (Eng, Seiller) ;
- introduces a model of computation "stellar resolution" (elementary bricks) ;

Transcendental Syntax

Quest for the essence of proofs. Given definition of proof ⟼ Refinement.

Reverse engineering (GoI). Start from computation, derives linear logic.

Transcenscendental Syntax (Girard). Logic emerging from computation.

- 4 vague informal papers ⟼ formalised (Eng, Seiller) ;
- introduces a model of computation "stellar resolution" (elementary bricks) ;
- “reconstruction" of linear logic.

Transcendental Syntax

Quest for the essence of proofs. Given definition of proof $\longmapsto$ Refinement.

Reverse engineering (GoI). Start from computation, derives linear logic.

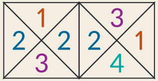
Transcenscendental Syntax (Girard). Logic emerging from computation.

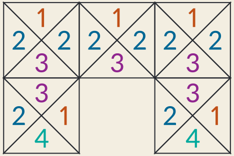
- 4 vague informal papers $\longmapsto$ formalised (Eng, Seiller) ;
- introduces a model of computation "stellar resolution" (elementary bricks) ;
- “reconstruction" of linear logic.

We begin by defining the stellar resolution.

Tile systems

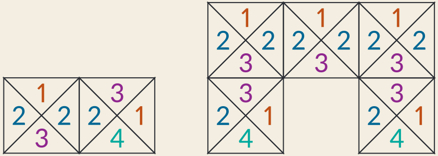
Wang tiles (Wang).





Tile systems

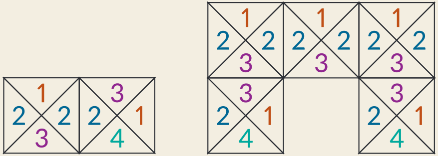
Wang tiles (Wang).



- Tiles on $\mathbf{Z}^2$.
- Turing-complete.

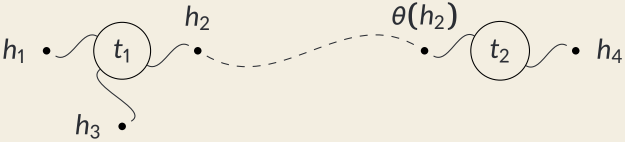
Tile systems

Wang tiles (Wang).



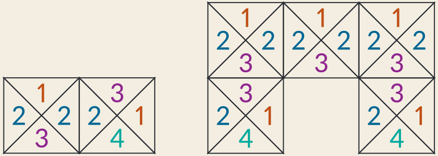
- Tiles on $\mathbf{Z}^2$.
- Turing-complete.

Flexible tiles (Jonoska).



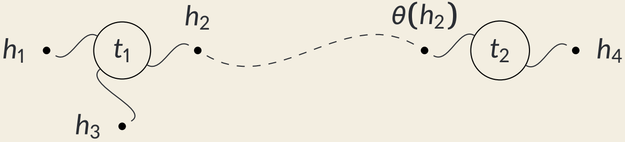
Tile systems

Wang tiles (Wang).



- Tiles on $\mathbf{Z}^2$.
- Turing-complete.

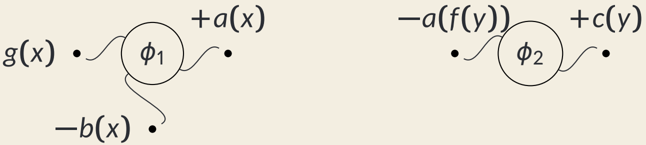
Flexible tiles (Jonoska).



- No planarity.
- Can encode "rigid tiling".
- Used in DNA computing.

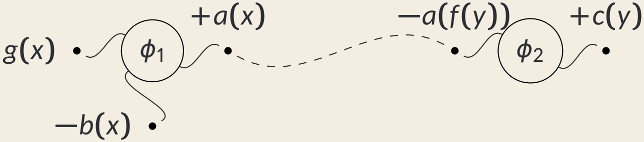
Stellar Resolution

or Girard's stars and constellations



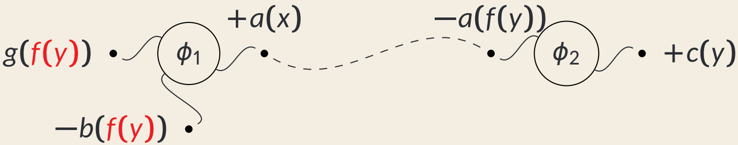
Stellar Resolution

or Girard's stars and constellations



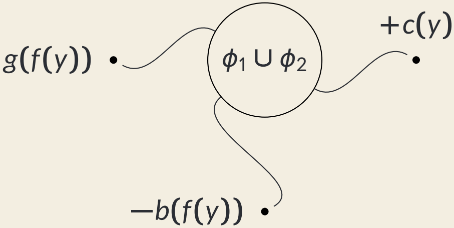
Stellar Resolution

or Girard's stars and constellations



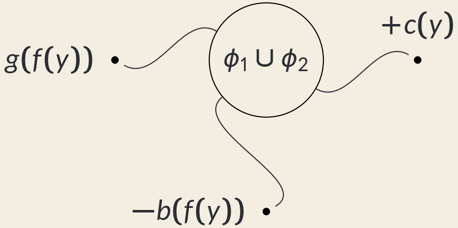
Stellar Resolution

or Girard's stars and constellations



Stellar Resolution

or Girard's stars and constellations



Constellation Φ (n stars)

$\downarrow$

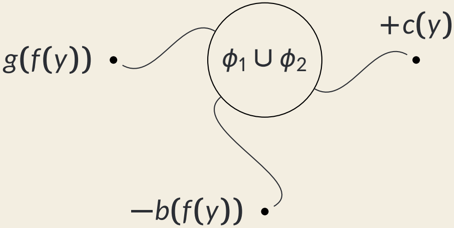
Diagrams (maximal tilings)

$\downarrow$

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

Stellar Resolution

or Girard's stars and constellations



Constellation Φ (n stars)

↓

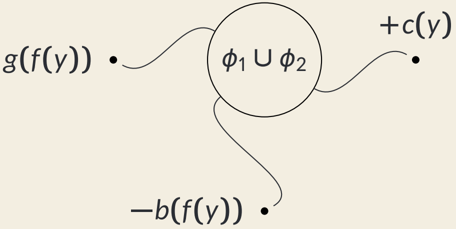
Diagrams (maximal tilings)

↓

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

Stellar Resolution

or Girard's stars and constellations



Constellation Φ (n stars)

$\downarrow$

Diagrams (maximal tilings)

$\downarrow$

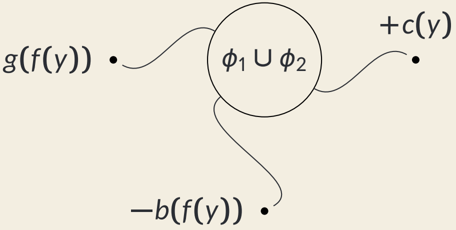
Constellation $\mathrm{Ex}(\Phi)$

A reformulation of Robinson's first-order resolution

$$[g(x), -b(x), \underline{+a(x)}] + [\underline{-a(f(y))}, +c(y)] \longrightarrow [g(f(y)), -b(f(y)), +c(y)].$$

Stellar Resolution

or Girard's stars and constellations



Constellation Φ (n stars)

$\downarrow$

Diagrams (maximal tilings)

$\downarrow$

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

A reformulation of Robinson's first-order resolution

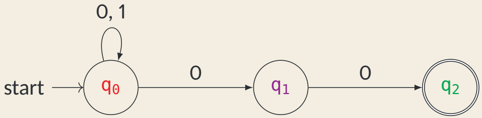
$$[g(x), -b(x), \underline{+a(x)}] + [\underline{-a(f(y))}, +c(y)] \longrightarrow [g(f(y)), -b(f(y)), +c(y)].$$

Very basic and well-known objects but new model of computation ?

Stellar Resolution

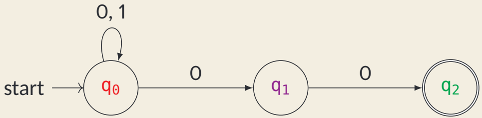
Automata and circuits unified

Extensible automata.



Stellar Resolution

Automata and circuits unified



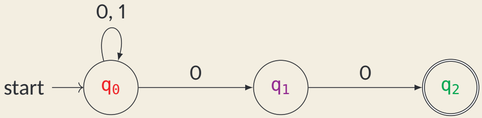
Extensible automata.

$[-i(w), +a(w, q_0)]+$

Stellar Resolution

Automata and circuits unified

Extensible automata.

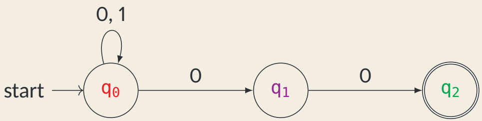


$$[-i(w), +a(w, \texttt{q}_0)] + [-a(0 \cdot w, \texttt{q}_0), +a(w, \texttt{q}_0)] + [-a(1 \cdot w, \texttt{q}_0), +a(w, \texttt{q}_0)] +$$

Stellar Resolution

Automata and circuits unified

Extensible automata.

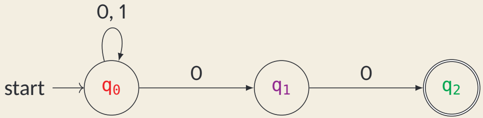


$[-i(w), +a(w, q_0)] + [-a(0 \cdot w, q_0), +a(w, q_0)] + [-a(1 \cdot w, q_0), +a(w, q_0)] +$
$[-a(0 \cdot w, q_0), +a(w, q_1)] +$

Stellar Resolution

Automata and circuits unified

Extensible automata.

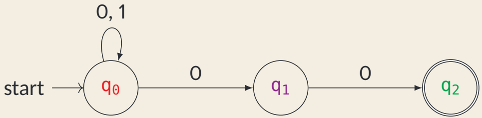


$$[-i(w), +a(w, q_0)] + [-a(0 \cdot w, q_0), +a(w, q_0)] + [-a(1 \cdot w, q_0), +a(w, q_0)] +$$
$$[-a(0 \cdot w, q_0), +a(w, q_1)] + [-a(0 \cdot w, q_1), +a(w, q_2)] +$$

Stellar Resolution

Automata and circuits unified

Extensible automata.

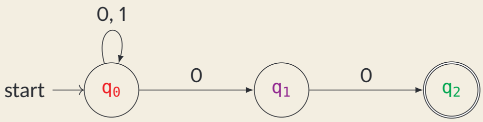


$$[-i(w), +a(w, q_0)] + [-a(0 \cdot w, q_0), +a(w, q_0)] + [-a(1 \cdot w, q_0), +a(w, q_0)] +$$
$$[-a(0 \cdot w, q_0), +a(w, q_1)] + [-a(0 \cdot w, q_1), +a(w, q_2)] + [-a(\epsilon, q_2), \mathtt{accept}]$$

Stellar Resolution

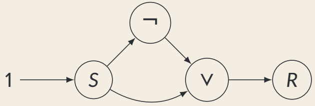
Automata and circuits unified

Extensible automata.



$$[-i(w), +a(w, q_0)] + [-a(0 \cdot w, q_0), +a(w, q_0)] + [-a(1 \cdot w, q_0), +a(w, q_0)] +$$
$$[-a(0 \cdot w, q_0), +a(w, q_1)] + [-a(0 \cdot w, q_1), +a(w, q_2)] + [-a(\epsilon, q_2), \texttt{accept}]$$

Modular circuits.

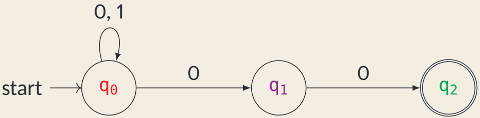


$$\Phi_{em} = [i_0(1), +c_0(1)]$$

Stellar Resolution

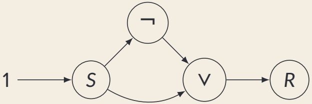
Automata and circuits unified

Extensible automata.



$[-i(w), +a(w, q_0)] + [-a(0 \cdot w, q_0), +a(w, q_0)] + [-a(1 \cdot w, q_0), +a(w, q_0)] +$
$[-a(0 \cdot w, q_0), +a(w, q_1)] + [-a(0 \cdot w, q_1), +a(w, q_2)] + [-a(\epsilon, q_2), \mathtt{accept}]$

Modular circuits.

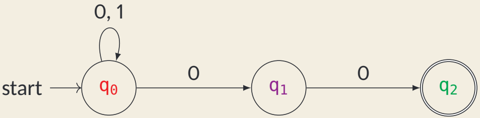


$$\Phi_{em} = [i_0(1), +c_0(1)] + [+c_0(x), +c_1(x), +c_2(x)]$$

Stellar Resolution

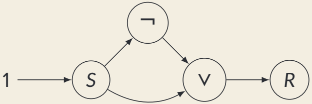
Automata and circuits unified

Extensible automata.



$$[-i(w), +a(w, q_0)] + [-a(0 \cdot w, q_0), +a(w, q_0)] + [-a(1 \cdot w, q_0), +a(w, q_0)] +$$
$$[-a(0 \cdot w, q_0), +a(w, q_1)] + [-a(0 \cdot w, q_1), +a(w, q_2)] + [-a(\epsilon, q_2), \mathtt{accept}]$$

Modular circuits.

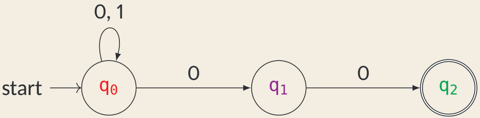


$$\Phi_{em} = [i_0(1), +c_0(1)] + [+c_0(x), +c_1(x), +c_2(x)]$$
$$+ [-c_2(x), -not(x, r), +c_3(r)] +$$
$$[-c_1(x), -c_3(y), -or(x, y, r), +c_4(r)]$$

Stellar Resolution

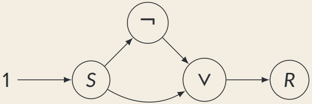
Automata and circuits unified

Extensible automata.



$$[-i(w), +a(w, q_0)] + [-a(0 \cdot w, q_0), +a(w, q_0)] + [-a(1 \cdot w, q_0), +a(w, q_0)] +$$
$$[-a(0 \cdot w, q_0), +a(w, q_1)] + [-a(0 \cdot w, q_1), +a(w, q_2)] + [-a(\epsilon, q_2), \mathtt{accept}]$$

Modular circuits.

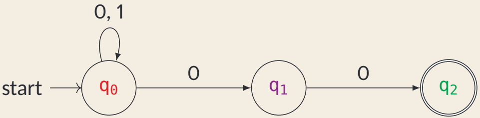


$$\Phi_{em} = [i_0(1), +c_0(1)] + [+c_0(x), +c_1(x), +c_2(x)]$$
$$+[-c_2(x), -not(x, r), +c_3(r)] +$$
$$[-c_1(x), -c_3(y), -or(x, y, r), +c_4(r)]$$
$$+[-c_4(x), r(x)]$$

Stellar Resolution

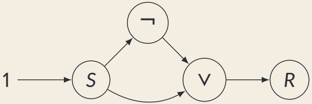
Automata and circuits unified

Extensible automata.



$$[-i(w), +a(w, q_0)] + [-a(0 \cdot w, q_0), +a(w, q_0)] + [-a(1 \cdot w, q_0), +a(w, q_0)] + [-a(0 \cdot w, q_0), +a(w, q_1)] + [-a(0 \cdot w, q_1), +a(w, q_2)] + [-a(\epsilon, q_2), \texttt{accept}]$$

Modular circuits.



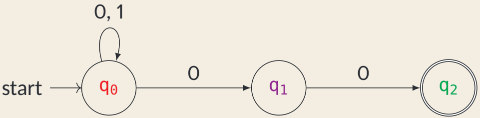
$$\Phi_{em} = [i_0(1), +c_0(1)] + [+c_0(x), +c_1(x), +c_2(x)] + [-c_2(x), -not(x, r), +c_3(r)] + [-c_1(x), -c_3(y), -or(x, y, r), +c_4(r)] + [-c_4(x), r(x)]$$

Information flow inside a structure.

Stellar Resolution

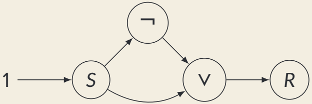
Automata and circuits unified

Extensible automata.



$$[-i(w), +a(w, q_0)] + [-a(0 \cdot w, q_0), +a(w, q_0)] + [-a(1 \cdot w, q_0), +a(w, q_0)] +$$
$$[-a(0 \cdot w, q_0), +a(w, q_1)] + [-a(0 \cdot w, q_1), +a(w, q_2)] + [-a(\epsilon, q_2), \mathtt{accept}]$$

Modular circuits.

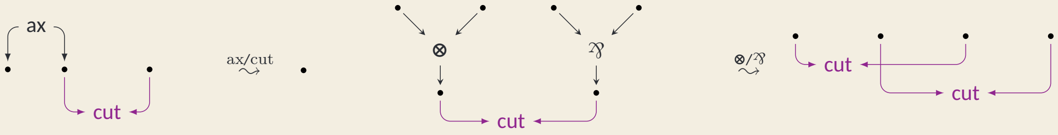


$$\Phi_{em} = [i_0(1), +c_0(1)] + [+c_0(x), +c_1(x), +c_2(x)]$$
$$+[-c_2(x), -not(x, r), +c_3(r)] \quad +$$
$$[-c_1(x), -c_3(y), -or(x, y, r), +c_4(r)]$$
$$+[-c_4(x), r(x)]$$

Information flow inside a structure. Turing-complete.

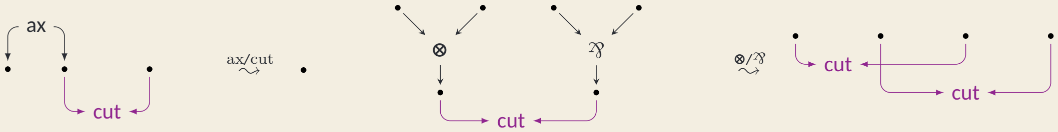
Reconstruction of the computational content of MLL

Cut-elimination for MLL (program execution). (Hyper)graph rewriting.

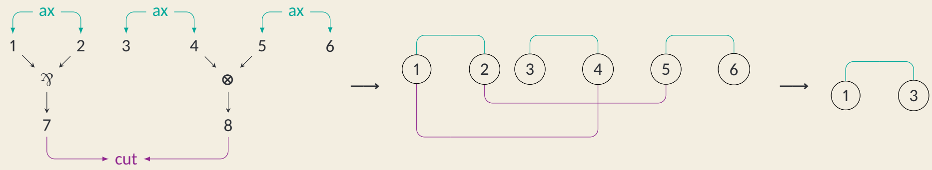


Reconstruction of the computational content of MLL

Cut-elimination for MLL (program execution). (Hyper)graph rewriting.

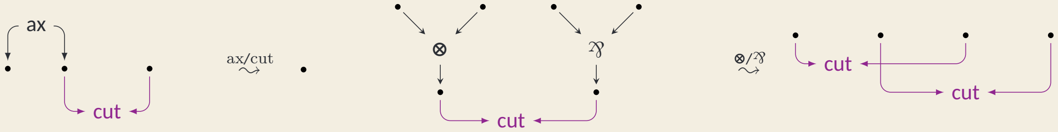


Geometry of Interaction. Maximal paths between axioms and cuts.

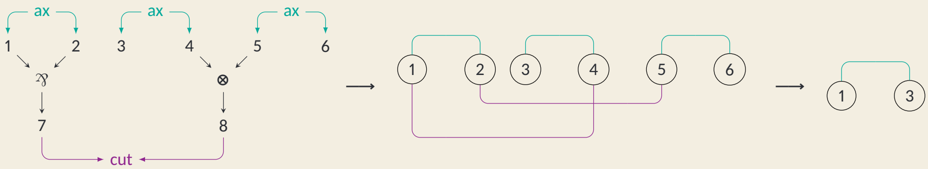


Reconstruction of the computational content of MLL

Cut-elimination for MLL (program execution). (Hyper)graph rewriting.



Geometry of Interaction. Maximal paths between axioms and cuts.

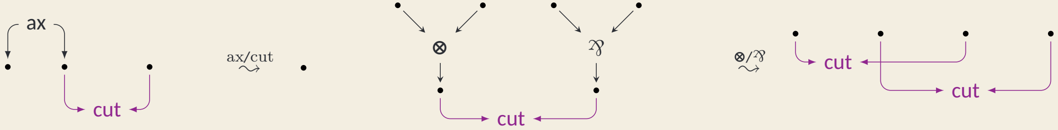


Transcendental Syntax (actually GoI 3). Tiling of binary stars.

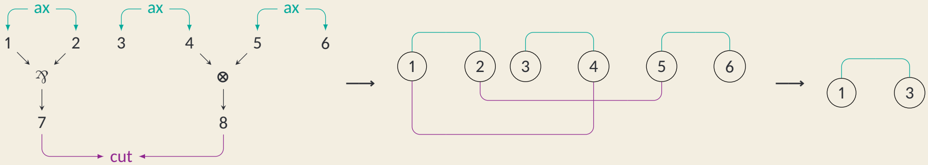
$$[+p_7(\mathtt{l} \cdot x), +p_7(\mathtt{r} \cdot x)]$$

Reconstruction of the computational content of MLL

Cut-elimination for MLL (program execution). (Hyper)graph rewriting.



Geometry of Interaction. Maximal paths between axioms and cuts.

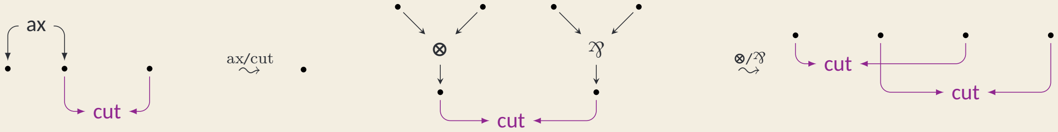


Transcendental Syntax (actually GoI 3). Tiling of binary stars.

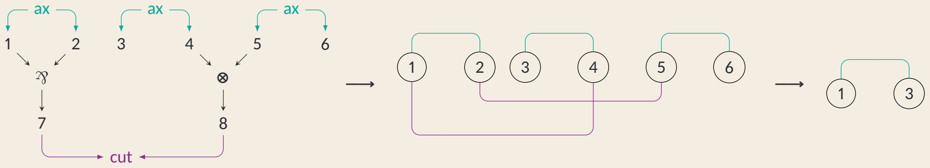
$$[+p_7(\mathtt{l}\cdot x), +p_7(\mathtt{r}\cdot x)] + [+p_3(x), +p_8(\mathtt{l}\cdot x)] + [+p_8(\mathtt{r}\cdot x), +p_6(x)]$$

Reconstruction of the computational content of MLL

Cut-elimination for MLL (program execution). (Hyper)graph rewriting.



Geometry of Interaction. Maximal paths between axioms and cuts.



Transcendental Syntax (actually GoI 3). Tiling of binary stars.

$[+p_7(\mathtt{l}\cdot x), +p_7(\mathtt{r}\cdot x)] + [+p_3(x), +p_8(\mathtt{l}\cdot x)] + [+p_8(\mathtt{r}\cdot x), +p_6(x)]$
$[-p_7(x), -p_8(x)]$.

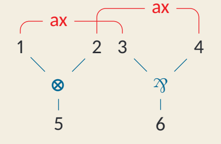
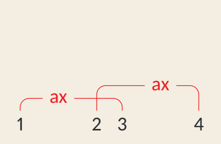
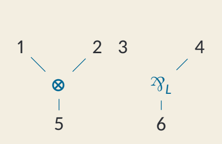
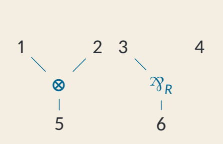
Reconstruction of the logical content of MLL

Only some proof-structure are "logically correct".

Reconstruction of the logical content of MLL

Only some proof-structure are "logically correct".

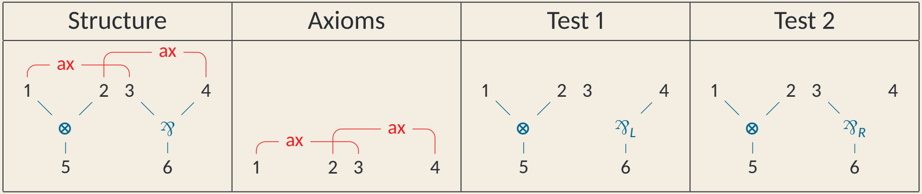
Danos-Regnier correctness criterion.

Structure	Axioms	Test 1	Test 2
			

Reconstruction of the logical content of MLL

Only some proof-structure are "logically correct".

Danos-Regnier correctness criterion.

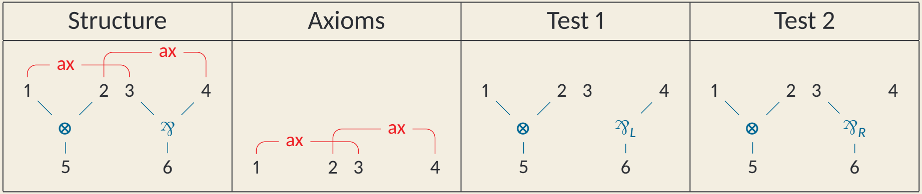


If Axioms+Test(i) is a tree, the structure is logically correct.

Reconstruction of the logical content of MLL

Only some proof-structure are "logically correct".

Danos-Regnier correctness criterion.



If Axioms+Test(i) is a tree, the structure is logically correct.

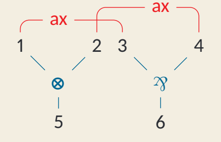
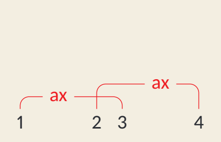
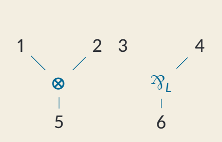
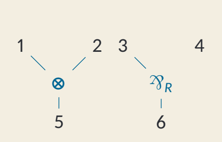
Transcendental Syntax. Testing as interaction between constellations.

- $\mathrm{Ex}(\Phi^{\mathrm{ax}}_{\mathscr{S}} \uplus \Phi^{\mathrm{test}(i)}_{\mathscr{S}}) = [p_1(x), \dots, p_2(x)]$ with conclusions $\{1, \dots, n\}$.

Reconstruction of the logical content of MLL

Only some proof-structure are "logically correct".

Danos-Regnier correctness criterion.

Structure	Axioms	Test 1	Test 2
			

If Axioms+Test(i) is a tree, the structure is logically correct.

Transcendental Syntax. Testing as interaction between constellations.

- $\mathrm{Ex}(\Phi^{\mathrm{ax}}_{\mathscr{S}} \uplus \Phi^{\mathrm{test}(i)}_{\mathscr{S}}) = [p_1(x), \ldots, p_2(x)]$ with conclusions $\{1, \ldots, n\}$.
- $\mathrm{Ex}(\Phi^{\mathrm{ax}}_{\mathscr{S}} \uplus \Phi^{\mathrm{test}(i)}_{\mathscr{S}})$ strongly normalising (MLL+MIX).

(Unit) testing in logic et primitive typing

Generalising the correctness criterion

Unit testing in programming. A program f "correct" if $f(a_i) = b_i$ for $1 \leq i \leq n$.

(Unit) testing in logic et primitive typing

Generalising the correctness criterion

Unit testing in programming. A program f "correct" if $f(a_i) = b_i$ for $1 \leq i \leq n$.

Transcendental testing. A constellation Φ is "correct" if $P(\Phi, \Phi_i)$ for $1 \leq i \leq n$.

(Unit) testing in logic et primitive typing

Generalising the correctness criterion

Unit testing in programming. A program f "correct" if $f(a_i) = b_i$ for $1 \leq i \leq n$.

Transcendental testing. A constellation Φ is "correct" if $P(\Phi, \Phi_i)$ for $1 \leq i \leq n$.

- Induces an orthogonality relation $\Phi \perp \Phi' \iff P(\Phi, \Phi')$ "passing the test".

(Unit) testing in logic et primitive typing

Generalising the correctness criterion

Unit testing in programming. A program f "correct" if $f(a_i) = b_i$ for $1 \leq i \leq n$.

Transcendental testing. A constellation Φ is "correct" if $P(\Phi, \Phi_i)$ for $1 \leq i \leq n$.

- Induces an orthogonality relation $\Phi \perp \Phi' \iff P(\Phi, \Phi')$ "passing the test".
- Set of tests `Tests` and orthogonal $\texttt{Tests}^{\perp}$ (all objects passing the tests).

(Unit) testing in logic et primitive typing

Generalising the correctness criterion

Unit testing in programming. A program f "correct" if $f(a_i) = b_i$ for $1 \leq i \leq n$.

Transcendental testing. A constellation Φ is "correct" if $P(\Phi, \Phi_i)$ for $1 \leq i \leq n$.

- Induces an orthogonality relation $\Phi \perp \Phi' \iff P(\Phi, \Phi')$ "passing the test".
- Set of tests $\texttt{Tests}$ and orthogonal $\texttt{Tests}^\perp$ (all objects passing the tests).
- Reformulation : a constellation Φ is correct (w.r.t. $\perp$) $\iff \Phi \in \texttt{Tests}^\perp$.

(Unit) testing in logic et primitive typing

Generalising the correctness criterion

Unit testing in programming. A program f "correct" if $f(a_i) = b_i$ for $1 \leq i \leq n$.

Transcendental testing. A constellation Φ is "correct" if $P(\Phi, \Phi_i)$ for $1 \leq i \leq n$.

- Induces an orthogonality relation $\Phi \perp \Phi' \iff P(\Phi, \Phi')$ "passing the test".
- Set of tests $\texttt{Tests}$ and orthogonal $\texttt{Tests}^\perp$ (all objects passing the tests).
- Reformulation : a constellation Φ is correct (w.r.t. $\perp$) $\iff \Phi \in \texttt{Tests}^\perp$.

Primitive typing generalised. Proof theory, type theory, programming languages, ...

(Unit) testing in logic et primitive typing

Generalising the correctness criterion

Unit testing in programming. A program f "correct" if $f(a_i) = b_i$ for $1 \leq i \leq n$.

Transcendental testing. A constellation Φ is "correct" if $P(\Phi, \Phi_i)$ for $1 \leq i \leq n$.

- Induces an orthogonality relation $\Phi \perp \Phi' \iff P(\Phi, \Phi')$ "passing the test".
- Set of tests $\texttt{Tests}$ and orthogonal $\texttt{Tests}^{\perp}$ (all objects passing the tests).
- Reformulation : a constellation Φ is correct (w.r.t. $\perp$) $\iff \Phi \in \texttt{Tests}^{\perp}$.

Primitive typing generalised. Proof theory, type theory, programming languages, ...

- Type label A with tests $\texttt{Tests}(A)$ (finite and chosen).

(Unit) testing in logic et primitive typing

Generalising the correctness criterion

Unit testing in programming. A program f "correct" if $f(a_i) = b_i$ for $1 \leq i \leq n$.

Transcendental testing. A constellation Φ is "correct" if $P(\Phi, \Phi_i)$ for $1 \leq i \leq n$.

- Induces an orthogonality relation $\Phi \perp \Phi' \iff P(\Phi, \Phi')$ "passing the test".
- Set of tests $\texttt{Tests}$ and orthogonal $\texttt{Tests}^\perp$ (all objects passing the tests).
- Reformulation : a constellation Φ is correct (w.r.t. $\perp$) $\iff \Phi \in \texttt{Tests}^\perp$.

Primitive typing generalised. Proof theory, type theory, programming languages, ...

- Type label A with tests $\texttt{Tests}(A)$ (finite and chosen).
- $t : A \iff t \in \texttt{Tests}(A)^\perp$.

Realisability and interactive typing

Realisability applied to linear logic

Realisability. $\mathbf{A} = \{t_1, \ldots, t_k, \ldots\}$ (computational behaviour) and typing with $t \in \mathbf{A}$.

Realisability and interactive typing

Realisability applied to linear logic

Realisability. $\mathbf{A} = \{t_1, \ldots, t_k, \ldots\}$ (computational behaviour) and typing with $t \in \mathbf{A}$.

Application of linear logic. Ludics (Girard), concurrent realisability (Beffara), ...

Realisability and interactive typing

Realisability applied to linear logic

Realisability. $\mathbf{A} = \{t_1, \ldots, t_k, \ldots\}$ (computational behaviour) and typing with $t \in \mathbf{A}$.

Application of linear logic. Ludics (Girard), concurrent realisability (Beffara), ...

- Choose a binary relation $\perp$ for "correct interaction" (e.g. program vs environment).

Realisability and interactive typing

Realisability applied to linear logic

Realisability. $\mathbf{A} = \{t_1, \ldots, t_k, \ldots\}$ (computational behaviour) and typing with $t \in \mathbf{A}$.

Application of linear logic. Ludics (Girard), concurrent realisability (Beffara), ...

- Choose a binary relation $\perp$ for "correct interaction" (e.g. program vs environment).
- Define $\mathbf{A}^{\perp} = \{\Phi \mid \forall \Phi' \in \mathbf{A}, \Phi \perp \Phi'\}$ (linear negation / duality).

Realisability and interactive typing

Realisability applied to linear logic

Realisability. $\mathbf{A} = \{t_1, \ldots, t_k, \ldots\}$ (computational behaviour) and typing with $t \in \mathbf{A}$.

Application of linear logic. Ludics (Girard), concurrent realisability (Beffara), ...

- Choose a binary relation $\perp$ for "correct interaction" (e.g. program vs environment).
- Define $\mathbf{A}^{\perp} = \{\Phi \mid \forall \Phi' \in \mathbf{A}, \Phi \perp \Phi'\}$ (linear negation / duality).
- $\mathbf{A}$ conduct $\iff \mathbf{A} = \mathbf{A}^{\perp\perp} \iff \exists \mathbf{B}.\ \mathbf{A} = \mathbf{B}^{\perp}$ (**A** characterised by tests **B**).

Realisability and interactive typing

Realisability applied to linear logic

Realisability. $\mathbf{A} = \{t_1, \ldots, t_k, \ldots\}$ (computational behaviour) and typing with $t \in \mathbf{A}$.

Application of linear logic. Ludics (Girard), concurrent realisability (Beffara), ...

- Choose a binary relation $\perp$ for "correct interaction" (e.g. program vs environment).
- Define $\mathbf{A}^\perp = \{\Phi \mid \forall \Phi' \in \mathbf{A}, \Phi \perp \Phi'\}$ (linear negation / duality).
- $\mathbf{A}$ conduct $\iff \mathbf{A} = \mathbf{A}^{\perp\perp} \iff \exists \mathbf{B}. \mathbf{A} = \mathbf{B}^\perp$ ($\mathbf{A}$ characterised by tests $\mathbf{B}$).
- Assembling conducts : $\mathbf{A} \otimes \mathbf{B} = \{\Phi_A \uplus \Phi_B \mid \Phi_A \in \mathbf{A}, \Phi_B \in \mathbf{B}\}^{\perp\perp}$.

Realisability and interactive typing

Realisability applied to linear logic

Realisability. $\mathbf{A} = \{t_1, \ldots, t_k, \ldots\}$ (computational behaviour) and typing with $t \in \mathbf{A}$.

Application of linear logic. Ludics (Girard), concurrent realisability (Beffara), ...

- Choose a binary relation $\perp$ for "correct interaction" (e.g. program vs environment).
- Define $\mathbf{A}^\perp = \{\Phi \mid \forall \Phi' \in \mathbf{A}, \Phi \perp \Phi'\}$ (linear negation / duality).
- $\mathbf{A}$ conduct $\iff \mathbf{A} = \mathbf{A}^{\perp\perp} \iff \exists \mathbf{B}.\ \mathbf{A} = \mathbf{B}^\perp$ ($\mathbf{A}$ characterised by tests $\mathbf{B}$).
- Assembling conducts : $\mathbf{A} \otimes \mathbf{B} = \{\Phi_A \uplus \Phi_B \mid \Phi_A \in \mathbf{A}, \Phi_B \in \mathbf{B}\}^{\perp\perp}$.
- Retrieving 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

Realisability applied to linear logic

Realisability. $\mathbf{A} = \{t_1, \ldots, t_k, \ldots\}$ (computational behaviour) and typing with $t \in \mathbf{A}$.

Application of linear logic. Ludics (Girard), concurrent realisability (Beffara), ...

- Choose a binary relation $\perp$ for "correct interaction" (e.g. program vs environment).
- Define $\mathbf{A}^\perp = \{\Phi \mid \forall \Phi' \in \mathbf{A}, \Phi \perp \Phi'\}$ (linear negation / duality).
- $\mathbf{A}$ conduct $\iff \mathbf{A} = \mathbf{A}^{\perp\perp} \iff \exists \mathbf{B}.\ \mathbf{A} = \mathbf{B}^\perp$ ($\mathbf{A}$ characterised by tests $\mathbf{B}$).
- Assembling conducts : $\mathbf{A} \otimes \mathbf{B} = \{\Phi_A \uplus \Phi_B \mid \Phi_A \in \mathbf{A}, \Phi_B \in \mathbf{B}\}^{\perp\perp}$.
- Retrieving other connectives : $\mathbf{A} \parr \mathbf{B} = (\mathbf{A}^\perp \otimes \mathbf{B}^\perp)^\perp$ and $\mathbf{A} \multimap \mathbf{B} = \mathbf{A}^\perp \parr \mathbf{B}$.

The two typing unified. label tests $\mathtt{Tests}(A)^\perp$ (finite) vs conduct $\mathbf{A}$ (potentially infinite).

Realisability and interactive typing

Realisability applied to linear logic

Realisability. $\mathbf{A} = \{t_1, \ldots, t_k, \ldots\}$ (computational behaviour) and typing with $t \in \mathbf{A}$.

Application of linear logic. Ludics (Girard), concurrent realisability (Beffara), ...

- Choose a binary relation $\perp$ for "correct interaction" (e.g. program vs environment).
- Define $\mathbf{A}^{\perp} = \{\Phi \mid \forall \Phi' \in \mathbf{A}, \Phi \perp \Phi'\}$ (linear negation / duality).
- $\mathbf{A}$ conduct $\iff \mathbf{A} = \mathbf{A}^{\perp\perp} \iff \exists \mathbf{B}.\ \mathbf{A} = \mathbf{B}^{\perp}$ ($\mathbf{A}$ characterised by tests $\mathbf{B}$).
- Assembling conducts : $\mathbf{A} \otimes \mathbf{B} = \{\Phi_A \uplus \Phi_B \mid \Phi_A \in \mathbf{A}, \Phi_B \in \mathbf{B}\}^{\perp\perp}$.
- Retrieving other connectives : $\mathbf{A} \parr \mathbf{B} = (\mathbf{A}^{\perp} \otimes \mathbf{B}^{\perp})^{\perp}$ and $\mathbf{A} \multimap \mathbf{B} = \mathbf{A}^{\perp} \parr \mathbf{B}$.

The two typing unified. label tests $\texttt{Tests}(A)^{\perp}$ (finite) vs conduct $\mathbf{A}$ (potentially infinite).

Adequation. $\texttt{Tests}(A)^{\perp} \subseteq \mathbf{A}$.

Realisability and interactive typing

Realisability applied to linear logic

Realisability. $\mathbf{A} = \{t_1, \ldots, t_k, \ldots\}$ (computational behaviour) and typing with $t \in \mathbf{A}$.

Application of linear logic. Ludics (Girard), concurrent realisability (Beffara), ...

- Choose a binary relation $\perp$ for "correct interaction" (e.g. program vs environment).
- Define $\mathbf{A}^\perp = \{\Phi \mid \forall \Phi' \in \mathbf{A}, \Phi \perp \Phi'\}$ (linear negation / duality).
- $\mathbf{A}$ conduct $\iff \mathbf{A} = \mathbf{A}^{\perp\perp} \iff \exists \mathbf{B}.\ \mathbf{A} = \mathbf{B}^\perp$ ($\mathbf{A}$ characterised by tests $\mathbf{B}$).
- Assembling conducts : $\mathbf{A} \otimes \mathbf{B} = \{\Phi_A \uplus \Phi_B \mid \Phi_A \in \mathbf{A}, \Phi_B \in \mathbf{B}\}^{\perp\perp}$.
- Retrieving other connectives : $\mathbf{A} \parr \mathbf{B} = (\mathbf{A}^\perp \otimes \mathbf{B}^\perp)^\perp$ and $\mathbf{A} \multimap \mathbf{B} = \mathbf{A}^\perp \parr \mathbf{B}$.

The two typing unified. label tests $\mathtt{Tests}(A)^\perp$ (finite) vs conduct $\mathbf{A}$ (potentially infinite).

Adequation. $\mathtt{Tests}(A)^\perp \subseteq \mathbf{A}$.

Typing of tests for MLL. $\mathtt{Tests}(A) \subseteq \mathbf{A}^\perp$. Co-existence with correctness witnesses.

Playground

On independent subjects

Atypic typing and complexity

Computational objects. Automata, logic programs, circuits, tiling models, ...

Atypic typing and complexity

Computational objects. Automata, logic programs, circuits, tiling models, ...

Two understanding of types.

- Type labels : specification with finite testing.

Atypic typing and complexity

Computational objects. Automata, logic programs, circuits, tiling models, ...

Two understanding of types.

- Type labels : specification with finite testing.
- Interactive types : behavioural analysis.

Atypic typing and complexity

Computational objects. Automata, logic programs, circuits, tiling models, ...

Two understanding of types.

- Type labels : specification with finite testing.
- Interactive types : behavioural analysis.

Implicit Computational Complexity (ICC). Capture classes with models.

Atypic typing and complexity

Computational objects. Automata, logic programs, circuits, tiling models, ...

Two understanding of types.

- Type labels : specification with finite testing.
- Interactive types : behavioural analysis.

Implicit Computational Complexity (ICC). Capture classes with models.

Previous works with flows (= binary stars) by Aubert & Bagnol. Capture of classes **P** and **(N)L** (with pointer machines).

Atypic typing and complexity

Computational objects. Automata, logic programs, circuits, tiling models, ...

Two understanding of types.

- Type labels : specification with finite testing.
- Interactive types : behavioural analysis.

Implicit Computational Complexity (ICC). Capture classes with models.

Previous works with flows (= binary stars) by Aubert & Bagnol. Capture of classes **P** and **(N)L** (with pointer machines).

$\longrightarrow$ Stellar Resolution can speak about **NP** and more. But what for ?

Atypic typing and complexity

Computational objects. Automata, logic programs, circuits, tiling models, ...

Two understanding of types.

- Type labels : specification with finite testing.
- Interactive types : behavioural analysis.

Implicit Computational Complexity (ICC). Capture classes with models.

Previous works with flows (= binary stars) by Aubert & Bagnol. Capture of classes **P** and **(N)L** (with pointer machines).

$\longrightarrow$ Stellar Resolution can speak about **NP** and more. But what for ?

Descriptive complexity. Capture classes with formulas.

- **P** and **NP** as classes of formulas (Immerman, Fagin).
- What about finite model theory (Model theory with finite structures/universes) ?

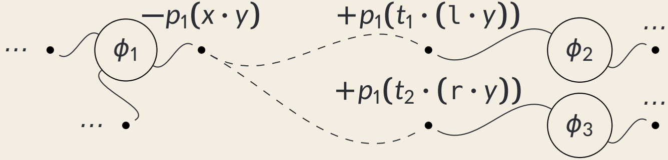
Extension to exponentials of linear logic

IMELL. $A, B ::= X_i \mid X_i^\perp \mid A \otimes B \mid A \parr B \mid A \Rightarrow B \mid (A \Rightarrow B)^\perp$ (arbitrary use of A in $A \Rightarrow B$).

Extension to exponentials of linear logic

IMELL. $A, B ::= X_i \mid X_i^\perp \mid A \otimes B \mid A \parr B \mid A \Rightarrow B \mid (A \Rightarrow B)^\perp$ (arbitrary use of A in $A \Rightarrow B$).

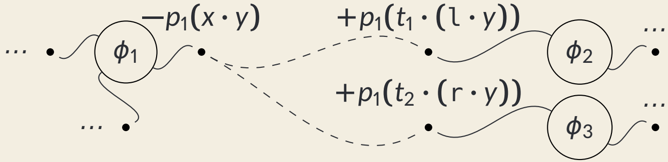
Duplication.



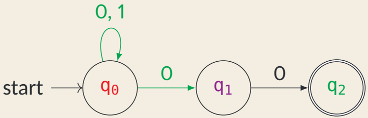
Extension to exponentials of linear logic

IMELL. $A, B ::= X_i \mid X_i^{\perp} \mid A \otimes B \mid A \parr B \mid A \Rightarrow B \mid (A \Rightarrow B)^{\perp}$ (arbitrary use of A in $A \Rightarrow B$).

Duplication.



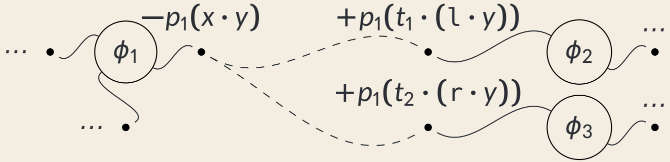
Erasure.



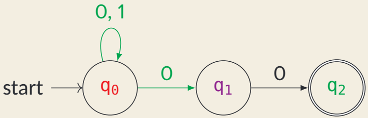
Extension to exponentials of linear logic

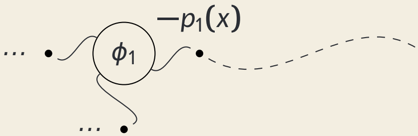
IMELL. $A, B ::= X_i \mid X_i^\perp \mid A \otimes B \mid A \parr B \mid A \Rightarrow B \mid (A \Rightarrow B)^\perp$ (arbitrary use of A in $A \Rightarrow B$).

Duplication.



Erasure.

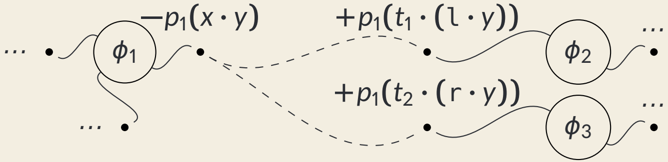




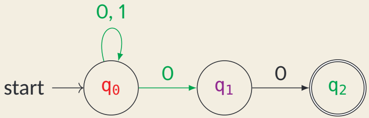
Extension to exponentials of linear logic

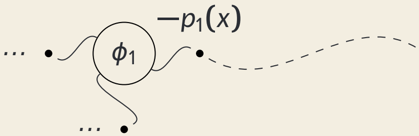
IMELL. $A, B ::= X_i \mid X_i^\perp \mid A \otimes B \mid A \parr B \mid A \Rightarrow B \mid (A \Rightarrow B)^\perp$ (arbitrary use of A in $A \Rightarrow B$).

Duplication.



Erasure.



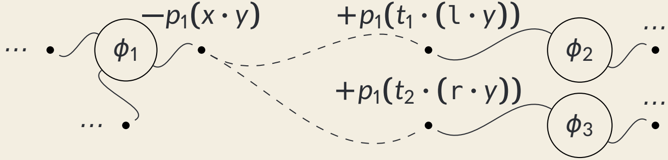


Logical correctness for IMELL. Work in progress. Uses features of stellar resolution.

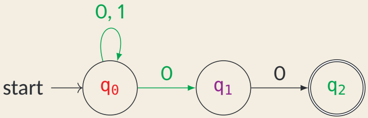
Extension to exponentials of linear logic

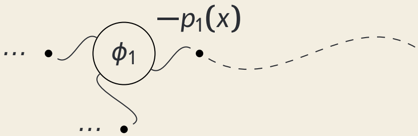
IMELL. $A, B ::= X_i \mid X_i^\perp \mid A \otimes B \mid A \parr B \mid A \Rightarrow B \mid (A \Rightarrow B)^\perp$ (arbitrary use of A in $A \Rightarrow B$).

Duplication.



Erasure.





Logical correctness for IMELL. Work in progress. Uses features of stellar resolution.

Alternative exponentials. Girard's expansionals **↓A**, **↑A**.

Conclusion

A new model of computation : Stellar Resolution.

Conclusion

A new model of computation : Stellar Resolution.

⟶ Turing complete, generalised circuit-automata.

Conclusion

A new model of computation : Stellar Resolution.

⟶ Turing complete, generalised circuit-automata.

Can naturally encode proof-nets with both cut-elimination and logical correctness.

Conclusion

A new model of computation : Stellar Resolution.

⟶ Turing complete, generalised circuit-automata.

Can naturally encode proof-nets with both cut-elimination and logical correctness.

⟶ More generally, can speak about "logic" in a broader sense.

Conclusion

A new model of computation : Stellar Resolution.
⟶ Turing complete, generalised circuit-automata.

Can naturally encode proof-nets with both cut-elimination and logical correctness.
⟶ More generally, can speak about "logic" in a broader sense.
⟶ Two understanding of types in the same framework (labels vs behavioural descriptions).

Conclusion

A new model of computation : Stellar Resolution.
⟶ Turing complete, generalised circuit-automata.

Can naturally encode proof-nets with both cut-elimination and logical correctness.
⟶ More generally, can speak about "logic" in a broader sense.
⟶ Two understanding of types in the same framework (labels vs behavioural descriptions).

A lot of connexions yet to be understood : models of DNA computing, automata theory, computational complexity, relationship between logic and computation, complex systems, ...

Conclusion

A new model of computation : Stellar Resolution.
⟶ Turing complete, generalised circuit-automata.

Can naturally encode proof-nets with both cut-elimination and logical correctness.
⟶ More generally, can speak about "logic" in a broader sense.
⟶ Two understanding of types in the same framework (labels vs behavioural descriptions).

A lot of connexions yet to be understood : models of DNA computing, automata theory, computational complexity, relationship between logic and computation, complex systems, ...

Thank you for listening to my talk.