A taste of Girard’s Transcendental Syntax

Team LoVe – LIPN Université Sorbone Paris Nord

Boris ENG & Thomas Seiller

Transcendental Syntax

Geometry of Interaction : proof-nets from the mathematics of cut-elimination

Transcendental Syntax

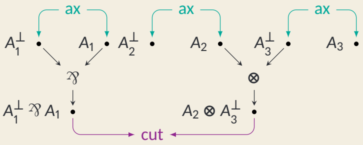
Geometry of Interaction : proof-nets from the mathematics of cut-elimination

- "Multiplicatives" : proofs are permutations, cut-elimination connects permutations

Transcendental Syntax

Geometry of Interaction : proof-nets from the mathematics of cut-elimination

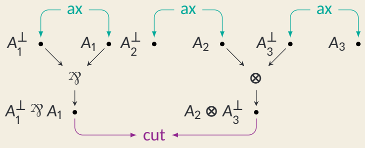
- "Multiplicatives" : proofs are permutations, cut-elimination connects permutations

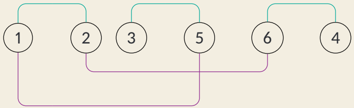


Transcendental Syntax

Geometry of Interaction : proof-nets from the mathematics of cut-elimination

- "Multiplicatives" : proofs are permutations, cut-elimination connects permutations

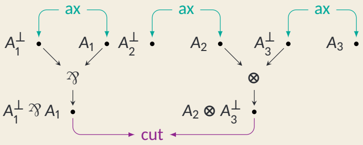


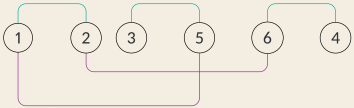


Transcendental Syntax

Geometry of Interaction : proof-nets from the mathematics of cut-elimination

- "Multiplicatives" : proofs are permutations, cut-elimination connects permutations



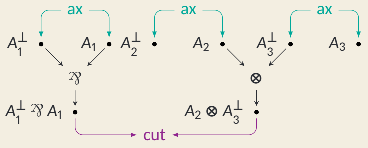


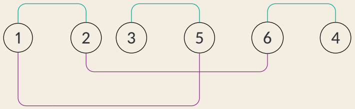
- "GoI 1,2,4,5" : interpretation operator algebras

Transcendental Syntax

Geometry of Interaction : proof-nets from the mathematics of cut-elimination

- "Multiplicatives" : proofs are permutations, cut-elimination connects permutations



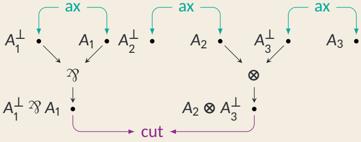


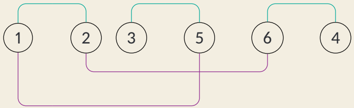
- "GoI 1,2,4,5" : interpretation operator algebras
- GoI 3 :

Transcendental Syntax

Geometry of Interaction : proof-nets from the mathematics of cut-elimination

- "Multiplicatives" : proofs are permutations, cut-elimination connects permutations



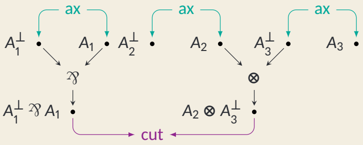


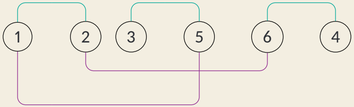
- "GoI 1,2,4,5" : interpretation operator algebras
- GoI 3 :
  - proofs as pairs of terms $(a_1 \leftrightharpoons b_1) + \ldots + (a_n \leftrightharpoons b_n)$ (flows)

Transcendental Syntax

Geometry of Interaction : proof-nets from the mathematics of cut-elimination

- "Multiplicatives" : proofs are permutations, cut-elimination connects permutations



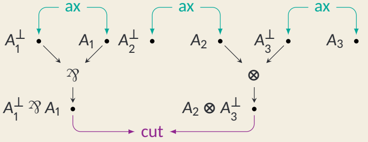


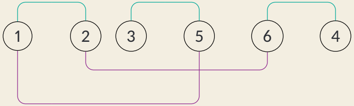
- "GoI 1,2,4,5" : interpretation operator algebras
- GoI 3 :
  - proofs as pairs of terms $(a_1 \leftrightharpoons b_1) + \ldots + (a_n \leftrightharpoons b_n)$ (flows)
  - cut-elimination as resolution (unification)

Transcendental Syntax

Geometry of Interaction : proof-nets from the mathematics of cut-elimination

- "Multiplicatives" : proofs are permutations, cut-elimination connects permutations



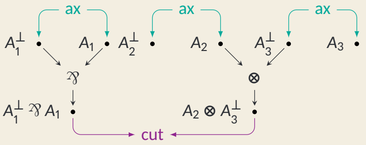


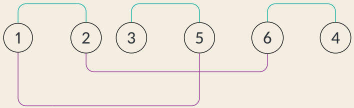
- "GoI 1,2,4,5" : interpretation operator algebras
- GoI 3 :
  - proofs as pairs of terms $(a_1 \leftrightharpoons b_1) + \ldots + (a_n \leftrightharpoons b_n)$ (flows)
  - cut-elimination as resolution (unification)
- GoI 6 : extension of this approach

Transcendental Syntax

Geometry of Interaction : proof-nets from the mathematics of cut-elimination

- "Multiplicatives" : proofs are permutations, cut-elimination connects permutations





- "GoI 1,2,4,5" : interpretation operator algebras
- GoI 3 :
  - proofs as pairs of terms $(a_1 \leftrightharpoons b_1) + \ldots + (a_n \leftrightharpoons b_n)$ (flows)
  - cut-elimination as resolution (unification)
- GoI 6 : extension of this approach
- Transcendental Syntax : same but with different name and motivations.

Stellar Resolution

Term unification

First-order terms. $t, u ::= x \mid f(t_1, \ldots, t_n)$

Stellar Resolution

Term unification

First-order terms. $t, u ::= x \mid f(t_1, \dots, t_n)$

Unification. $t_1 \doteq t_2$: can we find $\theta : Vars \longmapsto Terms$ such that $\theta t_1 = \theta t_2$?

Stellar Resolution

Term unification

First-order terms. $t, u ::= x \mid f(t_1, \ldots, t_n)$

Unification. $t_1 \doteq t_2$: can we find $\theta : \mathit{Vars} \mapsto \mathit{Terms}$ such that $\theta t_1 = \theta t_2$?

Matching. up-to-renaming $\alpha t_1 \doteq t_2$

Stellar Resolution

Term unification

First-order terms. $t, u ::= x \mid f(t_1, \dots, t_n)$

Unification. $t_1 \doteq t_2$: can we find $\theta : Vars \longmapsto Terms$ such that $\theta t_1 = \theta t_2$?

Matching. up-to-renaming $\alpha t_1 \doteq t_2$

↳ for $x \doteq f(x) \simeq_\alpha y \doteq f(x)$ we have $\theta = y \longmapsto f(x)$

Stellar Resolution

Stars and constellations

A reformulation of Robinson's first-order clausal resolution (logic programming).

Stellar Resolution

Stars and constellations

A reformulation of Robinson's first-order clausal resolution (logic programming).

Rays (atoms). $r ::= t \mid +c(t_1, \dots, t_n) \mid -c(t_1, \dots, t_n)$ where c is called a "*colour*".

Stellar Resolution

Stars and constellations

A reformulation of Robinson's first-order clausal resolution (logic programming).

Rays (atoms). $r ::= t \mid +c(t_1, \ldots, t_n) \mid -c(t_1, \ldots, t_n)$ where c is called a "*colour*".

Stars (clauses). finite and non-empty multiset $\phi = [r_1, \ldots, r_n]$.

Stellar Resolution

Stars and constellations

A reformulation of Robinson's first-order clausal resolution (logic programming).

Rays (atoms). $r ::= t \mid +c(t_1, \ldots, t_n) \mid -c(t_1, \ldots, t_n)$ where c is called a "*colour*".

Stars (clauses). finite and non-empty multiset $\phi = [r_1, \ldots, r_n]$.

↳ $[x, +f(z), -g(h(x, y))]$

Stellar Resolution

Stars and constellations

A reformulation of Robinson's first-order clausal resolution (logic programming).

Rays (atoms). $r ::= t \mid +c(t_1, ..., t_n) \mid -c(t_1, ..., t_n)$ where c is called a "*colour*".

Stars (clauses). finite and non-empty multiset $\phi = [r_1, ..., r_n]$.

↳ $[x, +f(z), -g(h(x, y))]$

Constellations (programs). multiset $\Phi = \phi_1 + ... + \phi_m + ...$ (the variables are locals).

Stellar Resolution

Stars and constellations

A reformulation of Robinson's first-order clausal resolution (logic programming).

Rays (atoms). $r ::= t \mid +c(t_1, \ldots, t_n) \mid -c(t_1, \ldots, t_n)$ where c is called a "*colour*".

Stars (clauses). finite and non-empty multiset $\phi = [r_1, \ldots, r_n]$.

↳ $[x, +f(z), -g(h(x, y))]$

Constellations (programs). multiset $\Phi = \phi_1 + \ldots + \phi_m + \ldots$ (the variables are locals).

↳ $[+add(0, y, y)] + [+add(s(x), y, s(z)), -add(x, y, z)]$

Stellar Resolution

Stars and constellations

A reformulation of Robinson's first-order clausal resolution (logic programming).

Rays (atoms). $r ::= t \mid +c(t_1, \ldots, t_n) \mid -c(t_1, \ldots, t_n)$ where c is called a "*colour*".

Stars (clauses). finite and non-empty multiset $\phi = [r_1, \ldots, r_n]$.

↳ $[x, +f(z), -g(h(x, y))]$

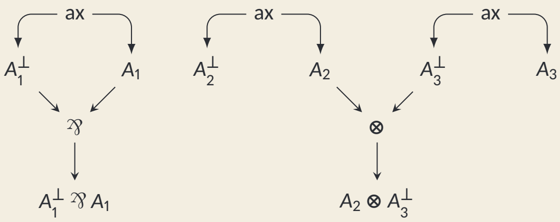
Constellations (programs). multiset $\Phi = \phi_1 + \ldots + \phi_m + \ldots$ (the variables are locals).

↳ $[+add(0, y, y)] + [+add(s(x), y, s(z)), -add(x, y, z)]$

Unlike logic programming : no logic/meaning, no contradiction $\bot$, no goal/query.

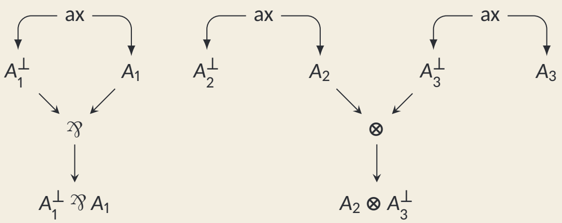
Multiplicative Linear Logic

Interpreting the dynamics of proofs



Multiplicative Linear Logic

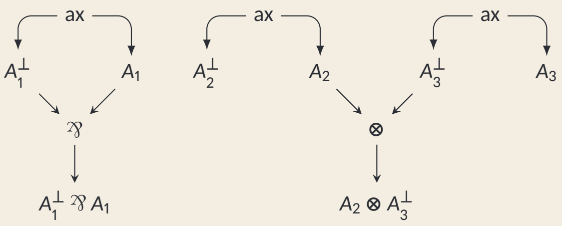
Interpreting the dynamics of proofs



$p_{A_1^\perp ⅋ A_1}(\mathtt{l} \cdot x)$	$p_{A_1^\perp ⅋ A_1}(\mathtt{r} \cdot x)$

Multiplicative Linear Logic

Interpreting the dynamics of proofs

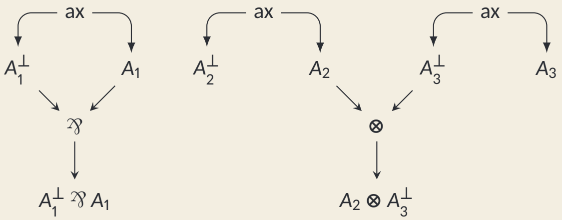


$p_{A_1^\perp ⅋ A_1}(\mathtt{l} \cdot \mathtt{x})$	$p_{A_1^\perp ⅋ A_1}(\mathtt{r} \cdot \mathtt{x})$

$p_{A_2^\perp}(\mathtt{x})$	$p_{A_2 \otimes A_3^\perp}(\mathtt{l} \cdot \mathtt{x})$

Multiplicative Linear Logic

Interpreting the dynamics of proofs



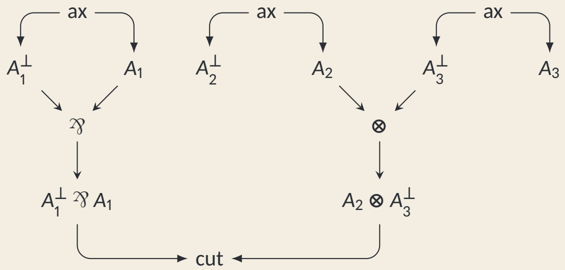
$p_{A_1^\perp \parr A_1}(\mathtt{l} \cdot \mathtt{x})$	$p_{A_1^\perp \parr A_1}(\mathtt{r} \cdot \mathtt{x})$

$p_{A_2^\perp}(x)$	$p_{A_2 \otimes A_3^\perp}(\mathtt{l} \cdot \mathtt{x})$

$p_{A_2 \otimes A_3^\perp}(\mathtt{r} \cdot \mathtt{x})$	$p_{A_3}(x)$

Multiplicative Linear Logic

Interpreting the dynamics of proofs



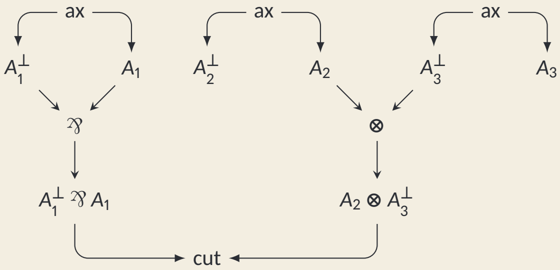
$p_{A_1^\perp \parr A_1}(\mathtt{l} \cdot x)$	$p_{A_1^\perp \parr A_1}(\mathtt{r} \cdot x)$

$p_{A_2^\perp}(x)$	$p_{A_2 \otimes A_3^\perp}(\mathtt{l} \cdot x)$

$p_{A_2 \otimes A_3^\perp}(\mathtt{r} \cdot x)$	$p_{A_3}(x)$

Multiplicative Linear Logic

Interpreting the dynamics of proofs



$+c.p_{A_1^\perp ⅋ A_1}(\mathtt{l}\cdot x)$	$+c.p_{A_1^\perp ⅋ A_1}(\mathtt{r}\cdot x)$

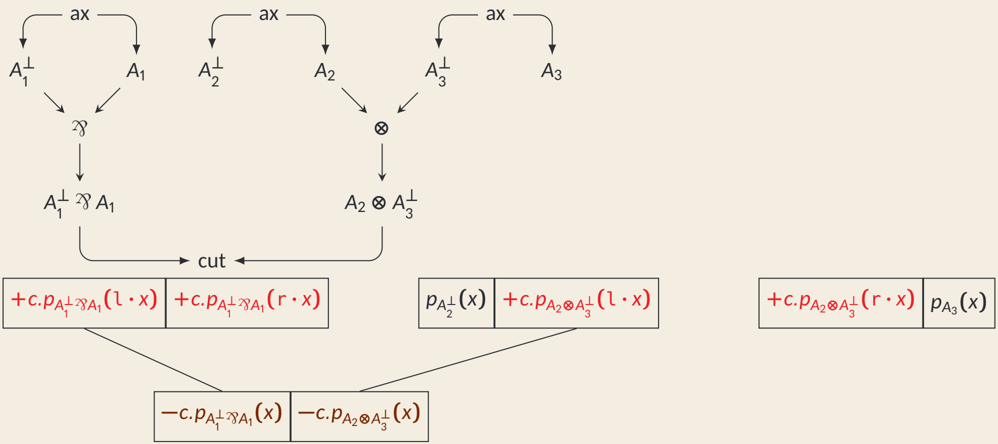
$p_{A_2^\perp}(x)$	$+c.p_{A_2\otimes A_3^\perp}(\mathtt{l}\cdot x)$

$+c.p_{A_2\otimes A_3^\perp}(\mathtt{r}\cdot x)$	$p_{A_3}(x)$

$-c.p_{A_1^\perp ⅋ A_1}(x)$	$-c.p_{A_2\otimes A_3^\perp}(x)$

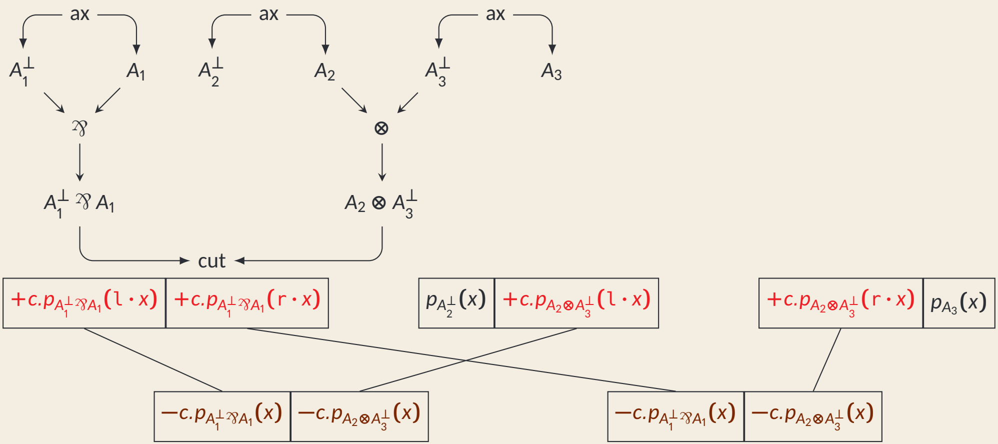
Multiplicative Linear Logic

Interpreting the dynamics of proofs



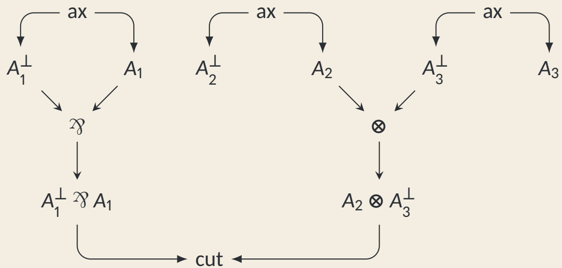
Multiplicative Linear Logic

Interpreting the dynamics of proofs



Multiplicative Linear Logic

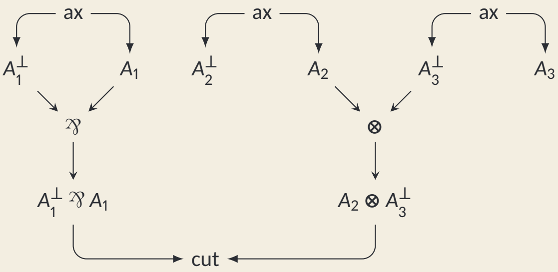
Interpreting the dynamics of proofs



$p_{A_2^\perp}(x)$	$p_{A_3}(x)$

Multiplicative Linear Logic

Interpreting the dynamics of proofs

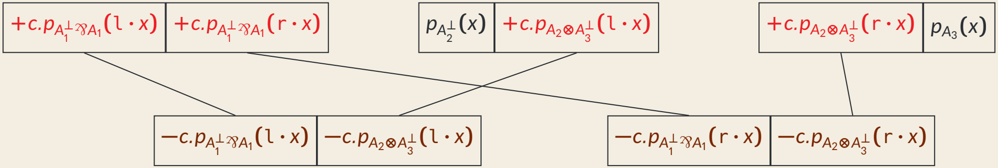


$p_{A_2^\perp}(x)$	$p_{A_3}(x)$

Cut-elimination : resolution of contraints on addresses

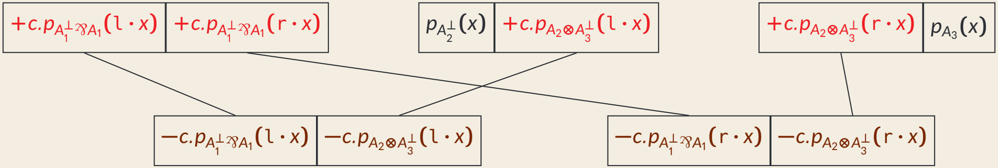
Multiplicative Linear Logic

Liberalisation of proofs



Multiplicative Linear Logic

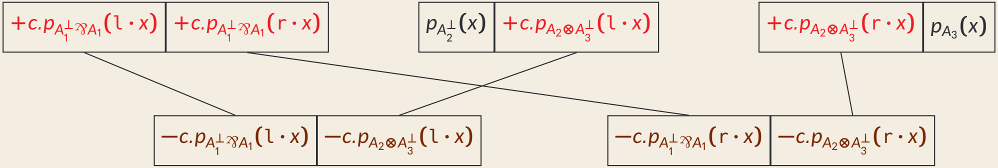
Liberalisation of proofs



- pre-proof of $\vdash A$ $\{[p_A(x)]\}$

Multiplicative Linear Logic

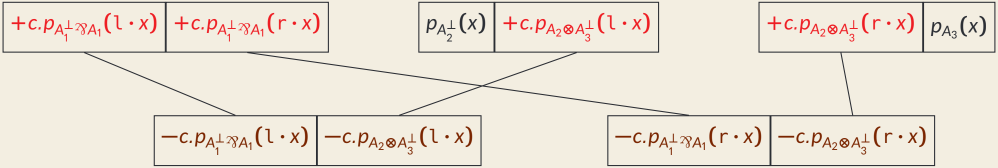
Liberalisation of proofs



- pre-proof of $\vdash A$ $\{[p_A(x)]\}$
- n-ary axioms $\{[p_{A_1}(t_1), \ldots, p_{A_n}(t_n)]\}$

Multiplicative Linear Logic

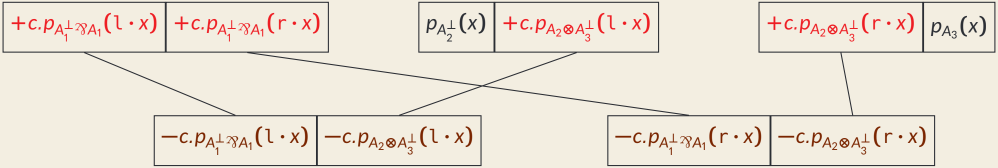
Liberalisation of proofs



- pre-proof of $\vdash A$ $\{[p_A(x)]\}$
- n-ary axioms $\{[p_{A_1}(t_1), \dots, p_{A_n}(t_n)]\}$
- standalone link $A \otimes B$ $[p_A(x)] + [p_B(x)]$

Multiplicative Linear Logic

Liberalisation of proofs

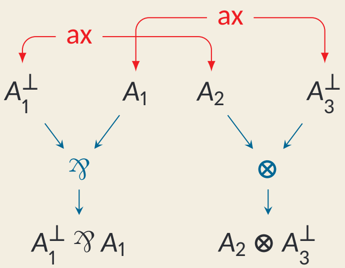


- pre-proof of $\vdash A$ $\{[p_A(x)]\}$
- n-ary axioms $\{[p_{A_1}(t_1), \ldots, p_{A_n}(t_n)]\}$
- standalone link $A \otimes B$ $[p_A(x)] + [p_B(x)]$

Generalises permutations but also partitions [Acclavio, Maieli]

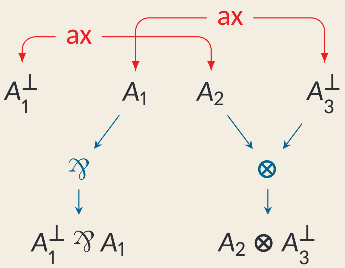
Multiplicative Linear Logic

Girard's factory : vehicle and tests



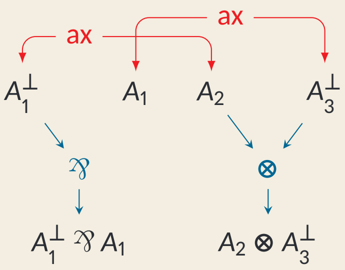
Multiplicative Linear Logic

Girard's factory : vehicle and tests



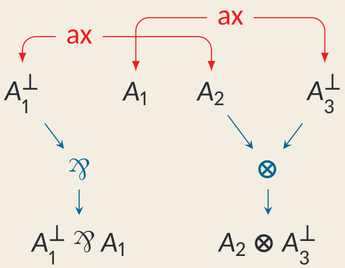
Multiplicative Linear Logic

Girard's factory : vehicle and tests



Multiplicative Linear Logic

Girard's factory : vehicle and tests



Danos-Regnier correctness $\longrightarrow$ Vehicle + Test = certification of proof-net

Multiplicative Linear Logic

Girard's factory : vehicle and tests

$+t.p_{A\otimes B}(\mathtt{l}\cdot x)$	$+t.p_{A^{\perp}⅋B^{\perp}}(\mathtt{l}\cdot x)$

$+t.p_{A\otimes B}(\mathtt{r}\cdot x)$	$+t.p_{A^{\perp}⅋B^{\perp}}(\mathtt{r}\cdot x)$

Multiplicative Linear Logic

Girard's factory : vehicle and tests

$\boxed{+t.p_{A\otimes B}(l\cdot x)}\boxed{+t.p_{A^\perp \parr B^\perp}(l\cdot x)}$ $\boxed{+t.p_{A\otimes B}(r\cdot x)}\boxed{+t.p_{A^\perp \parr B^\perp}(r\cdot x)}$

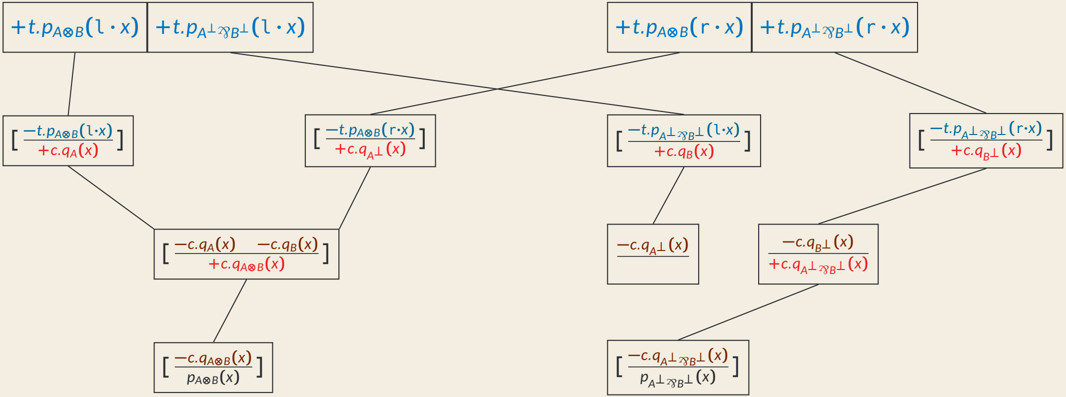
$\boxed{\left[\frac{-t.p_{A\otimes B}(l\cdot x)}{+c.q_A(x)}\right]}$ $\boxed{\left[\frac{-t.p_{A\otimes B}(r\cdot x)}{+c.q_{A^\perp}(x)}\right]}$ $\boxed{\left[\frac{-t.p_{A^\perp \parr B^\perp}(l\cdot x)}{+c.q_B(x)}\right]}$ $\boxed{\left[\frac{-t.p_{A^\perp \parr B^\perp}(r\cdot x)}{+c.q_{B^\perp}(x)}\right]}$

$\boxed{\left[\frac{-c.q_A(x) \quad -c.q_B(x)}{+c.q_{A\otimes B}(x)}\right]}$ $\boxed{\frac{-c.q_{A^\perp}(x)}{}}$ $\boxed{\frac{-c.q_{B^\perp}(x)}{+c.q_{A^\perp \parr B^\perp}(x)}}$

$\boxed{\left[\frac{-c.q_{A\otimes B}(x)}{p_{A\otimes B}(x)}\right]}$ $\boxed{\left[\frac{-c.q_{A^\perp \parr B^\perp}(x)}{p_{A^\perp \parr B^\perp}(x)}\right]}$

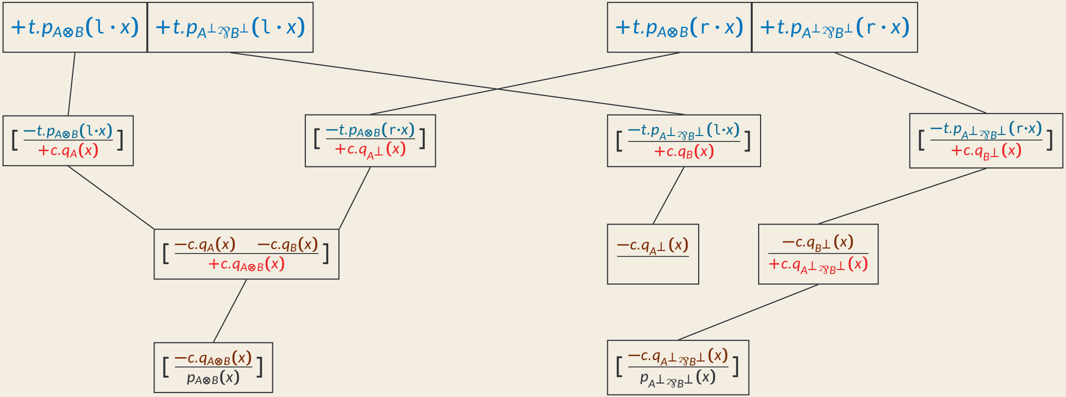
Multiplicative Linear Logic

Girard's factory : vehicle and tests



Multiplicative Linear Logic

Girard's factory : vehicle and tests



correct iff for all test Φ_T we have $\mathrm{Ex}(\Phi_V \uplus \Phi_T) = [p_{A_1}(x), \ldots, p_{A_n}(x)]$.

Multiplicative Linear Logic

Testing and typing

Similar to testing in programming but with finitely many tests. $\Phi, \Phi' : \mathrm{Ex}(\Phi \uplus \Phi')$?

Multiplicative Linear Logic

Testing and typing

Similar to testing in programming but with finitely many tests. $\Phi, \Phi' : \mathrm{Ex}(\Phi \uplus \Phi')$?

Multiplicative Linear Logic

Testing and typing

Similar to testing in programming but with finitely many tests. $\Phi, \Phi' : \mathrm{Ex}(\Phi \uplus \Phi')$?

- $\Phi \perp \Phi'$ when $|\mathrm{Ex}(\Phi \uplus \Phi')| < \infty$: MLL+MIX (acyclic tests).

Multiplicative Linear Logic

Testing and typing

Similar to testing in programming but with finitely many tests. $\Phi, \Phi' : \mathrm{Ex}(\Phi \uplus \Phi')$?

- $\Phi \perp \Phi'$ when $|\mathrm{Ex}(\Phi \uplus \Phi')| < \infty$: MLL+MIX (acyclic tests).
- $\Phi \perp \Phi'$ when $|\mathrm{Ex}(\Phi \uplus \Phi')| = 1$: MLL (acyclic and connected tests).

Multiplicative Linear Logic

Testing and typing

Similar to testing in programming but with finitely many tests. $\Phi, \Phi' : \text{Ex}(\Phi \uplus \Phi')$?

- $\Phi \perp \Phi'$ when $|\text{Ex}(\Phi \uplus \Phi')| < \infty$: MLL+MIX (acyclic tests).
- $\Phi \perp \Phi'$ when $|\text{Ex}(\Phi \uplus \Phi')| = 1$: MLL (acyclic and connected tests).

We use realisability techniques (as in Ludics). From a chosen $\perp$:

Multiplicative Linear Logic

Testing and typing

Similar to testing in programming but with finitely many tests. $\Phi, \Phi' : \text{Ex}(\Phi \uplus \Phi')$?

- $\Phi \perp \Phi'$ when $|\text{Ex}(\Phi \uplus \Phi')| < \infty$: MLL+MIX (acyclic tests).
- $\Phi \perp \Phi'$ when $|\text{Ex}(\Phi \uplus \Phi')| = 1$: MLL (acyclic and connected tests).

We use realisability techniques (as in Ludics). From a chosen $\perp$:

- pre-type **A** : set of constellations.

Multiplicative Linear Logic

Testing and typing

Similar to testing in programming but with finitely many tests. $\Phi, \Phi' : \mathrm{Ex}(\Phi \uplus \Phi')$?

- $\Phi \perp \Phi'$ when $|\mathrm{Ex}(\Phi \uplus \Phi')| < \infty$: MLL+MIX (acyclic tests).
- $\Phi \perp \Phi'$ when $|\mathrm{Ex}(\Phi \uplus \Phi')| = 1$: MLL (acyclic and connected tests).

We use realisability techniques (as in Ludics). From a chosen $\perp$:

- pre-type $\mathbf{A}$: set of constellations.
- linear negation $\sim \mathbf{A} := \mathbf{A}^{\perp} := \{\Phi' \mid \forall \Phi \in \mathbf{A}, \Phi \perp \Phi'\}$.

Multiplicative Linear Logic

Testing and typing

Similar to testing in programming but with finitely many tests. $\Phi, \Phi' : \mathrm{Ex}(\Phi \uplus \Phi')$?

- $\Phi \perp \Phi'$ when $|\mathrm{Ex}(\Phi \uplus \Phi')| < \infty$: MLL+MIX (acyclic tests).
- $\Phi \perp \Phi'$ when $|\mathrm{Ex}(\Phi \uplus \Phi')| = 1$: MLL (acyclic and connected tests).

We use realisability techniques (as in Ludics). From a chosen $\perp$:

- pre-type $\mathbf{A}$: set of constellations.
- linear negation $\sim \mathbf{A} := \mathbf{A}^{\perp} := \{\Phi' \mid \forall \Phi \in \mathbf{A}, \Phi \perp \Phi'\}$.
- type : $\mathbf{A} = \mathbf{A}^{\perp\perp}$.

Multiplicative Linear Logic

Testing and typing

Similar to testing in programming but with finitely many tests. $\Phi, \Phi' : \mathrm{Ex}(\Phi \uplus \Phi')$?

- $\Phi \perp \Phi'$ when $|\mathrm{Ex}(\Phi \uplus \Phi')| < \infty$: MLL+MIX (acyclic tests).
- $\Phi \perp \Phi'$ when $|\mathrm{Ex}(\Phi \uplus \Phi')| = 1$: MLL (acyclic and connected tests).

We use realisability techniques (as in Ludics). From a chosen $\perp$:

- pre-type $\mathbf{A}$: set of constellations.
- linear negation $\sim \mathbf{A} := \mathbf{A}^{\perp} := \{\Phi' \mid \forall \Phi \in \mathbf{A}, \Phi \perp \Phi'\}$.
- type : $\mathbf{A} = \mathbf{A}^{\perp\perp}$.
- tensor : $\mathbf{A} \otimes \mathbf{B} = \{\Phi_A \uplus \Phi_B \mid \Phi_A \in \mathbf{A}, \Phi_B \in \mathbf{B}\}^{\perp\perp}$ when $\mathbf{A}, \mathbf{B}$ not matchable.

Multiplicative Linear Logic

Testing and typing

Similar to testing in programming but with finitely many tests. $\Phi, \Phi' : \mathrm{Ex}(\Phi \uplus \Phi')$?

- $\Phi \perp \Phi'$ when $|\mathrm{Ex}(\Phi \uplus \Phi')| < \infty$: MLL+MIX (acyclic tests).
- $\Phi \perp \Phi'$ when $|\mathrm{Ex}(\Phi \uplus \Phi')| = 1$: MLL (acyclic and connected tests).

We use realisability techniques (as in Ludics). From a chosen $\perp$:

- pre-type $\mathbf{A}$: set of constellations.
- linear negation $\sim \mathbf{A} := \mathbf{A}^{\perp} := \{\Phi' \mid \forall \Phi \in \mathbf{A}, \Phi \perp \Phi'\}$.
- type : $\mathbf{A} = \mathbf{A}^{\perp\perp}$.
- tensor : $\mathbf{A} \otimes \mathbf{B} = \{\Phi_A \uplus \Phi_B \mid \Phi_A \in \mathbf{A}, \Phi_B \in \mathbf{B}\}^{\perp\perp}$ when $\mathbf{A}, \mathbf{B}$ not matchable.
- Types as descriptions of computation, not contraints.

Conclusion

Other linear logic fragments

Conclusion

Other linear logic fragments

- exponentials (IMELL) : work in progress

Conclusion

Other linear logic fragments

- exponentials (IMELL) : work in progress
- additives, neutrals, full exponentials : handled in second order

Conclusion

Other linear logic fragments

- exponentials (IMELL) : work in progress
- additives, neutrals, full exponentials : handled in second order
- first-order : internal colours + individuals $\forall x.A$ as multiplicatives.

Conclusion

Other linear logic fragments

- exponentials (IMELL) : work in progress
- additives, neutrals, full exponentials : handled in second order
- first-order : internal colours + individuals $\forall x.A$ as multiplicatives.

Natural encoding of several models :

Conclusion

Other linear logic fragments

- exponentials (IMELL) : work in progress
- additives, neutrals, full exponentials : handled in second order
- first-order : internal colours + individuals $\forall x.A$ as multiplicatives.

Natural encoding of several models :

- λ-calculus, logic programming (disjunctive clauses)

Conclusion

Other linear logic fragments

- exponentials (IMELL) : work in progress
- additives, neutrals, full exponentials : handled in second order
- first-order : internal colours + individuals $\forall x.A$ as multiplicatives.

Natural encoding of several models :

- λ-calculus, logic programming (disjunctive clauses)
  ↳ logico-functional space ?

Conclusion

Other linear logic fragments

- exponentials (IMELL) : work in progress
- additives, neutrals, full exponentials : handled in second order
- first-order : internal colours + individuals $\forall x.A$ as multiplicatives.

Natural encoding of several models :

- λ-calculus, logic programming (disjunctive clauses)
  ↳ logico-functional space ?
- Wang's tiles, abstract tile assembly model (aTAM) used in DNA computing

Conclusion

Other linear logic fragments

- exponentials (IMELL) : work in progress
- additives, neutrals, full exponentials : handled in second order
- first-order : internal colours + individuals $\forall x.A$ as multiplicatives.

Natural encoding of several models :

- λ-calculus, logic programming (disjunctive clauses)
  ↳ logico-functional space ?
- Wang's tiles, abstract tile assembly model (aTAM) used in DNA computing
  ↳ cyclic (grid) diagrams