A small step towards an Open Source Logic: Studying Models of Computation for Linear Realisability

Valentin Maestracci

Aix-Marseille Université

6th November 2025

Basics of Proofs

Logic is about *reasoning* on sentences.

Basics of Proofs

Logic is about *reasoning* on sentences.

Fluffy $\longrightarrow$ 

Basics of Proofs

Logic is about *reasoning* on sentences.

Fluffy $\longrightarrow$ 

If I know that:

- $A :=$ "Fluffy is Red " is true.
- $A \Rightarrow B :=$ "IF Fluffy is Red THEN Fluffy is Cute" is true.

Conclude $B :=$ "Fluffy is cute" is true. (*whatever the sentence were*)

Basics of Proofs

Logic is about *reasoning* on sentences.

Fluffy $\longrightarrow$ 

If I know that:

- $A :=$ "Fluffy is Red " is true.
- $A \Rightarrow B :=$ "IF Fluffy is Red THEN Fluffy is Cute" is true.

Conclude $B :=$ "Fluffy is cute" is true. (*whatever the sentence were*)

Logicians write this as follows:

$$\frac{\vdash A \qquad \vdash A \Rightarrow B}{\vdash B}\ \text{modus ponens}$$

Basics of Programming

A program is usually a sequential list of instructions:

1. Take the chocolate
2. Grind it
3. Add it to the batter
4. Cook it
5. You get: A Cake!

Basics of Programming

A program is usually a sequential list of instructions:

1. Take the chocolate
2. Grind it
3. Add it to the batter
4. Cook it
5. You get: A Cake!
6. Take the chocolate
7. Grind it
8. Add it to the batter
9. Cook it
10. You get: A Cake!

Basics of Programming

A program is usually a sequential list of instructions:

1. Take the chocolate
2. Grind it
3. Add it to the batter
4. Cook it
5. You get: A Cake!
6. Take the chocolate
7. Grind it
8. Add it to the batter
9. Cook it
10. You get: A Cake!
11. Take the chocolate
12. Grind it

Avoid repeating ourselves: group instructions $\Rightarrow$ functions (recipes).

Function *bake*(∗*thing*∗ : *Ingredient*) $\Rightarrow$ *Cake*:

1. Take the ∗*thing*∗
2. Grind it
3. Add it to the batter
4. Cook it
5. You get: A Cake!

Avoid repeating ourselves: group instructions $\Rightarrow$ functions (recipes).

Function *bake*(∗*thing*∗ : *Ingredient*) $\Rightarrow$ *Cake*:

1. Take the ∗*thing*∗
2. Grind it
3. Add it to the batter
4. Cook it
5. You get: A Cake!

And then we use the function as a new instruction:

1. *bake*(*chocolate*) : *Cake*
2. *bake*(*chocolate*) : *Cake*
3. *bake*(*vanilla*) : *Cake*

(Typing is optional)

Basics of Programming

Programs are about *using functions*.

If I have an object

- $\mathit{thing} : \mathit{Ingredient}$.
- A function (recipe) $\mathit{recipe\_function} : \mathit{Ingredient} \Rightarrow \mathit{Cake}$

Then I can create a Cake.

Computer Scientists write this as follows:

$$\frac{\vdash \mathit{thing} : \mathit{Ingredient} \qquad \vdash \mathit{recipe\_function} : \mathit{Ingredient} \Rightarrow \mathit{Cake}}{\vdash \mathit{recipe\_function}(\mathit{thing}) : \mathit{Cake}}\ \text{app}$$

The proof/program correspondence

$$\frac{\vdash A \qquad \vdash A \Rightarrow B}{\vdash B}\ \text{modus ponens}$$

$$\frac{\vdash \mathit{thing} : \mathit{Ingredient} \qquad \vdash \mathit{recipe\_function} : \mathit{Ingredient} \Rightarrow \mathit{Cake}}{\vdash \mathit{recipe\_function}(\mathit{thing}) : \mathit{Cake}}\ \text{app}$$

The proof/program correspondence

$$\frac{\vdash A \qquad \vdash A \Rightarrow B}{\vdash B}\ \text{modus ponens}$$

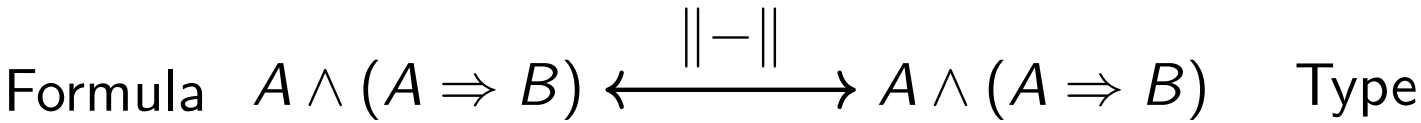
$$\frac{\vdash \mathit{thing} : \mathit{Ingredient} \qquad \vdash \mathit{recipe\_function} : \mathit{Ingredient} \Rightarrow \mathit{Cake}}{\vdash \mathit{recipe\_function}(\mathit{thing}) : \mathit{Cake}}\ \text{app}$$

Oh my god these are the same things!

Simply Typed λ-calculus
$\Leftrightarrow$ Minimal Implicative Logic

Syntactic World

Proofs **Programs**



Syntactic World

Proofs **Programs**

$$\text{Formula} \quad A \wedge (A \Rightarrow B) \overset{\|-\|}{\longleftrightarrow} A \wedge (A \Rightarrow B) \quad \text{Type}$$

$$\text{Proof} \overset{\|-\|}{\longleftrightarrow} \text{Program}$$

Syntactic World

Proofs **Programs**

Formula $A \wedge (A \Rightarrow B) \overset{\|-\|}{\longleftrightarrow} A \wedge (A \Rightarrow B)$ Type

Proof $\overset{\|-\|}{\longleftrightarrow}$ Program

Cut-Elimination $\overset{\|-\|}{\longleftrightarrow}$ Computation

Compilation and Semantics

$$\texttt{Code} \xrightarrow{\|-\|} \texttt{Program} \in \textit{MOC}$$

$$\texttt{Type} \xrightarrow{\|-\|} \texttt{Behaviour}$$

Here, we will **relate** Models of Computation (MOC).

Compilation and Semantics

$$\texttt{Code} \xrightarrow{\|-\|} \texttt{Program} \in \mathit{MOC}$$

$$\texttt{Type} \xrightarrow{\|-\|} \texttt{Behaviour}$$

Here, we will **relate** Models of Computation (MOC).

$\Rightarrow$ Need to **preserve dynamics/meaning/behaviours**: $\|\texttt{Code}\colon A\| \in \|A\|$

A quest for the right model of computation

We saw earlier that

$$\text{Simply Typed } \lambda\text{-calculus} \Leftrightarrow \text{Minimal Implicative Logic}$$

$$\text{Simply Typed } \lambda\text{-calculus} \subset \lambda\text{-calculus}$$

A quest for the right model of computation

We saw earlier that

$$\text{Simply Typed } \lambda\text{-calculus} \Leftrightarrow \text{Minimal Implicative Logic}$$

$$\text{Simply Typed } \lambda\text{-calculus} \subset \lambda\text{-calculus}$$

We will now study:

$$\text{Fragment?} \Leftrightarrow \text{Linear Logic}$$

$$\text{Fragment?} \subset \text{Model of Computation?}$$

Intuition for Linear Logic

$$\mathit{Chocolate} \wedge (\mathit{Chocolate} \Rightarrow \mathit{Cake}) \vdash \mathit{Cake} \wedge \mathit{Cake}$$

Intuition for Linear Logic

$$Chocolate \wedge (Chocolate \Rightarrow Cake) \vdash Cake \wedge Cake$$

Linear Logic is a "resource-aware" logic:

- $Chocolate \otimes (Chocolate \multimap Cake) \not\vdash Cake \otimes Cake$
- $Chocolate \otimes (Chocolate \multimap Cake) \vdash Cake$

$\wedge$ splits: $\otimes$, &

Intuition for Linear Logic

$$Chocolate \wedge (Chocolate \Rightarrow Cake) \vdash Cake \wedge Cake$$

Linear Logic is a "resource-aware" logic:

- $Chocolate \otimes (Chocolate \multimap Cake) \not\vdash Cake \otimes Cake$
- $Chocolate \otimes (Chocolate \multimap Cake) \vdash Cake$

$\wedge$ splits: $\otimes$, &

Describe a shop: You have 5€

Intuition for Linear Logic

$$\mathit{Chocolate} \wedge (\mathit{Chocolate} \Rightarrow \mathit{Cake}) \vdash \mathit{Cake} \wedge \mathit{Cake}$$

Linear Logic is a "resource-aware" logic:

- $\mathit{Chocolate} \otimes (\mathit{Chocolate} \multimap \mathit{Cake}) \not\vdash \mathit{Cake} \otimes \mathit{Cake}$
- $\mathit{Chocolate} \otimes (\mathit{Chocolate} \multimap \mathit{Cake}) \vdash \mathit{Cake}$

$\wedge$ splits: $\otimes$, &

Describe a shop: You have 5€

- Multiplicative: $5€ \multimap \mathit{Butter} \otimes 5€$

Intuition for Linear Logic

$$\mathit{Chocolate} \wedge (\mathit{Chocolate} \Rightarrow \mathit{Cake}) \vdash \mathit{Cake} \wedge \mathit{Cake}$$

Linear Logic is a "resource-aware" logic:

- $\mathit{Chocolate} \otimes (\mathit{Chocolate} \multimap \mathit{Cake}) \not\vdash \mathit{Cake} \otimes \mathit{Cake}$
- $\mathit{Chocolate} \otimes (\mathit{Chocolate} \multimap \mathit{Cake}) \vdash \mathit{Cake}$

$\wedge$ splits: $\otimes$, &

Describe a shop: You have 5€

- Multiplicative: $5€ \multimap \mathit{Butter} \otimes 5€$ $(\times)$ (Free Butter)

Intuition for Linear Logic

$$\mathit{Chocolate} \wedge (\mathit{Chocolate} \Rightarrow \mathit{Cake}) \vdash \mathit{Cake} \wedge \mathit{Cake}$$

Linear Logic is a "resource-aware" logic:

- $\mathit{Chocolate} \otimes (\mathit{Chocolate} \multimap \mathit{Cake}) \not\vdash \mathit{Cake} \otimes \mathit{Cake}$
- $\mathit{Chocolate} \otimes (\mathit{Chocolate} \multimap \mathit{Cake}) \vdash \mathit{Cake}$

$\wedge$ splits: $\otimes$, &

Describe a shop: You have 5€

- Multiplicative: $5€ \multimap \mathit{Butter} \otimes 5€$ $(\times)$ (Free Butter)
- Additive: $5€ \multimap \mathit{Butter} \,\&\, 5€$

Intuition for Linear Logic

$$\mathit{Chocolate} \wedge (\mathit{Chocolate} \Rightarrow \mathit{Cake}) \vdash \mathit{Cake} \wedge \mathit{Cake}$$

Linear Logic is a "resource-aware" logic:

- $\mathit{Chocolate} \otimes (\mathit{Chocolate} \multimap \mathit{Cake}) \not\vdash \mathit{Cake} \otimes \mathit{Cake}$
- $\mathit{Chocolate} \otimes (\mathit{Chocolate} \multimap \mathit{Cake}) \vdash \mathit{Cake}$

$\wedge$ splits: $\otimes$, &

Describe a shop: You have 5€

- Multiplicative: $5€ \multimap \mathit{Butter} \otimes 5€$ ($\times$) (Free Butter)
- Additive: $5€ \multimap \mathit{Butter} \,\&\, 5€$ (✓) (Famous French Saying)
- Additive: $5€ \multimap \mathit{Cake} \,\&\, (\mathit{Eat\ it\ too})$ (English Equivalent)

The operators of Linear Logic

- Multiplicatives: (And) $\otimes$, (Or) $\parr$
- Additives: (And) &, (Or) $\oplus$
- Exponentials: !, ?

The operators of Linear Logic

- Multiplicatives: (And) $\otimes$, (Or) $\parr$
- Additives: (And) &, (Or) $\oplus$
- Exponentials: !, ?

Exponential $!A =$ has many A as you want.

Real Shop: $!(5€ \multimap \textit{Butter} \,\&\, 5€)$ (In Math, every A is $!A$)

The operators of Linear Logic

- Multiplicatives: (And) $\otimes$, (Or) $\parr$
- Additives: (And) &, (Or) $\oplus$
- Exponentials: !, ?

Exponential $!A =$ has many A as you want.

Real Shop: $!(5€ \multimap \mathit{Butter} \,\&\, 5€)$ (In Math, every A is $!A$)

Seely Isomorphism: $!(S \,\&\, M) \simeq !S \otimes !M$

(Tito's observation: plural of "Madam, Sir" is "Ladies AND Gentleman")

Sequent-Calculus: (Multiplicative) Linear Logic

$$\frac{}{\vdash A, A^{\perp}}\ \text{ax} \qquad \frac{\vdash \Gamma, A \quad \vdash \Delta, A^{\perp}}{\vdash \Gamma, \Delta}\ \text{cut}$$

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

$$\dfrac{\dfrac{}{\vdash A^{\perp}, A}\ \text{ax} \qquad \dfrac{}{\vdash B^{\perp}, B}\ \text{ax}}{\vdash A^{\perp}, B^{\perp}, (A \otimes B)}\ \otimes \Bigg/ \parr$$

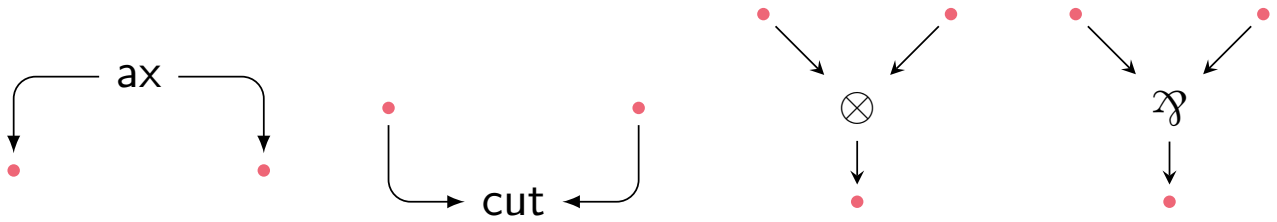
$$\vdash A^{\perp} \parr B^{\perp}, (A \otimes B)$$

Our second Model of Computation: *MLL* + cut-elimination.

(Interesting, quite basic)

Proof Structure: another Model of Computation

$$\frac{}{\vdash A, A^{\perp}}\ \text{ax} \qquad \frac{\vdash \Gamma, A \quad \vdash \Delta, A^{\perp}}{\vdash \Gamma, \Delta}\ \text{cut} \qquad \frac{\vdash \Gamma, A \quad \vdash \Delta, B}{\vdash \Gamma, \Delta, A \otimes B}\ \otimes \qquad \frac{\vdash \Gamma, A, B}{\vdash \Gamma, A \parr B}\ \parr$$



Axiom Cut Tensor Par

Define compilation $\|-\|$: **Proof Sequent** $\longrightarrow$ **Proof Structure**

Correctness Criteria

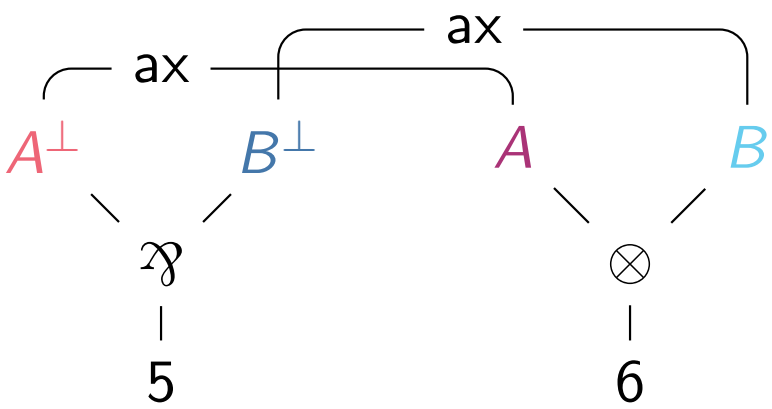
Correctness Criteria

$\Rightarrow$ Correctness Criteria (Tests): Long Trip, Danos-Regnier etc...

Theorem (Sequentialisation)

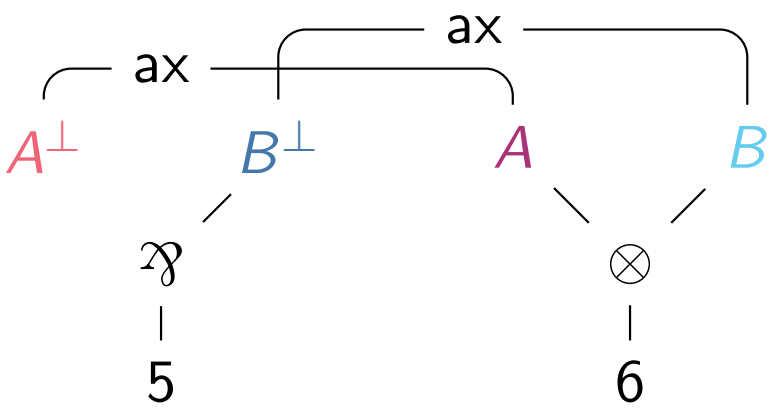
If $\mathcal{S}$ passes the tests then $\exists \pi$ proof such that $\|\pi\| = \mathcal{S}$.

The Danos-Regnier Criterion



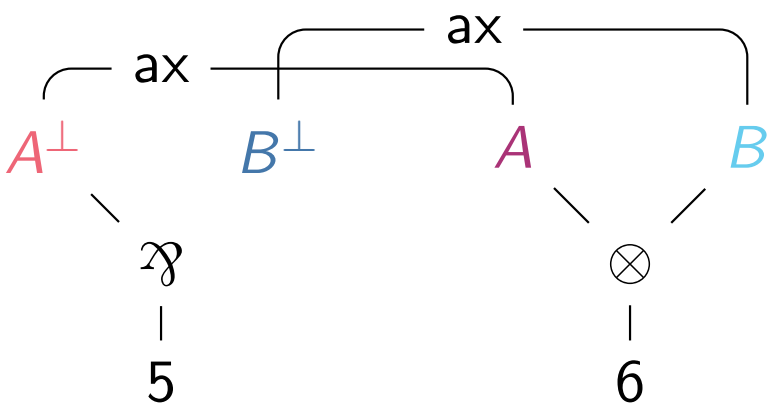
$$\dfrac{\dfrac{}{\vdash A^\perp, A}\,\text{ax} \qquad \dfrac{}{\vdash B^\perp, B}\,\text{ax}}{\dfrac{\vdash A^\perp, B^\perp, (A \otimes B)}{\vdash A^\perp \parr B^\perp, (A \otimes B)}\,\parr}\,\otimes$$

The Danos-Regnier Criterion



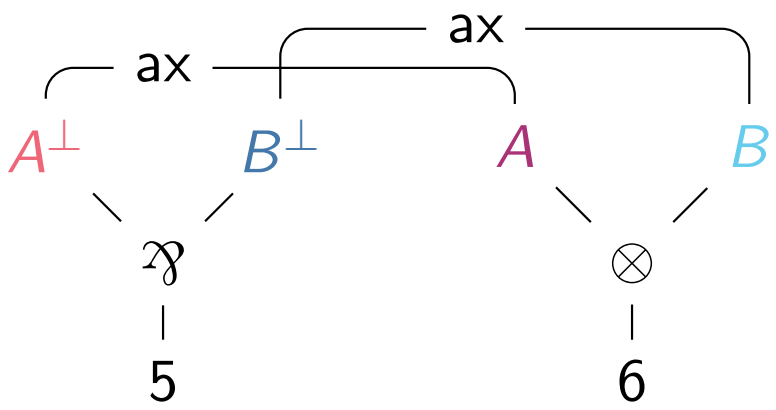
$$\dfrac{\dfrac{}{\vdash A^\perp, A}\,\text{ax} \qquad \dfrac{}{\vdash B^\perp, B}\,\text{ax}}{\dfrac{\vdash A^\perp, B^\perp, (A \otimes B)}{\vdash A^\perp \parr B^\perp, (A \otimes B)}\,\parr}\,\otimes$$

The Danos-Regnier Criterion



$$\dfrac{\dfrac{}{\vdash A^\perp, A}\,\text{ax} \qquad \dfrac{}{\vdash B^\perp, B}\,\text{ax}}{\dfrac{\vdash A^\perp, B^\perp, (A \otimes B)}{\vdash A^\perp \parr B^\perp, (A \otimes B)}\,\parr}\,\otimes$$

The Danos-Regnier Criterion

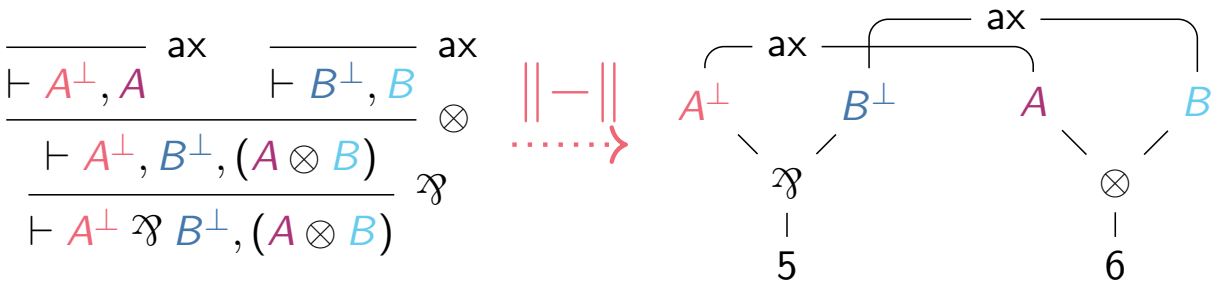


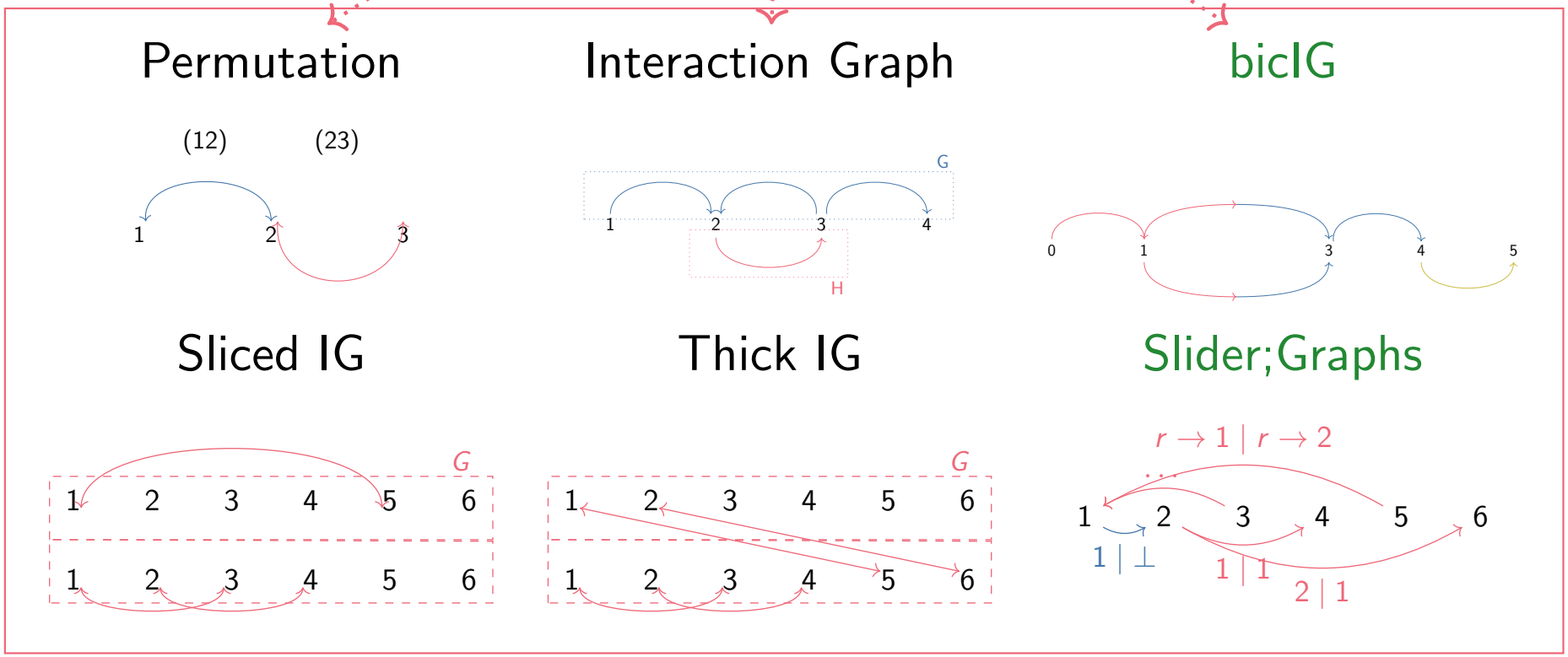
$$\dfrac{\dfrac{\dfrac{}{\vdash A^\perp, A}\,\text{ax} \qquad \dfrac{}{\vdash B^\perp, B}\,\text{ax}}{\vdash A^\perp, B^\perp, (A \otimes B)}\,\otimes}{\vdash A^\perp \parr B^\perp, (A \otimes B)}\,\parr$$

Theorem (Danos-Regnier Criterion)

Proof Structure $\mathcal{S}$ correct
$\Leftrightarrow$ *For all switching φ, $\mathcal{S}^\varphi$ connected + acyclic.*

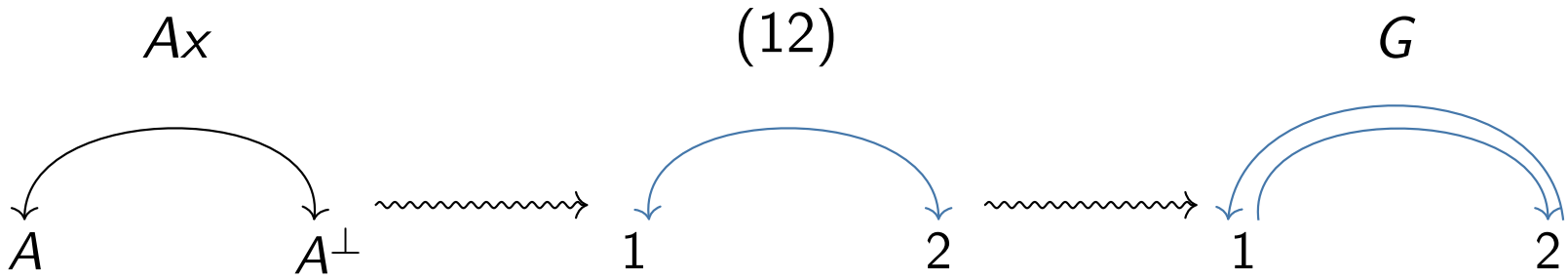
On the Menu Today





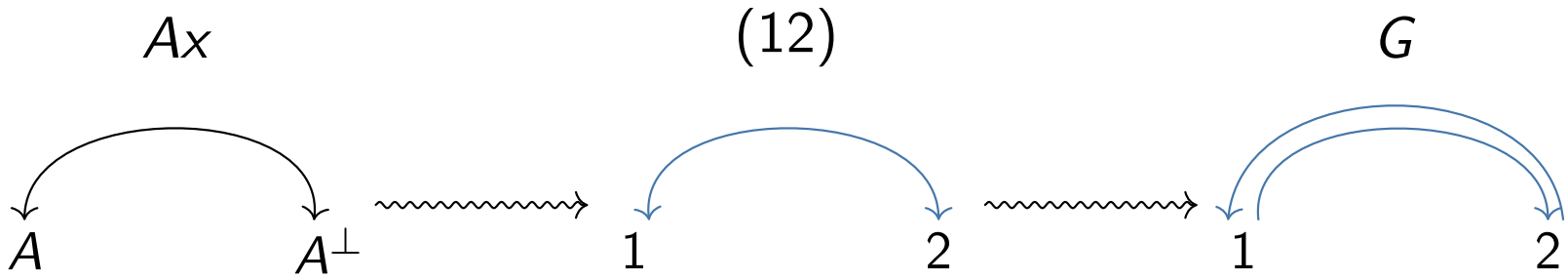
Model of Reference: Seiller's Interaction Graphs

Generalise **Proof Structure** $\longrightarrow$ **Graphs**:

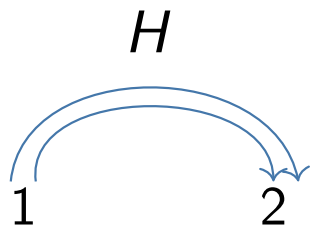


Model of Reference: Seiller's Interaction Graphs

Generalise **Proof Structure** $\longrightarrow$ **Graphs**:



New powers unlocked:



Interaction Graph: the formal definition

Definition (Interaction Graph: Program)

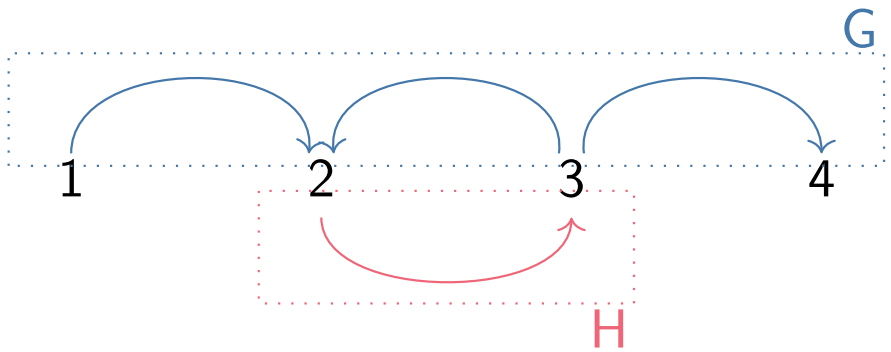
A directed graph G_L:

- Locations L (Vertices)
- Primitives E (Colored Edges)

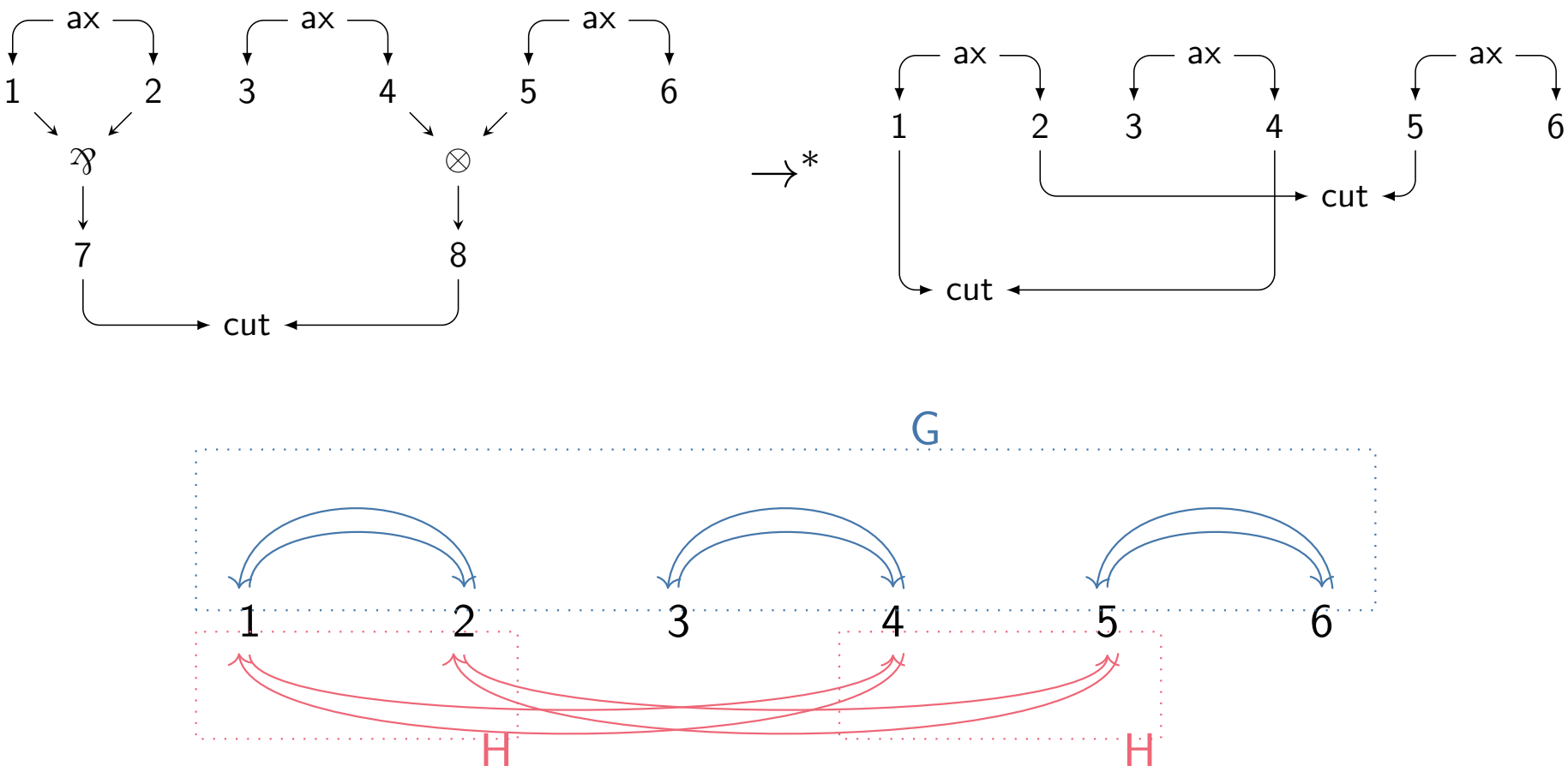
Source/target maps $s, t : E \to L$.

Definition (Paths)

A path in G = sequence of edges $e_1 e_2 \dots e_n$ with $s_G(e_{i+1}) = t_G(e_i)$.
Alternated when colors alternate.



Recap: Translation



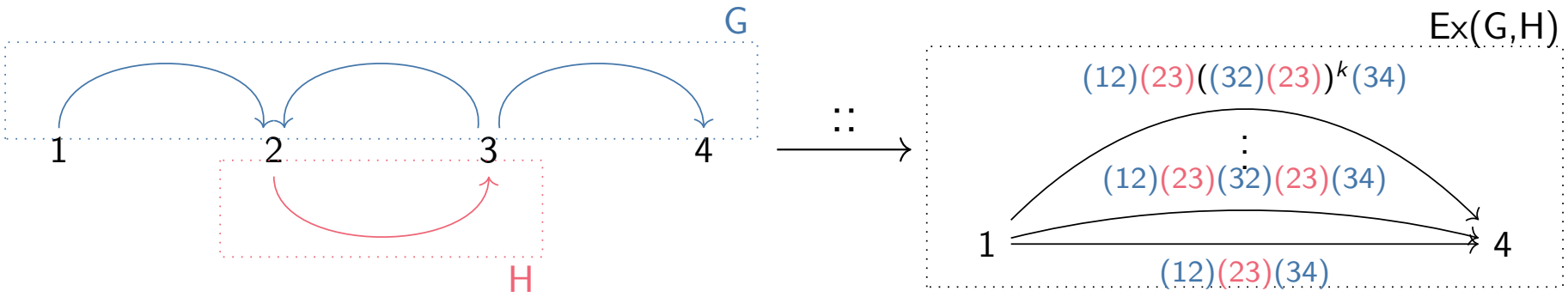
Execution of Interaction Graph

Definition (Binary Execution)

Graph $\mathrm{Ex}(G, H)$ (also $G :: H$):

- Location: $L := L_G \ominus L_H$
- Edges: Alternating paths in $G \leftrightarrows H$ of endpoints in V.

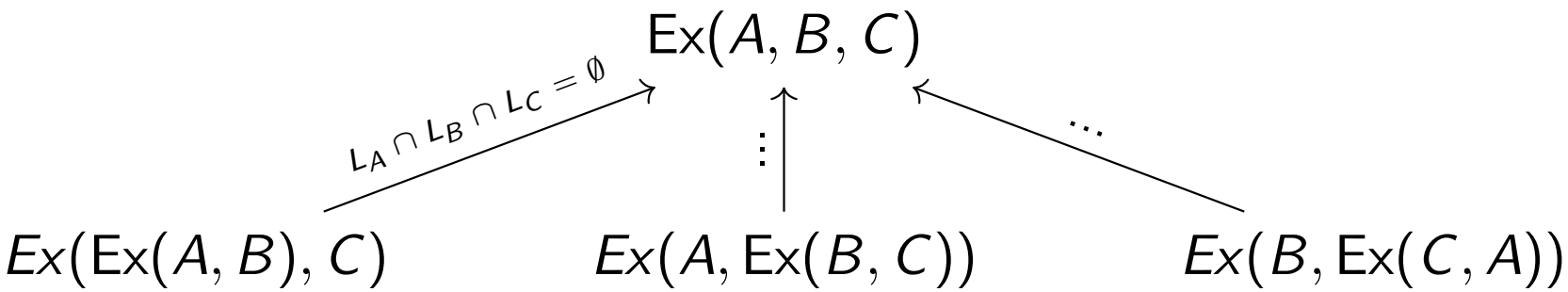
$$\underbrace{\text{Glueing, Saturation}}_{\text{Composition}}\text{, Hiding}$$



Finally a contribution: bicIG

One can make execution *unary*:

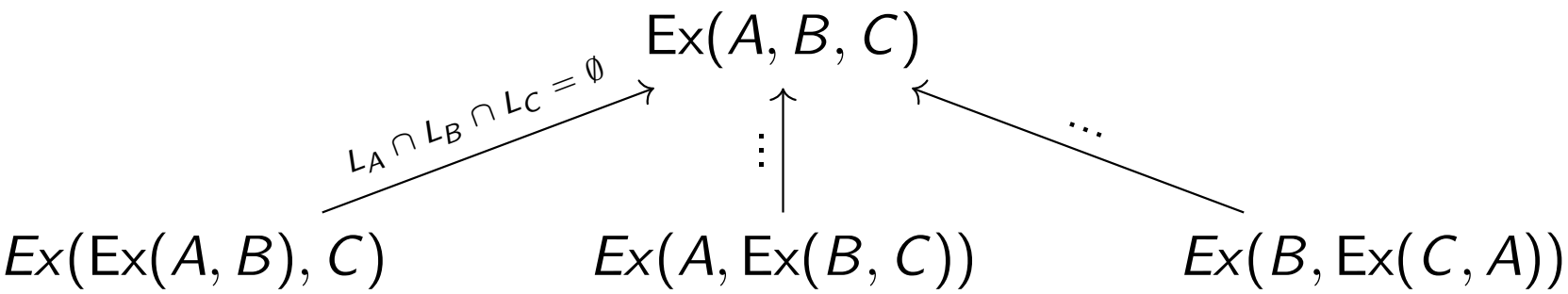
$$A :: B :: C \Rightarrow \mathrm{Ex}_{???}(A \barwedge B \barwedge C)$$

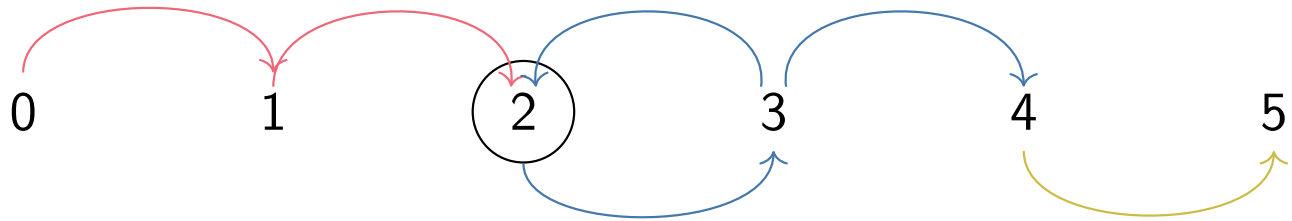


Finally a contribution: bicIG

One can make execution *unary*:

$$A :: B :: C \Rightarrow \mathrm{Ex}_L(A \barwedge B \barwedge C)$$

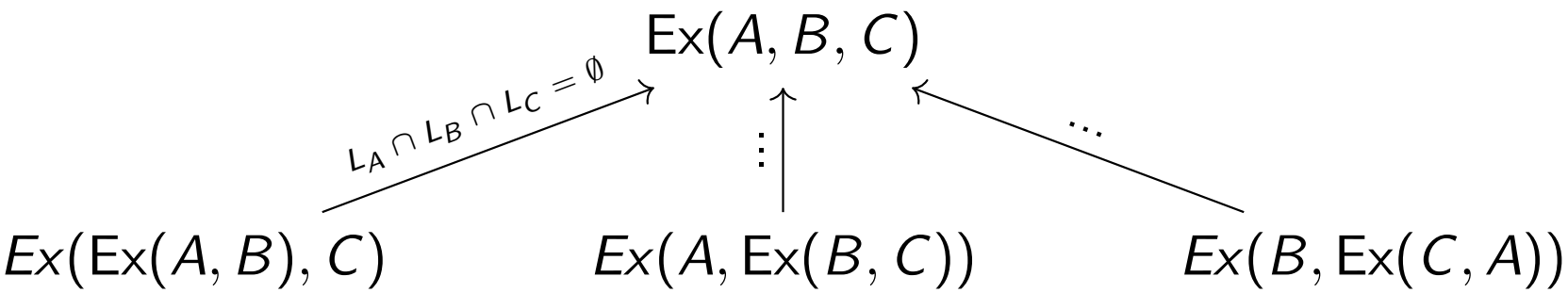


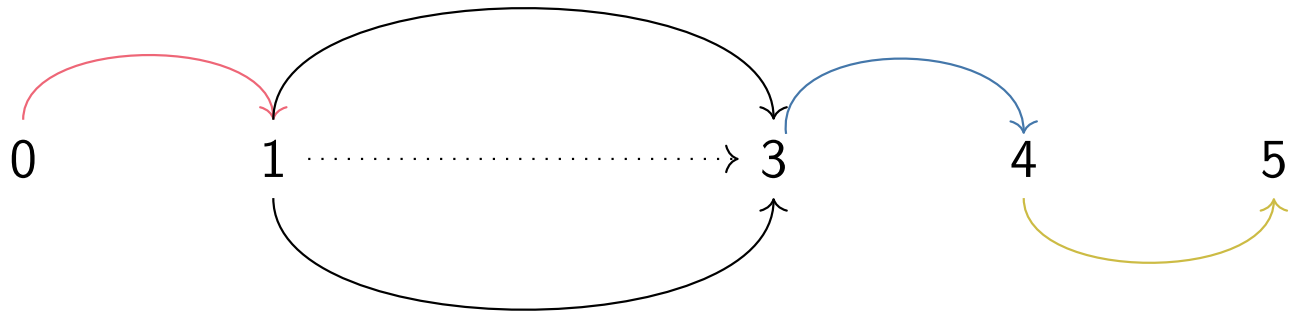


Finally a contribution: bicIG

One can make execution *unary*:

$$A :: B :: C \Rightarrow \mathrm{Ex}(A \between B \between C)$$

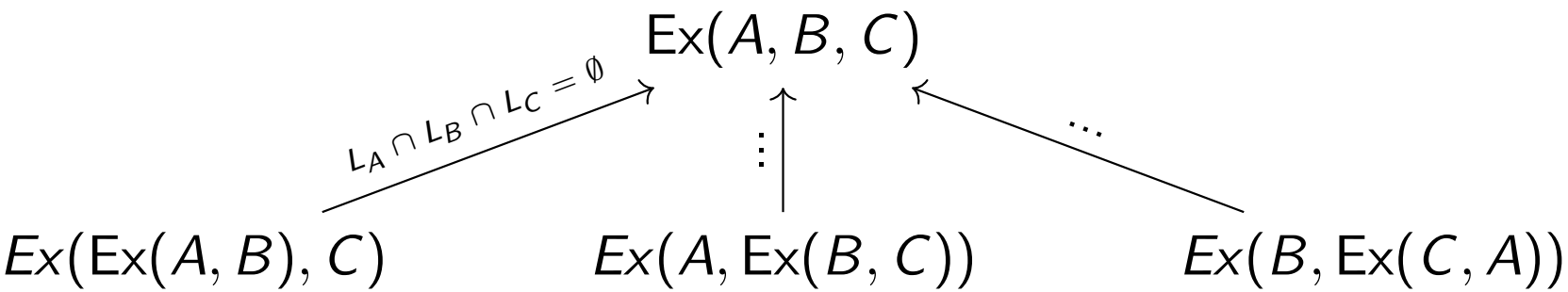


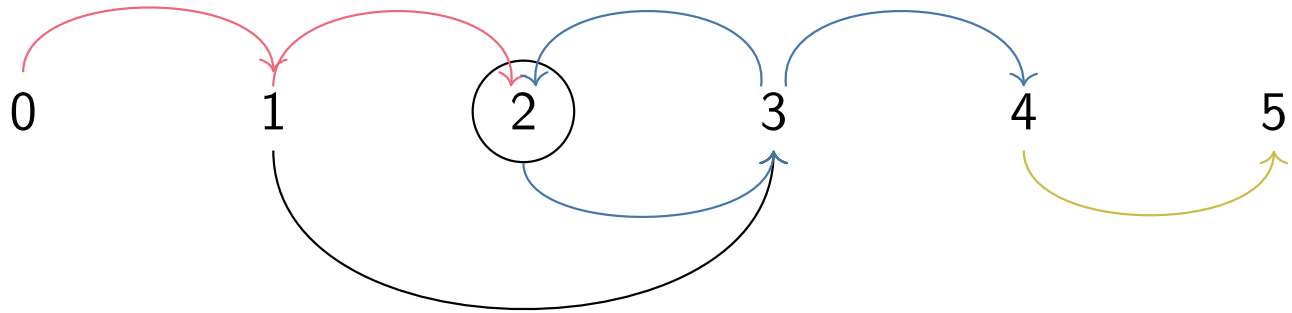


Finally a contribution: bicIG

One can make execution *unary*:

$$A :: B :: C \Rightarrow \mathrm{Ex}(A \between B \between C)$$

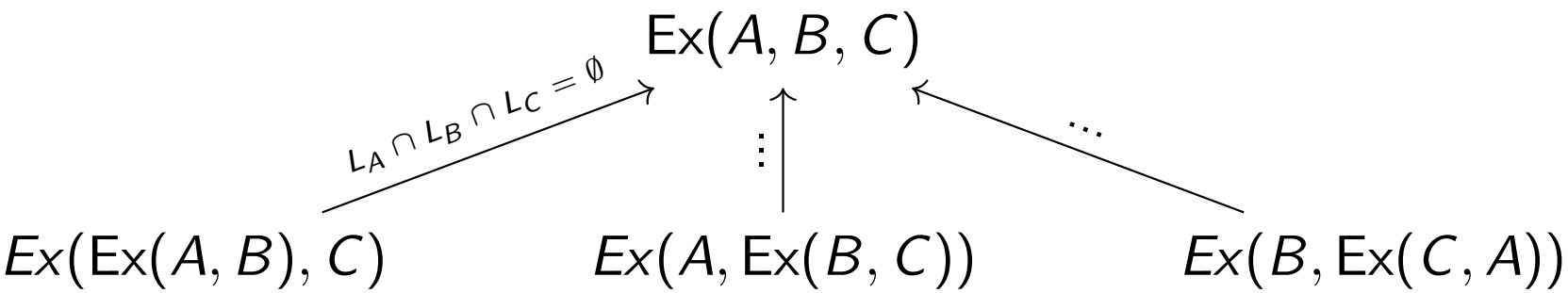


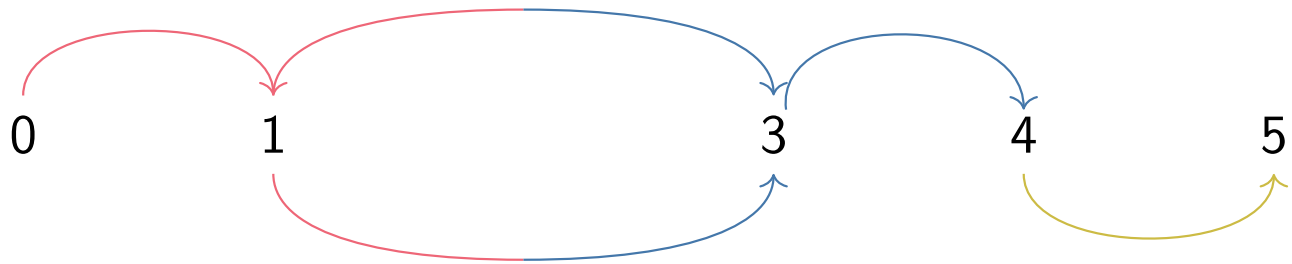


Finally a contribution: bicIG

One can make execution *unary*:

$$A :: B :: C \Rightarrow \mathrm{Ex}(A \barwedge B \barwedge C)$$

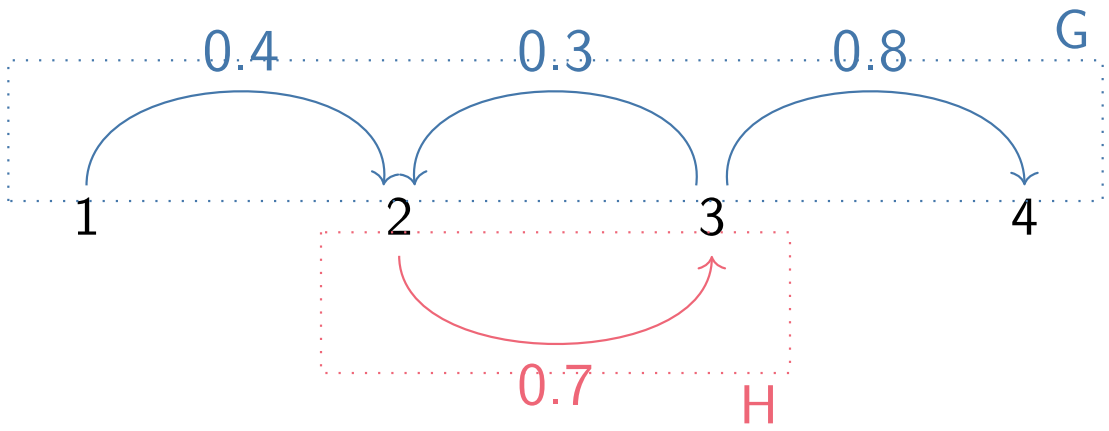




$\Rightarrow$ bi-colored Interaction Graphs (bicIG) allows **Medium Steps**

A really similar model: Weighted Interaction Graphs

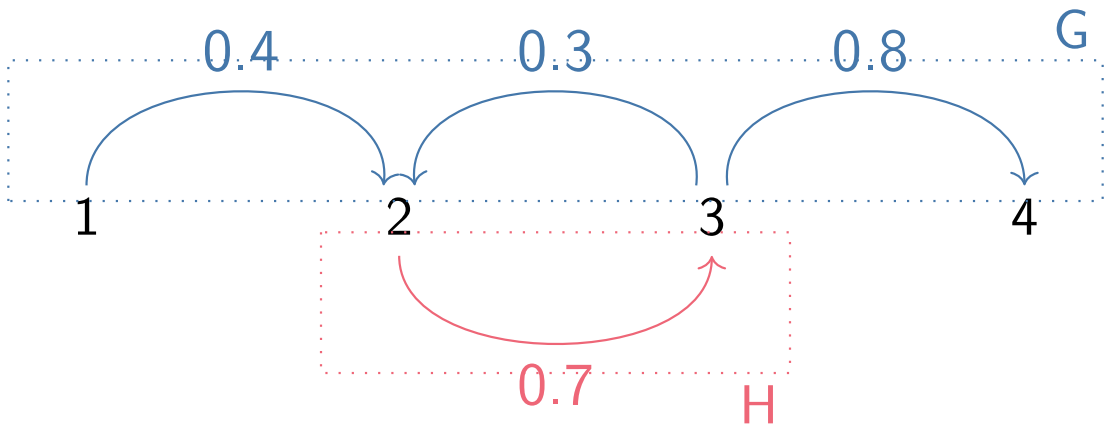
Now an edge e has weight $w(e) \in \bar{\mathbb{R}}$.



$\Rightarrow$ **Same combinatorics** as before, but we *compose* edge-like primitives

A really similar model: Weighted Interaction Graphs

Now an edge e has weight $w(e) \in \bar{\mathbb{R}}$.



$\Rightarrow$ **Same combinatorics** as before, but we *compose* edge-like primitives

- Primitive: $e : 1 \to 2$ with $w(e) = 0.4$
- Combination: $e; e' : 1 \to 3$, $w(e; e') = 0.4 \times 0.7$

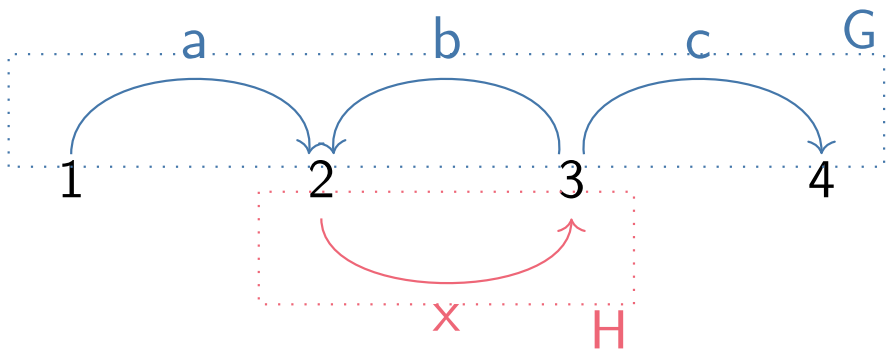
The unifying notion of diagram

Traditionally, execution = maximal alternated paths.

The unifying notion of diagram

Traditionally, execution = maximal alternated paths.

Paths = combine Edges → **Diagram** = combine "Primitives"



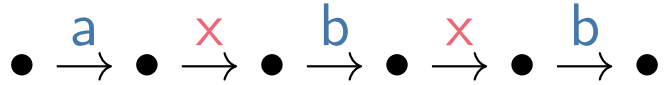
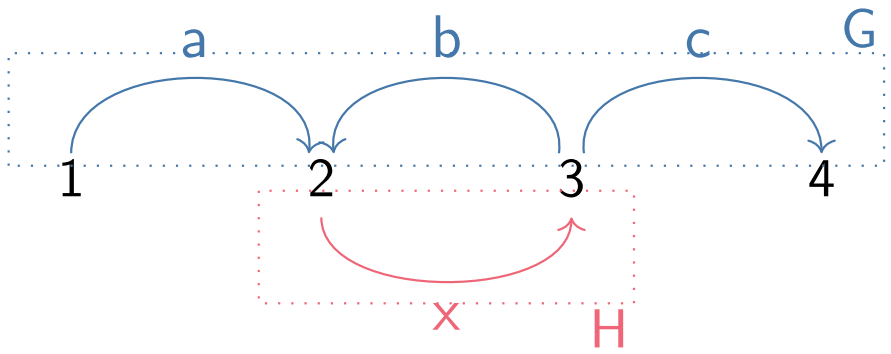


Diagram = Attempt at taking a path (more general, might fail)

The unifying notion of diagram

Traditionally, execution = maximal alternated paths.

Paths = combine Edges → **Diagram** = combine "Primitives"



$$\bullet \xrightarrow{a} \bullet \xrightarrow{x} \bullet \xrightarrow{b} \bullet \xrightarrow{x} \bullet \xrightarrow{b} \bullet$$

Diagram = Attempt at taking a path (more general, might fail)

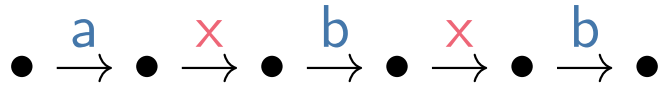
$$\Downarrow \delta := 1 \to 4.$$

Let us formalise Diagrams

Definition (Arities of a vertex)

In a graph $G = (V, E, s, t)$, a vertex is said to have:

- $+ar(v) := |\{e \in E \mid v = t(e)\}| \Rightarrow$ outgoing degree
- $-ar(v) := |\{e \in E \mid v = s(e)\}| \Rightarrow$ incoming degree
- $ar(v) := +ar(v) + -ar(v) \Rightarrow$ number of things touching v



Definition (Wire-Network Property)

Directed graph $G = (V, E, \mathtt{in}, \mathtt{out})$ has it when for all v in V:

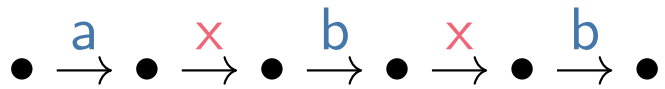
- $1 \leq ar(v)$ (touch an edge)
- $+ar(v), -ar(v) \leq 1$ (no more than one input and output)

Finally, diagrams!

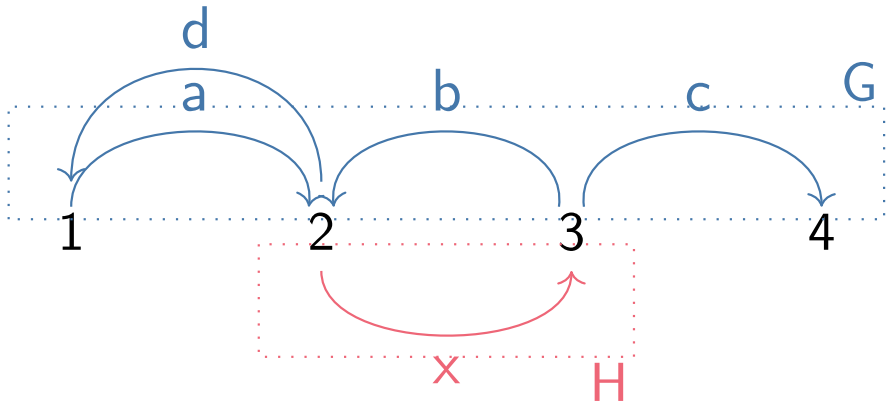
Definition (Diagram)

A diagram δ on a program P is given by:

- A shape: graph $|\delta|$ with wire-network property + connected + no cycle.
- An indexor: $\delta : E_{|\delta|} \to |P|$ (primitive above every edge)



Diagrams can be Bad



Not saturated :

$$\bullet \xrightarrow{a} \bullet \xrightarrow{x} \bullet \xrightarrow{b} \bullet \xrightarrow{x} \bullet \xrightarrow{b} \boxed{2 \in \mathrm{Cut}}$$

Not alternated :

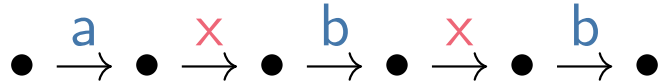
$$\bullet \xrightarrow{a} \bullet \xrightarrow{x} \bullet \xrightarrow{b} \boxed{\bullet} \xrightarrow{d} \bullet \xrightarrow{c} \bullet$$

Not correct :

$$\bullet \xrightarrow{a} \bullet \xrightarrow{x} \boxed{3 \neq 1} \xrightarrow{a} \bullet \xrightarrow{x} \bullet \xrightarrow{c} \bullet$$

Execution

If δ is **valid** one can **retract** it:



$\Downarrow \delta = (a; x; b; x; c) : 1 \to 4$ (observationally the same).

Definition (Execution)

Given a program P_L, set of locations K, define new program:

$$\mathsf{Ex}_K(P_L) := (\Downarrow \mathtt{VDiags}_K)_{L \cap \overline{K}}$$

$$\Rightarrow G :: H := \mathsf{Ex}_{(L_G \cap L_H)}(G \sqcup H)$$

What was the point again?

Theorem (Church-Rosser)

When $L \cap K = \emptyset$,

$$\mathrm{Ex}_L(\mathrm{Ex}_K(H)) = \mathrm{Ex}_{L \sqcup K}(H) = \mathrm{Ex}_K(\mathrm{Ex}_L(H))$$

Theorem (Traditional Associativity)

When $L_A \cap L_B \cap L_C = \emptyset$

$$\mathrm{Ex}(\mathrm{Ex}(A, B), C) = \mathrm{Ex}(A, B, C) = \mathrm{Ex}(A, \mathrm{Ex}(B, C))$$

$$\begin{aligned} &\mathrm{Ex}_{\overline{L_A \ominus L_B \ominus L_C}}(A + B + C) \\ &= \mathrm{Ex}_{(L_A \ominus L_B) \cap L_C}(\mathrm{Ex}_{L_A \cap L_B}(A + B) + C) \\ &= \ldots \end{aligned}$$

Realisability

Seiller's notion of **Asbtract Model of Computation**:

- Programs with locations (proofs):

$$p, q, r \in \mathbb{P}$$

- Combined through associative Execution (cut-elim):

$$\text{"} :: \text{"} : \mathbb{P}_L \times \mathbb{P}_K \to \mathbb{P}_{L \ominus K}$$

- Behaviours from observation with adjunction (termination):

$$\perp \subseteq \mathbb{P} \times \mathbb{P}$$

$\Rightarrow$ Adjunction required: $a :: b \perp c$ iff $a \perp b :: c$

Realisability

Seiller's notion of **Asbtract Model of Computation**:

- Programs with locations (proofs):

$$p, q, r \in \mathbb{P}$$

- Combined through associative Execution (cut-elim):

$$\text{"}::\text{"} : \mathbb{P}_L \times \mathbb{P}_K \to \mathbb{P}_{L \ominus K}$$

- Behaviours from observation with adjunction (termination):

$$\perp \subseteq \mathbb{P} \times \mathbb{P}$$

$\Rightarrow$ Adjunction required: $a :: b \perp c$ iff $a \perp b :: c$

Definition (Behaviour)

Set of program $B = T^{\perp}$ behaving well wrt to tests.

Linear Realisability for MLL (BHK)

Definition (Tensor)

From A_L, B_K, with $L \cap K = \emptyset$

$$(A \otimes B)_{L \sqcup K} := \{a :: b \mid a \in A, b \in B\}^{\perp\perp}$$

Definition (Implication)

From A_L, B_K, with $L \cap K = \emptyset$

$$(A \multimap B)_{L \sqcup K} := \{p \mid \forall a \in A_L, p :: a \in B_K\}$$

Linear Realisability for MLL (BHK)

Definition (Tensor)

From A_L, B_K, with $L \cap K = \emptyset$

$$(A \otimes B)_{L \sqcup K} := \{a :: b \mid a \in A, b \in B\}^{\perp\perp}$$

Definition (Implication)

From A_L, B_K, with $L \cap K = \emptyset$

$$(A \multimap B)_{L \sqcup K} := \{p \mid \forall a \in A_L, p :: a \in B_K\}$$

$(A \multimap B) = (A \otimes (B^{\perp}))^{\perp}$ and thus a behaviour.

Linear Realisability for MLL (BHK)

Definition (Tensor)

From A_L, B_K, with $L \cap K = \emptyset$

$$(A \otimes B)_{L \sqcup K} := \{a :: b \mid a \in A, b \in B\}^{\perp\perp}$$

Definition (Implication)

From A_L, B_K, with $L \cap K = \emptyset$

$$(A \multimap B)_{L \sqcup K} := \{p \mid \forall a \in A_L, p :: a \in B_K\}$$

$(A \multimap B) = (A \otimes (B^{\perp}))^{\perp}$ and thus a behaviour.

Theorem (GOAL: Correction)

$$\pi : \vdash A \Rightarrow \|\pi\| \in \|A\|$$

Funny thing

Definition (Lifting ::)

From A_L, B_K:
$(A :: B)_{L \ominus K} := \{a :: b \mid a \in A, b \in B\}^{\perp\perp}$.

Definition (Lifting $\perp$)

From A_L, B_K:
$A \perp B := \forall a \in A, b \in B \colon a \perp b$.

Funny thing

Definition (Lifting ::)

From A_L, B_K:
$(A :: B)_{L \ominus K} := \{a :: b \mid a \in A, b \in B\}^{\perp\perp}$.

Definition (Lifting $\perp$)

From A_L, B_K:
$A \perp B := \forall a \in A, b \in B \colon a \perp b$.

Theorem

$(\mathbb{B}, ::, \perp)$ *is also a model of computation* $\Rightarrow$ *Higher Types.*

Variants of Interaction Graphs

Interaction Graphs are very powerful, two limitations:

- No additives
- Usage of Infinity

Variants of Interaction Graphs

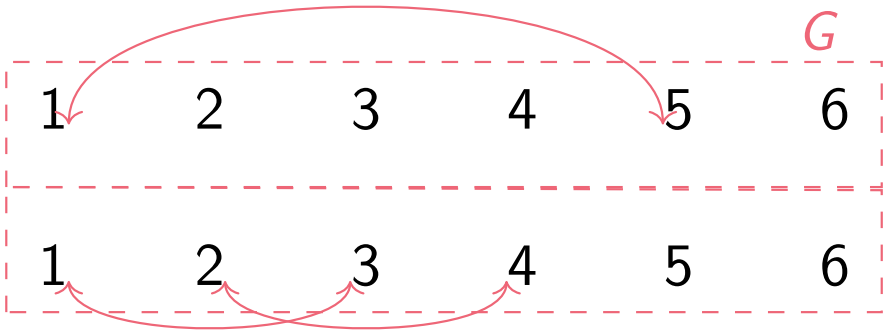
Interaction Graphs are very powerful, two limitations:

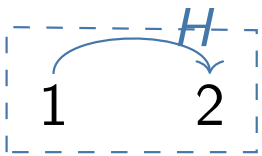
- No additives
- Usage of Infinity

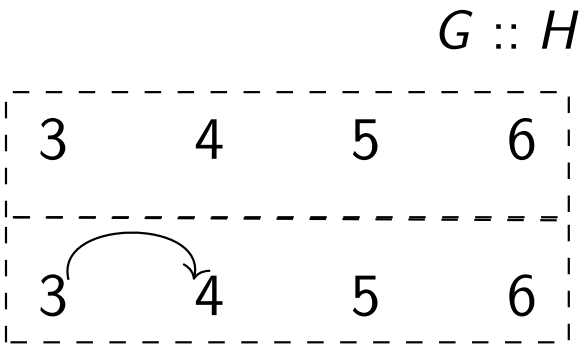
Can be extended:

- Additives: add slices $\Rightarrow$ Sliced Interaction Graphs
- Seely: add links between slices $\Rightarrow$ Thick Interaction Graphs
- Second order: ...

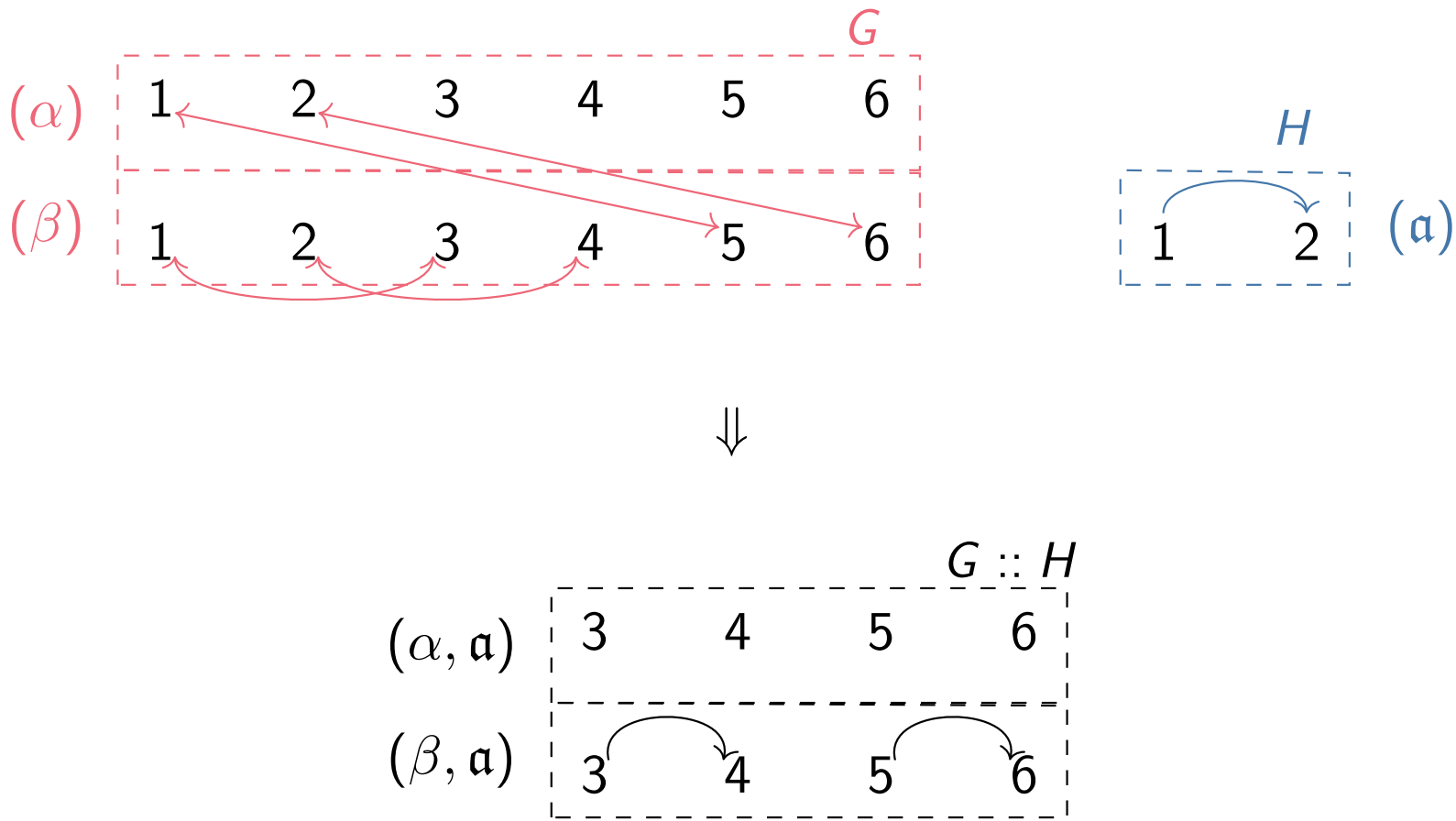
What about additives?







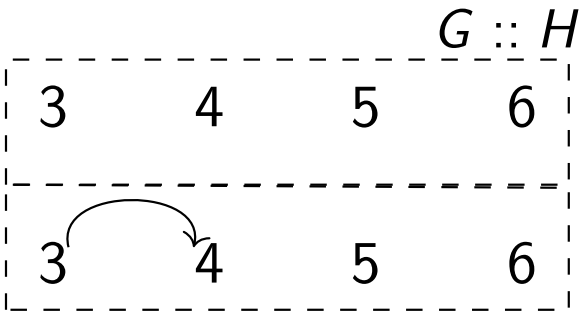
What about Thick Graphs ?



We compute all possible slices everytime.

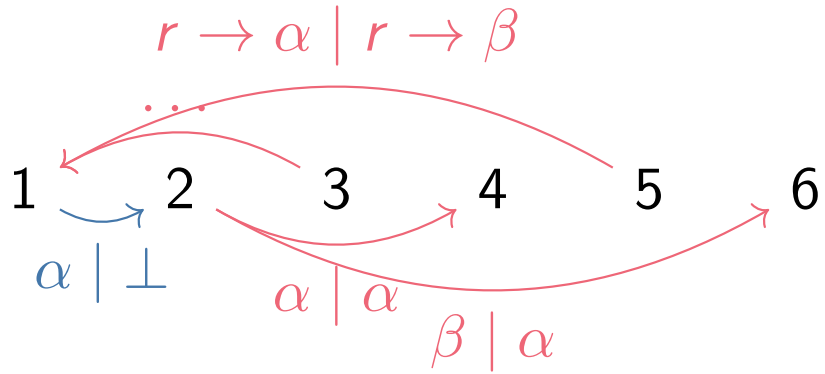
Useless slices

There is no need to remember the slice above!



Let's compute slices dynamically instead!

My little baby: Slider;Graphs



$\Rightarrow$ Primitives are **precondition / postcondition**

$e : r \to \alpha \mid r \to \beta$:= if traveler wants to go through:

- **Requires**: position in RED world $=$ slice α
- **After**: position in RED world $=$ slice β

Contraction with Slider;Graphs

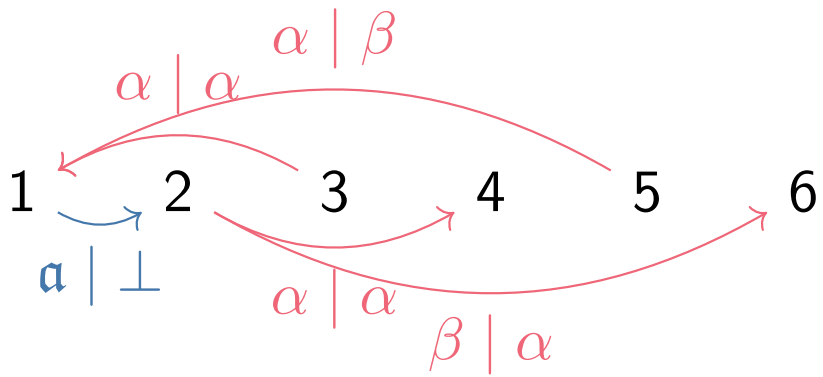
Some examples:

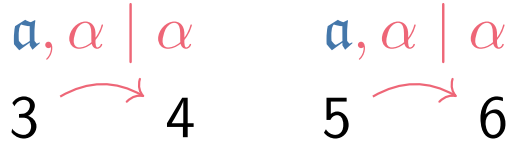
- $(r \to \alpha \mid r \to \beta); (r \to \gamma \mid r \to \alpha)$ fails
- $(r \to \alpha \mid r \to \beta); (r \to \beta, b \to \mathfrak{b} \mid r \to \alpha) = (r \to \alpha, b \to \mathfrak{b} \mid r \to \alpha)$

Contraction with Slider;Graphs

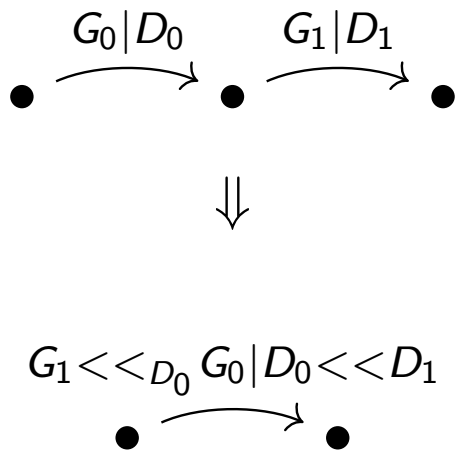
Some examples:

- $(r \to \alpha \mid r \to \beta); (r \to \gamma \mid r \to \alpha)$ fails
- $(r \to \alpha \mid r \to \beta); (r \to \beta, b \to \mathfrak{b} \mid r \to \alpha) = (r \to \alpha, b \to \mathfrak{b} \mid r \to \alpha)$





Composition of arrows



Theorem (Associativity of Composition of Primitives)

Given trackers G_0, G_1, G_2 and D_0, D_1, we have:

$$G_2 <<_{D_0<<D_1} (G_1 <<_{D_0} G_0) = (G_2 <<_{D_1} G_1) <<_{D_0} G_0$$

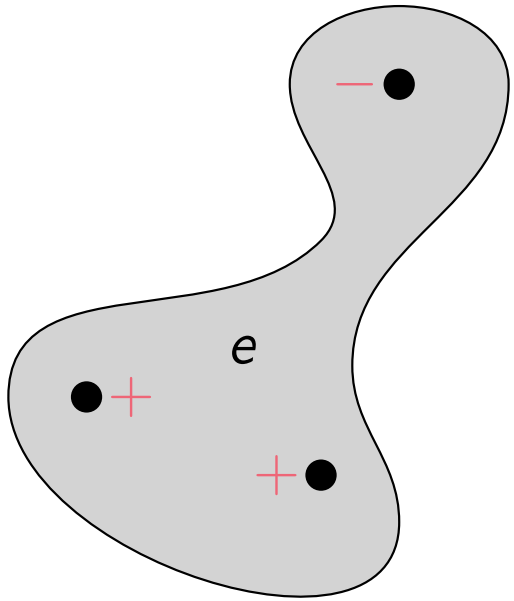
$\Rightarrow$ **Diagrams** makes this an associative model of computation **for free**.

Directed Hypergraphs

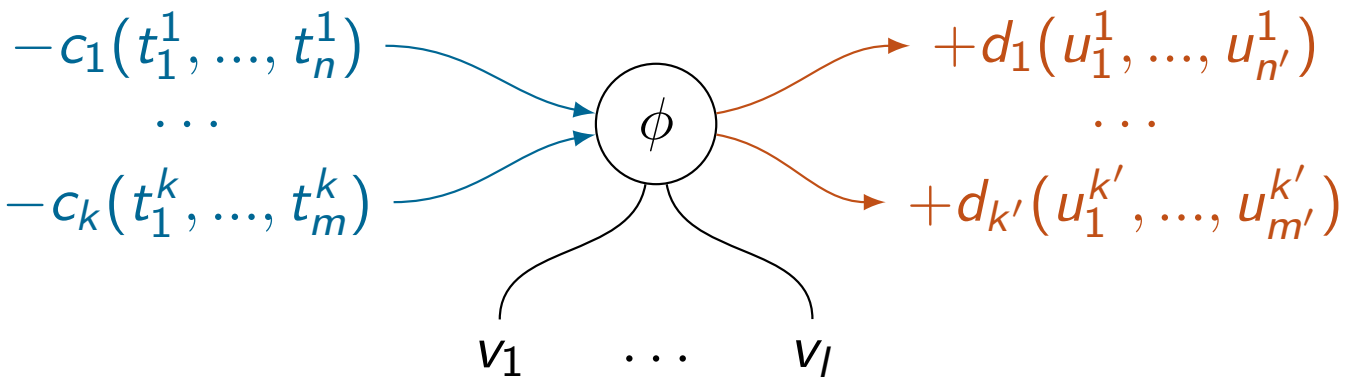
Definition (Directed hypergraph)

$$H = (V, E, \mathtt{in}, \mathtt{out})$$

With $\mathtt{in}, \mathtt{out} \colon E \to \mathcal{P}(V)$ *inputs/ouptputs* of hyperedges.



Constellation Model (Eng, Seiller, inspired by Girard)



A star: the primitive element

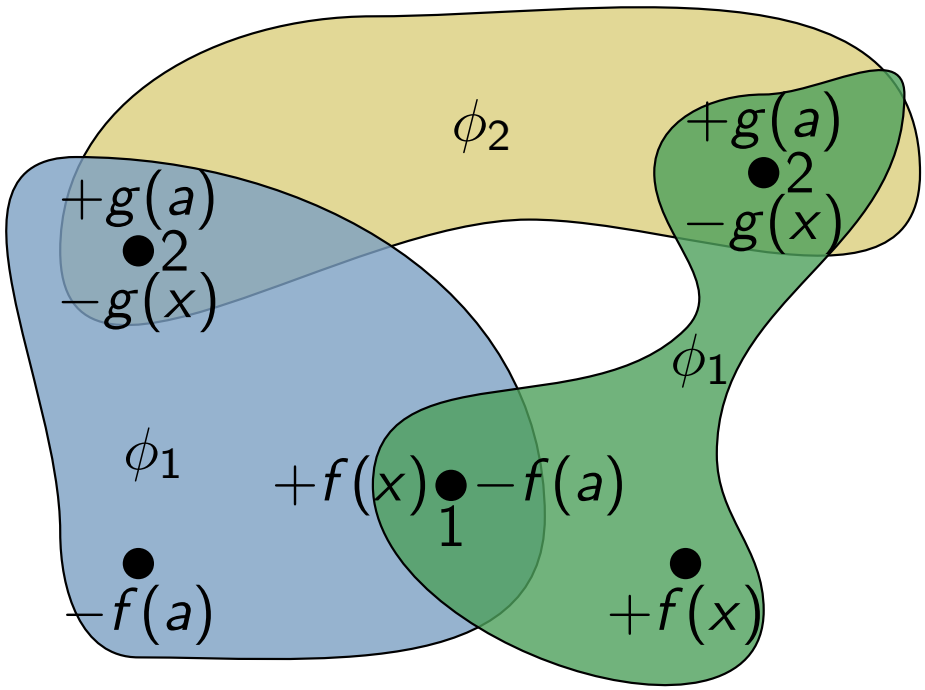
- Directed Graphs $\rightarrow$ Directed Hypergraphs
- Compose hyperedge $\rightarrow$ Glue + Unification (1st order Terms)
- Location $\rightarrow$ Colors

$\Rightarrow$ Wild combinatorics $\sim$ Tilings.

Example of Diagram

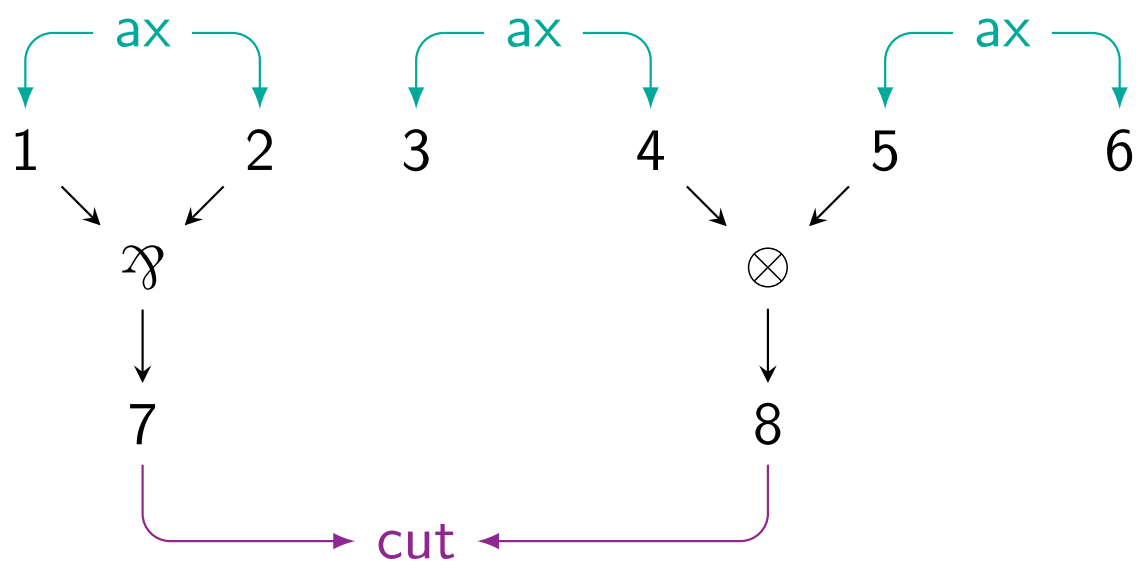
Take a constellation made of:

- $\phi_1 := [+1.f(x), -1.f(a), -2.g(x)]$
- $\phi_2 := [+2.g(a), +2.g(a)]$



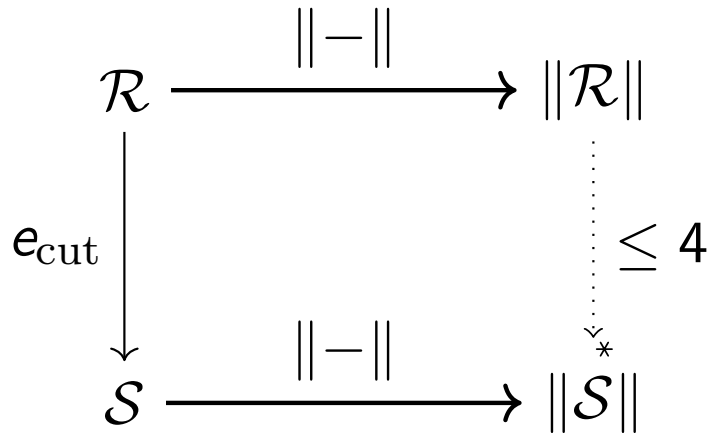
$\Rightarrow$ **Every vertex has max** 1 **input,** 1 **output** (Wire-Network).

Interpretation of Proofs



- Axiom: $[+1(X), +2(X)] \parallel [+3(X), +4(X)] \parallel [+5(X), +6(X)]$
- Rewiring:
  $[-1(X), +7(l.X)] \parallel [-2(X), +7(r.X)]$
  $[-4(X), +8(l.X)] \parallel [-5(X), +8(r.X)]$
- Cuts: $[-7(X), -8(X)]$

Cut-Elimination Simulation Theorem



Theorem (Simulate Cut-Elimination: Little Steps)

If $\mathcal{R} \rightsquigarrow_{e_{\mathrm{cut}}} \mathcal{S}$ *and* $\|e_{\mathrm{cut}}\| = [-v_1(X), -v_2(X)]$

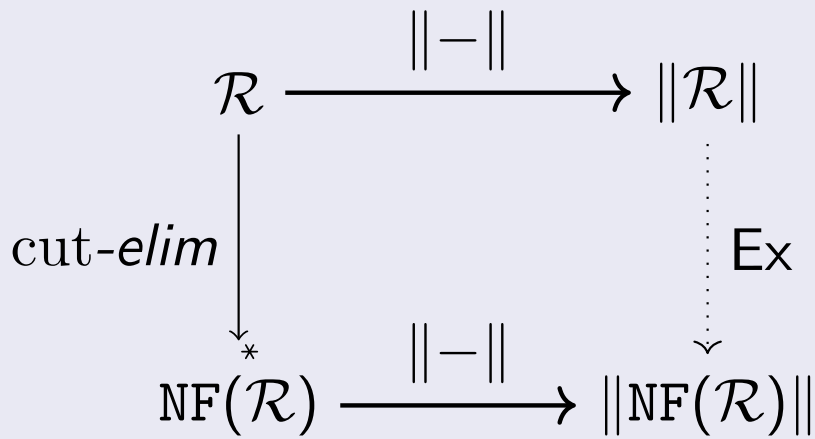
$$\mathsf{Ex}_{v_1,v_2}(\|\mathcal{R}\|) = \|\mathcal{S}\|$$

Correction

Theorem (Full Reduction $\sim$ GOI)

When $\mathcal{R} \rightsquigarrow^* \mathcal{S}$ with $\mathcal{S}$ in normal form:

$$\mathsf{Ex}(\|\mathcal{R}\|) = \|\mathcal{S}\|$$



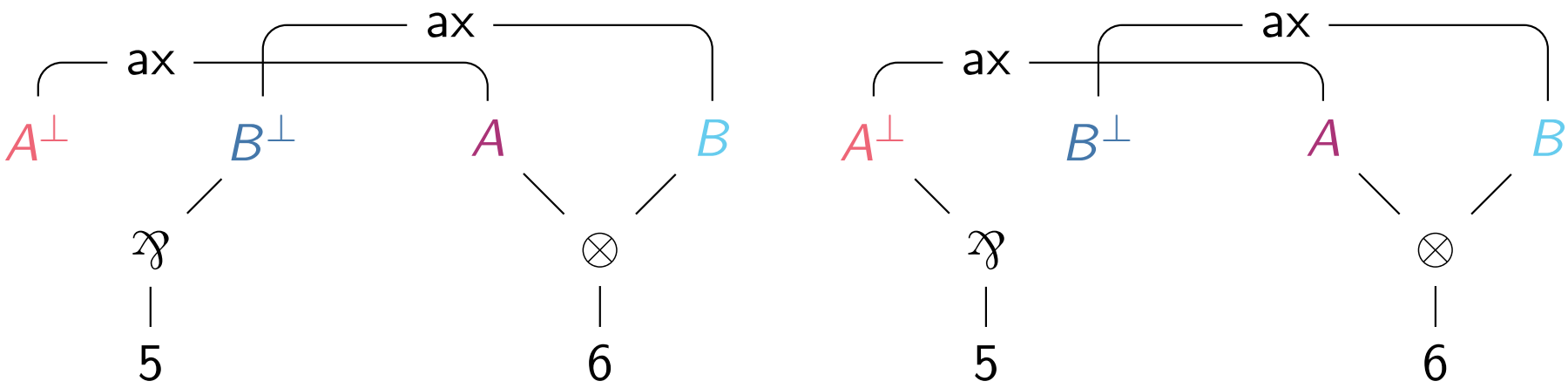
(In particular, Proof-Nets)

Theorem (Correction)

Let $\vdash \mathcal{R} : \Gamma$ be an MLL Proof-Net, Ω an interpretation of atoms.

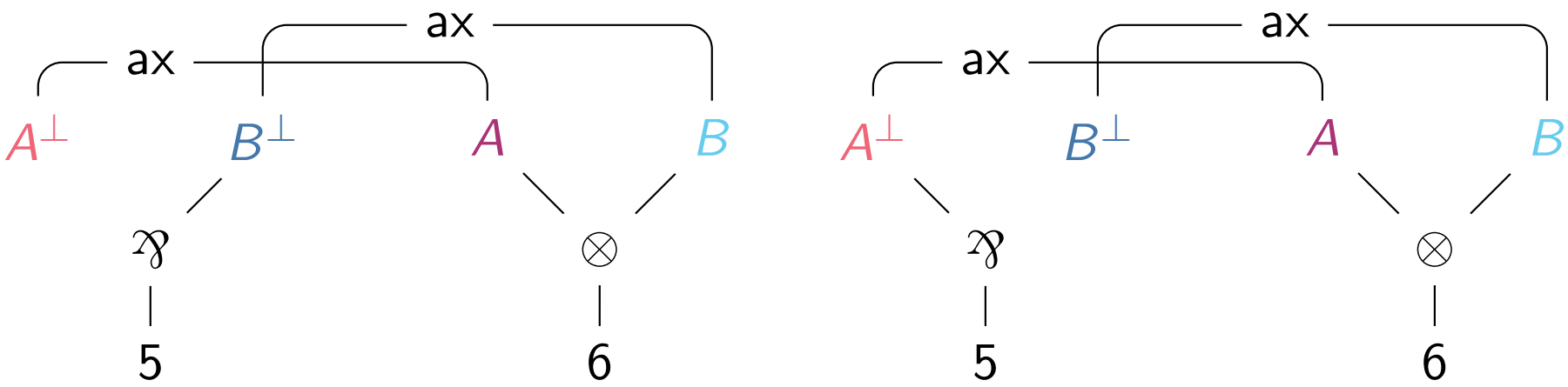
$$\mathsf{Ex}(\|\mathcal{R}\|) \in \|\vdash \Gamma\|_\Omega$$

A look back at Danos-Regnier



Test φ defines **hypergraph lower part** $\rightsquigarrow \|\mathcal{S}^\varphi\|$

A look back at Danos-Regnier



Test φ defines **hypergraph lower part** $\rightsquigarrow \|\mathcal{S}^\varphi\|$

Trial $\mathrm{T}_{\mathcal{S}}^{\varphi} := \|\mathcal{S}\| + \|\mathcal{S}^\varphi\|$ = Program VS Test.

Theorem (Internal Correctness Criterion)

Proof Structure $\mathcal{S}$ with $\mathtt{Concl}(\mathcal{S}) = \{v_1, ..., v_n\}$*:*

$$\mathcal{S} \text{ correct} \Leftrightarrow \forall \varphi, \mathrm{Ex}(\mathrm{T}_{\mathcal{S}}^{\varphi}) = [\{v_1(X), ..., v_n(X)\}]$$

Strong Completeness

Theorem (Internal Correctness : Orthogonality)

Let $\mathcal{S}$ be a Proof Structure (ie, not a priori correct).
Let $\mathtt{T}$ be the set of tests:

$$\mathcal{S} \text{ is correct } \Leftrightarrow \|\mathcal{S}\|_{ax} \perp \mathtt{T}$$

$\Phi \in \|\vdash \Gamma\|_\Omega$ proof-like: syntactically like a proof (axioms).

Theorem (Completeness for $\mathrm{MLL}{+}\mathrm{MIX}$)

If $\Phi \in \|\vdash \Gamma\|_\Omega$ and Φ proof-like w.r.t. $\vdash \Gamma$

Then $\exists \vdash \mathcal{S} : \Gamma$ such that $\|\mathcal{S}\| = \Phi$.

A weird sense of Fun?

The nice storytelling is over!

A weird sense of Fun?

The nice storytelling is over!

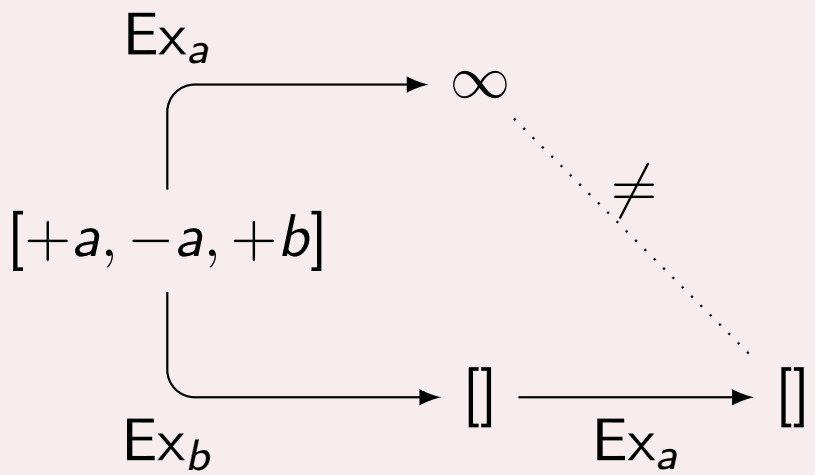
Let's have some FUN!
(By fun I mean discuss 3 technical points of interest)

- One combinatorial
- One logical
- One geometrical

These are Hypermodels

Execution can compute ∞ **many diagrams**.

Church Rosser Breaks?



Combinatorics: Problem of construction

Unique Normal Form $\simeq \mathsf{Ex}(\Phi)$ $\quad \| \quad$ Many Normal Form $\simeq \Downarrow \delta$

Combinatorics: Problem of construction

Unique Normal Form $\simeq \mathsf{Ex}(\Phi)$ $\quad \| \quad$ Many Normal Form $\simeq \Downarrow \delta$

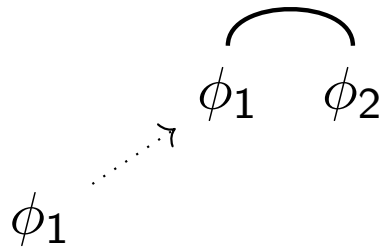
Build a diagram inductively:

ϕ_1

Combinatorics: Problem of construction

Unique Normal Form $\simeq \mathsf{Ex}(\Phi)$ $\quad \| \quad$ Many Normal Form $\simeq \Downarrow \delta$

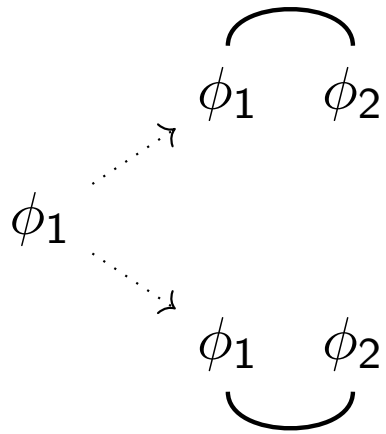
Build a diagram inductively:



Combinatorics: Problem of construction

Unique Normal Form $\simeq \mathsf{Ex}(\Phi)$ $\quad \| \quad$ Many Normal Form $\simeq \Downarrow \delta$

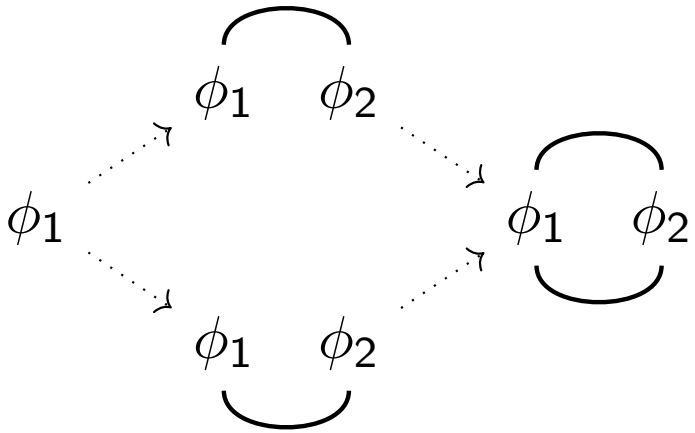
Build a diagram inductively:



Combinatorics: Problem of construction

Unique Normal Form $\simeq \mathsf{Ex}(\Phi)$ $\quad \| \quad$ Many Normal Form $\simeq \Downarrow \delta$

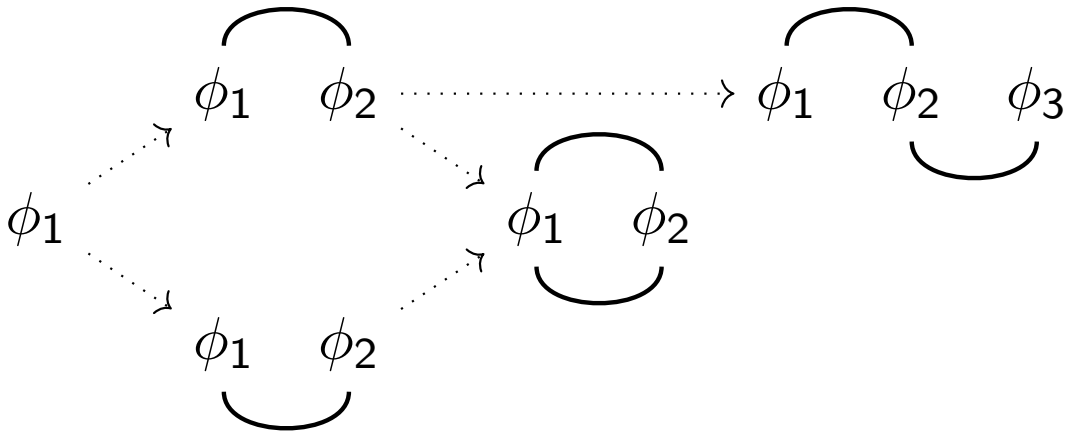
Build a diagram inductively:



Combinatorics: Problem of construction

Unique Normal Form $\simeq \mathsf{Ex}(\Phi)$ $\quad \| \quad$ Many Normal Form $\simeq \Downarrow \delta$

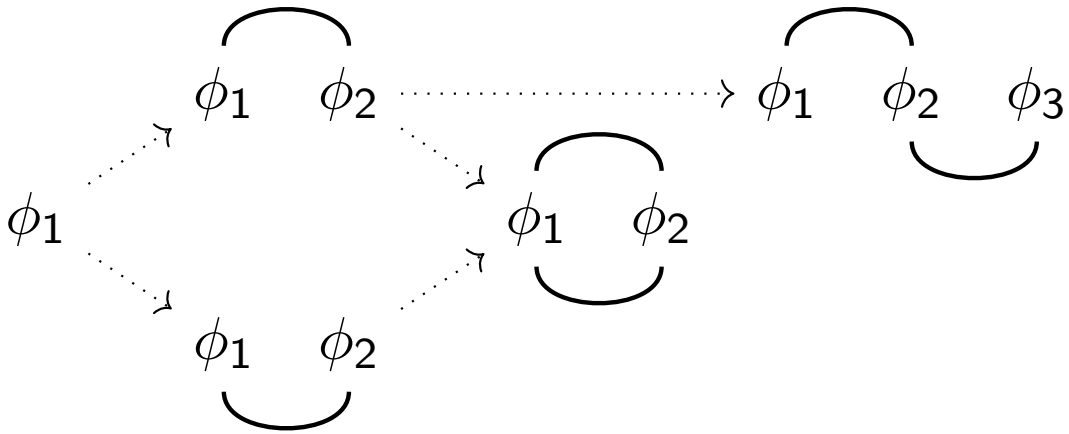
Build a diagram inductively:



Combinatorics: Problem of construction

Unique Normal Form $\simeq \mathsf{Ex}(\Phi)$ $\quad \| \quad$ Many Normal Form $\simeq \Downarrow \delta$

Build a diagram inductively:

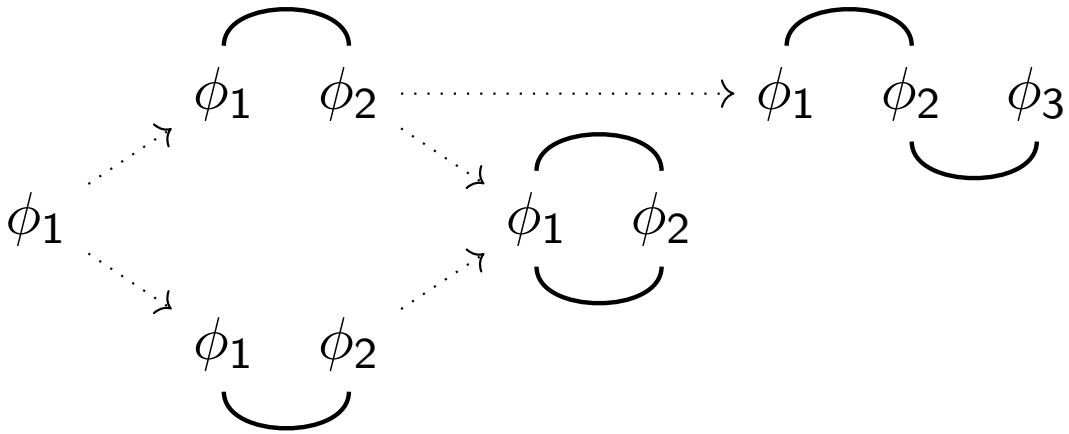


Compute diagram independently $\Rightarrow$ Count multiple times

Combinatorics: Problem of construction

Unique Normal Form $\simeq \mathrm{Ex}(\Phi)$ $\quad \| \quad$ Many Normal Form $\simeq \Downarrow \delta$

Build a diagram inductively:



Compute diagram independently $\Rightarrow$ Count multiple times

$\Rightarrow$ Open Problem: **Number of (connected) Constructions?**

A bottom-up process calculus

Example of program in the calculus:

$$*(+g(a,y)) \parallel *(-f(b) \mid +g(a,b) \mid +g(a,c))$$

A bottom-up process calculus

Example of program in the calculus:

$$*(+g(a,y)) \parallel *(-f(b) \mid +g(a,b) \mid +g(a,c))$$

Theorem

Let Φ be a constellation, $\exists \tau$ explicit bijection:

- *Good executions of process $\|\Phi\|$*
- *Connected constructions of diagrams in $\mathtt{CDiags}(\Phi)$*

Such that for an execution $\vec{s}$: $\|\Downarrow \tau(\vec{s})\| \simeq t(\vec{s})$

Compilation: Interaction Graphs and Stars

Definition (Interpretation $\|f\|_{ab}$)

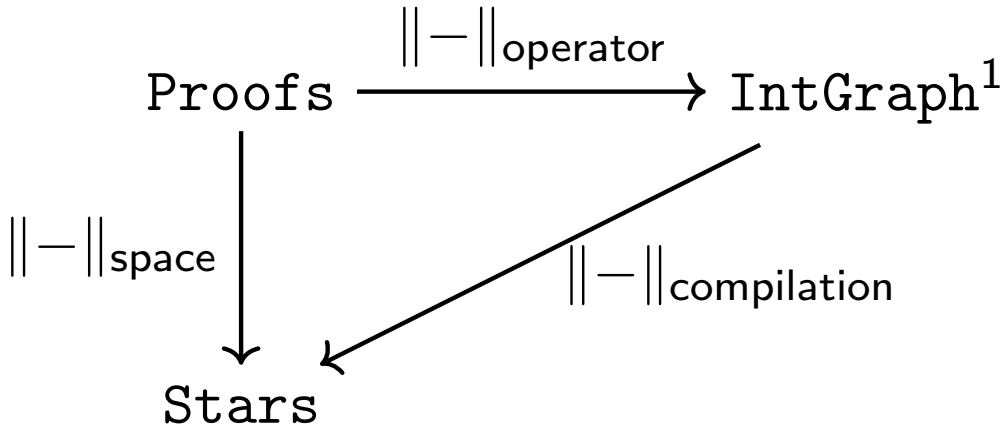
$$u \to v \rightsquigarrow [-a.u, +b.v]$$

Theorem

$$\mathsf{Ex}_b(\|F\|_{ab} + \|G\|_{bc}) = \|F.G\|_{ac}$$

$$\begin{aligned}\|F :: G\|_{io} = \mathsf{Ex}_{a,b}(& \\ & \|F_{ib}\| + \|F_{ao}\| \\ & + \|G_{ia}\| + \|G_{bo}\| \\ & + \|F_{a,b}\| + \|G_{b,a}\|)\end{aligned}$$

A Funny Thing Happened on the Way to the Stars



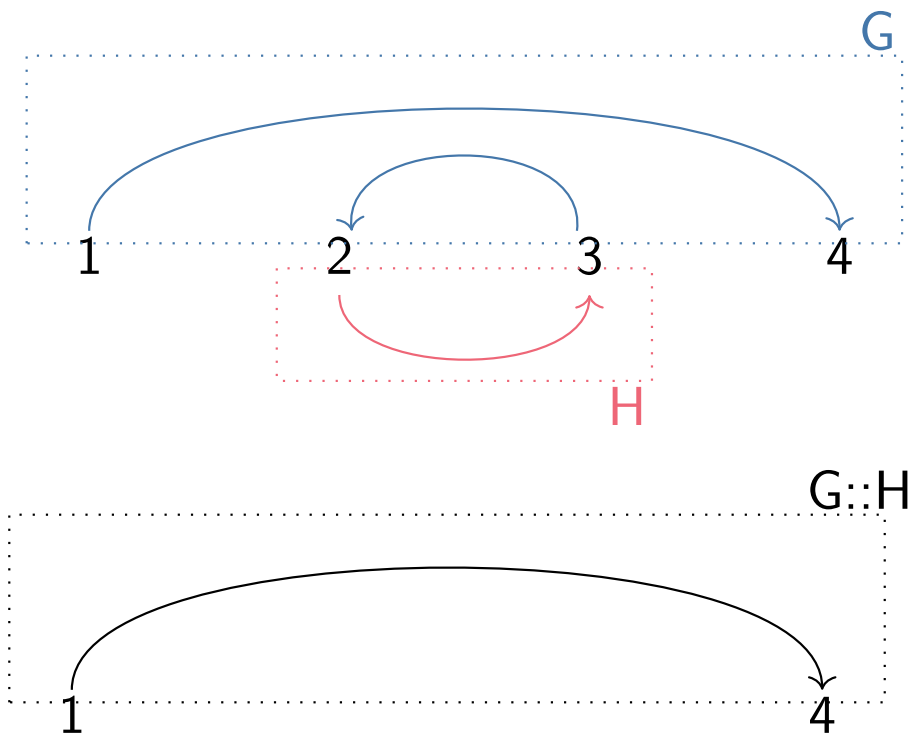
This does not commute!

(Two correctness criterion)

[1] Flows, actually

Geometric: Disappearance of internal cycles

Annoying phenomena:



Information loss is bad: Seiller's Project

Cannot internalise correctness

Orthogonality = Long trip criterion = exactly one cycle

$\Rightarrow$ Solution: adjoining a counter to programs.

Information loss is bad: Seiller's Project

Cannot internalise correctness

Orthogonality = Long trip criterion = exactly one cycle

$\Rightarrow$ Solution: adjoining a counter to programs.

- New programs: (G, a) with G graph, a number
- New execution: $(G, a) :: (H, b) = (G :: H, a + b + [\![G, H]\!])$

Information loss is bad: Seiller's Project

Cannot internalise correctness

Orthogonality = Long trip criterion = exactly one cycle

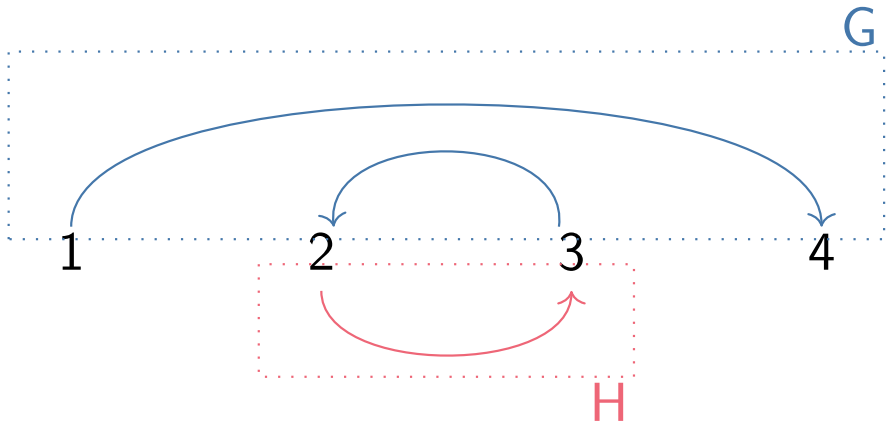
$\Rightarrow$ Solution: adjoining a counter to programs.

- New programs: (G, a) with G graph, a number
- New execution: $(G, a) :: (H, b) = (G :: H, a + b + [\![G, H]\!])$

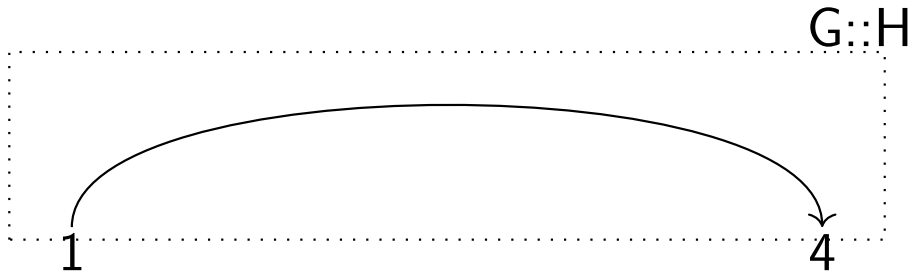
Still associative by 2-cocycle (trefoil) property:

$$[\![B, C]\!] + [\![A, B :: C]\!] = [\![A, B]\!] + [\![A :: B, C]\!]$$

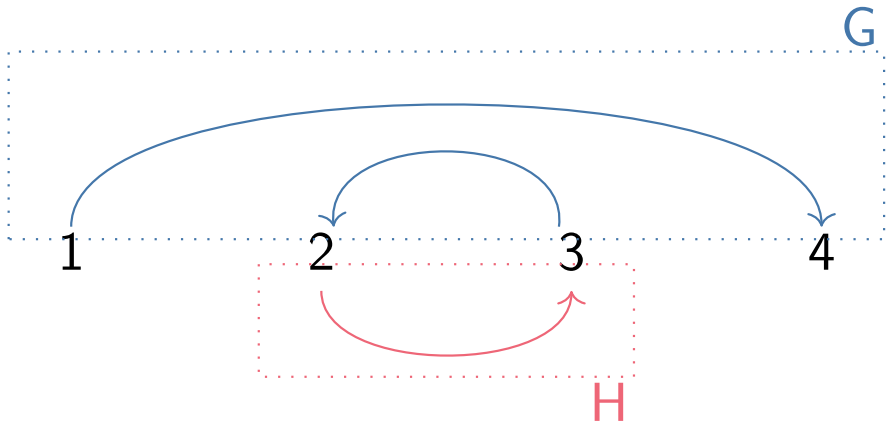
Why do we forget these cycles in the first place ?



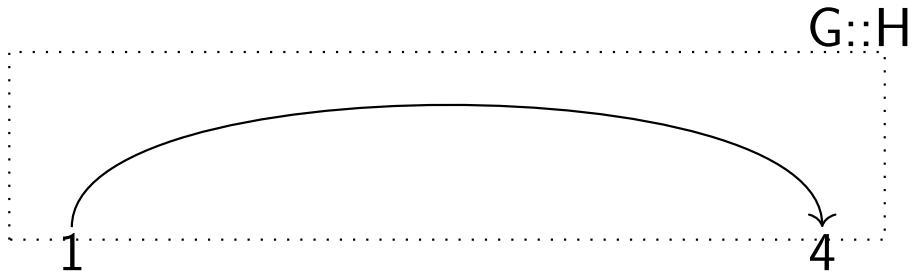
We are hiding 2 and 3...



Why do we forget these cycles in the first place ?



We are hiding 2 and 3...



In a discrete setting, the cycle *needs its locations to exist*.

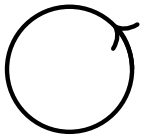
What could we do?

Seemingly two solutions to explore:

What could we do?

Seemingly two solutions to explore:

- **Use Geometry**: Some directed objects without boundaries:



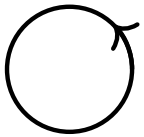
Geometric Realisation, homotopy/homology, Van Kampen etc...

Compose programs = glue on boundaries $\sim$ cobordisms.

What could we do?

Seemingly two solutions to explore:

- **Use Geometry**: Some directed objects without boundaries:



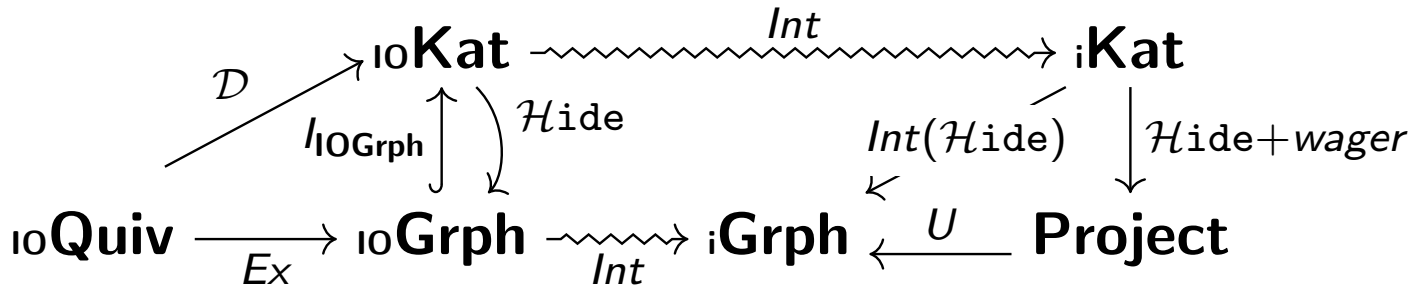
Geometric Realisation, homotopy/homology, Van Kampen etc...

Compose programs = glue on boundaries $\sim$ cobordisms.

- **Control Hiding**: Use Higher Cells (done in the PhD).

Also $Graphs \Rightarrow Semi - Categories$ so paths autocompute.

Summary


IOKat
Int
iKat
$\mathcal{D}$
$I_{\mathbf{IOGrph}}$
$\mathcal{H}$ide
$Int(\mathcal{H}\mathtt{ide})$
$\mathcal{H}$ide+wager
IOQuiv
Ex
IOGrph
Int
iGrph
U
Project

Why did I want to study that?

Why did I want to study that?

There are many proof-systems: NJ, NK, LK, Modal Logics etc. . .

Why did I want to study that?

There are many proof-systems: NJ, NK, LK, Modal Logics etc. . .

Creating a new proof-system: prove cut elimination again.

Why did I want to study that?

There are many proof-systems: NJ, NK, LK, Modal Logics etc. . .

Creating a new proof-system: prove cut elimination again.

Starting from cut-elimination (computation) $\Rightarrow$ Logics are fragments.

Why did I want to study that?

There are many proof-systems: NJ, NK, LK, Modal Logics etc. . .

Creating a new proof-system: prove cut elimination again.

Starting from cut-elimination (computation) $\Rightarrow$ Logics are fragments.

Worst case scenario: compile proofs $\rightarrow$ more expressive model of computation.

Why did I want to study that?

There are many proof-systems: NJ, NK, LK, Modal Logics etc. . .

Creating a new proof-system: prove cut elimination again.

Starting from cut-elimination (computation) $\Rightarrow$ Logics are fragments.

Worst case scenario: compile proofs $\rightarrow$ more expressive model of computation.

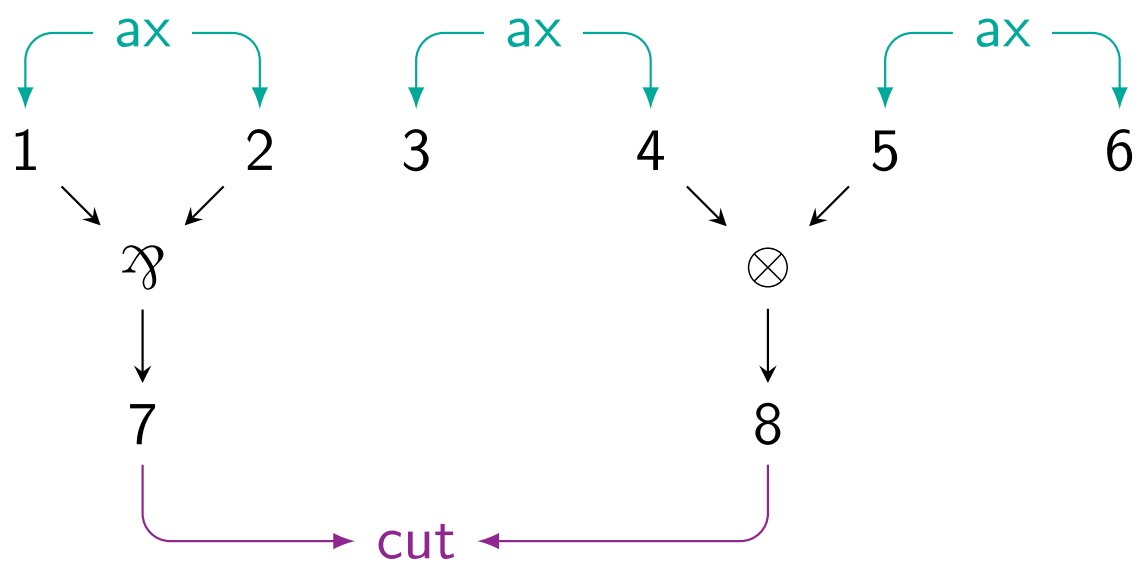
This might be a way to do a modular and "Open Source" Logic

What about the future ?

- Hypergraphings
- Stars with terms *as locations* (not colors).
- Do Full Linear Logic
- Explore richer behaviors: Affine Exponential, etc...

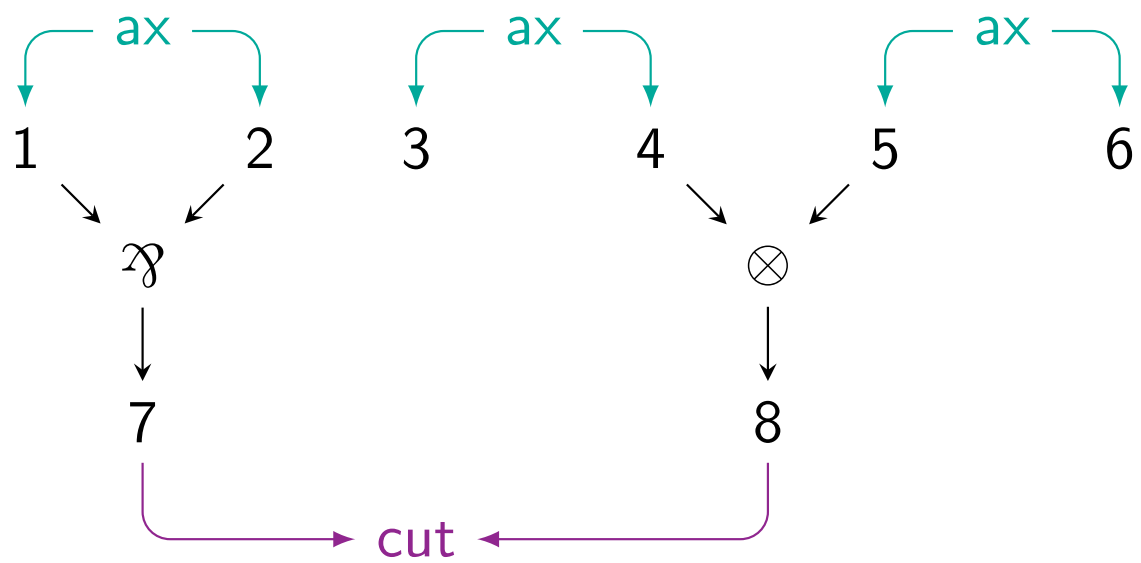
Thank you everyone!

Interpretation of Proofs



- Axiom: $[+1(X), +2(X)] \parallel [+3(X), +4(X)] \parallel [+5(X), +6(X)]$
- Rewiring:
  $[-1(X), +7(l.X)] \parallel [-2(X), +7(r.X)]$
  $[-4(X), +8(l.X)] \parallel [-5(X), +8(r.X)]$
- Cuts: $[-7(X), -8(X)]$

Cut-Elimination Simulation 1

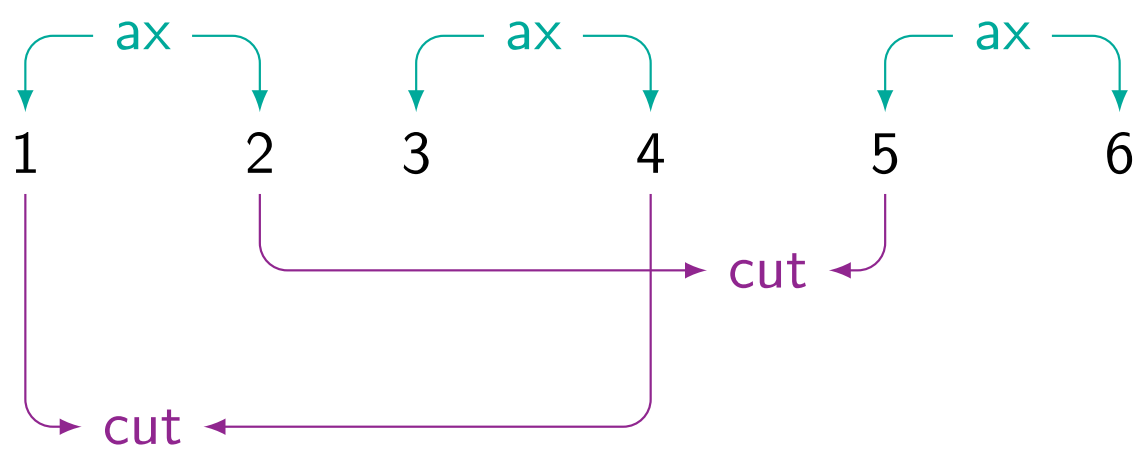


- Rewiring:
  $[-1(X), +7(l.X)] \parallel [-2(X), +7(r.X)]$
  $[-4(X), +8(l.X)] \parallel [-5(X), +8(r.X)]$
- Cuts: $[-7(X), -8(X)]$

$$[-1(X), +7(l.X)] \parallel [-7(X), -8(X)] \parallel [-4(X), +8(l.X)] \rightarrow [-1(X), -4(X)]$$

$$[-2(X), +7(r.X)] \parallel [-7(X), -8(X)] \parallel [-5(X), +8(r.X)] \rightarrow [-2(X), -5(X)]$$

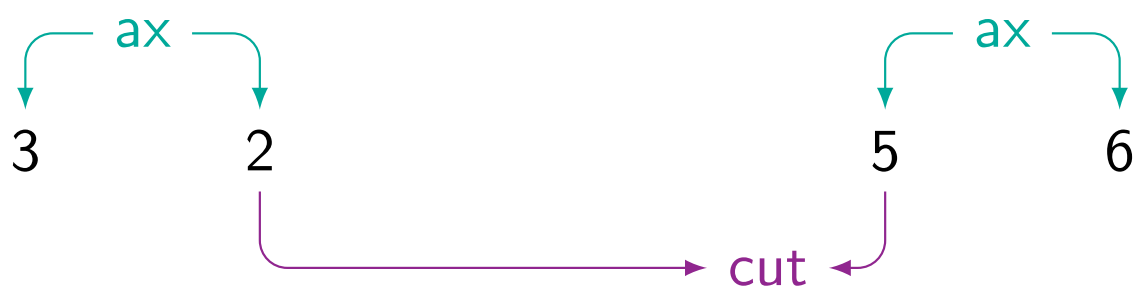
Cut-Elimination Simulation 2



- Axiom: $[+1(X), +2(X)] \parallel [+3(X), +4(X)] \parallel [+5(X), +6(X)]$
- Previous: $[-1(X), -4(X)] \parallel [-2(X), -5(X)]$
- Etc...

$$[+1(X), +2(X)] \parallel [-1(X), -4(X)] \parallel [+3(X), +4(X)] \rightarrow [+2(X), +3(X)]$$

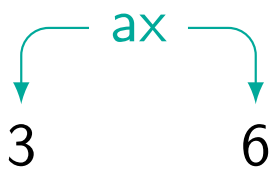
Cut-Elimination Simulation 3



- Axiom: $[+1(X), +2(X)] \parallel [+3(X), +4(X)] \parallel [+5(X), +6(X)]$
- Previous: $[-1(X), -4(X)] \parallel [+2(X), +3(X)] \parallel [-2(X), -5(X)]$
- Etc...

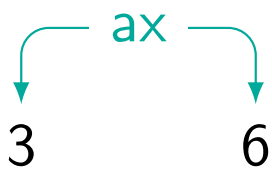
$$[+2(X), +3(X)] \parallel [-2(X), -5(X)] \parallel [+5(X), +6(X)] \rightarrow [+3(X), +6(X)]$$

Cut-Elimination Simulation 4



$$[+3(X), +6(X)]$$

Cut-Elimination Simulation 4



$$[+3(X), +6(X)]$$

Theorem (Simulate Cut-Elimination: Little Steps)

If $\mathcal{R} \rightsquigarrow_{e_{\mathrm{cut}}} \mathcal{S}$ *and* $\|e_{\mathrm{cut}}\| = [-v_1(X), -v_2(X)]$

$$\mathsf{Ex}_{v_1,v_2}(\|\mathcal{R}\|) = \|\mathcal{S}\|$$