



```
83
84
85 (** * Species Sum/Coproduct *)
86
87 Definition spec_sum (X Y : Species) : Species
88   := ((X.1 + Y.1)%type; sum_rect _ X.2 Y.2).
89
90 Lemma sigma_functor_sum (X : Type) (P Q : X -> Type) :
91   ({x : X & P x} + {x : X & Q x}) <=> {x : X & (P x + Q x)%type}.
92 Proof.
93   refine (equiv_adjointify _ _ _).
94   - intros [[x w] | [x w]]; exists x; [left | right]; apply w.
95   - intros [x [w | w]]; [left | right]; apply (x; w).
96   - intros [x [w | w]]; reflexivity.
97   - intros [[x w] | [x w]]; reflexivity.
98 Defined.
99
100 Definition stuff_spec_sum (P Q : FinSet -> Type) := fun A => (P A + Q A)%type.
101
102 Lemma stuff_spec_sum_correct (P Q : FinSet -> Type) :
103   spec_from_stuff (stuff_spec_sum P Q)
104   =
105   spec_sum (spec_from_stuff P) (spec_from_stuff Q).
106 Proof.
107   apply path_sigma_uncurried. refine (_; _).
108   - unfold stuff_spec_sum. simpl. symmetry.
109   - apply path_universe_uncurried.
110   - apply sigma_functor_sum.
111   - simpl. apply path_arrow. intros x.
112     refine ((transport_arrow _ _ _) @ _).
113     refine ((transport_const _ _) @ _).
114     path_via (transport idmap
115               (path_universe_uncurried (sigma_functor_sum FinSet P Q)
116                 x).1.
117               f_ap. f_ap. apply inv_V.
118               nath via ((sigma functor sum FinSet P Q) x).1.
```

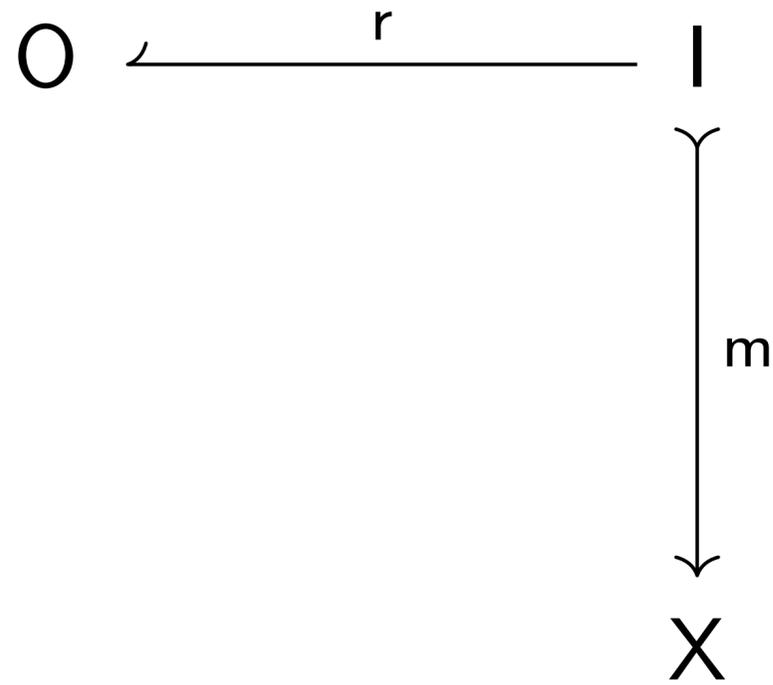
Plan of the talk:

- I. **A Case Study in Applied Category Theory:
from Categorical Rewriting to
Rule-algebraic Combinatorics**
- II. **The `coreact.wiki` Initiative**

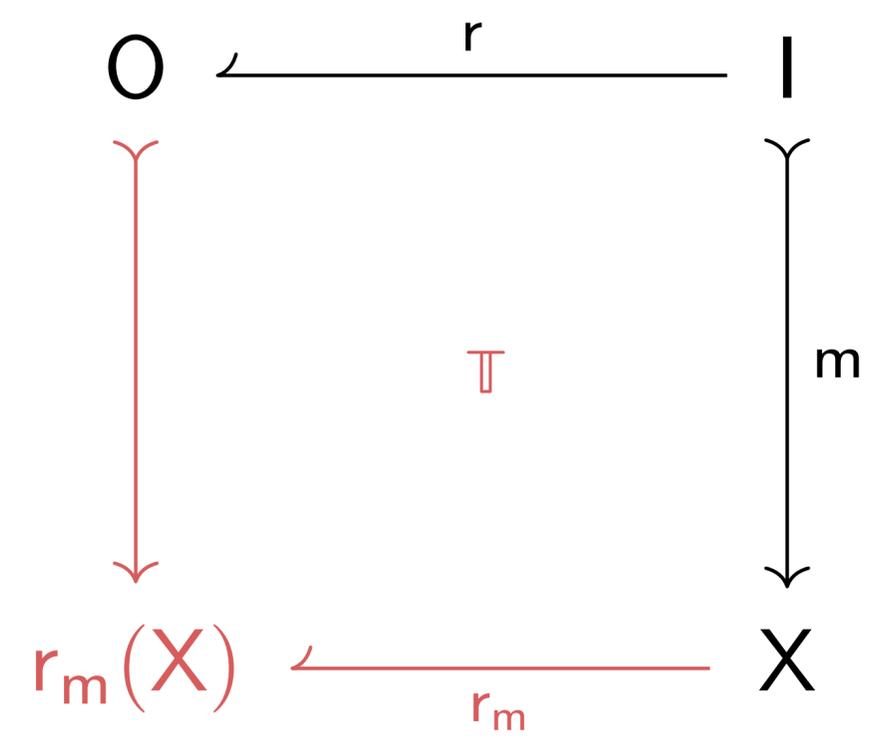
Plan of the talk:

- I. **A Case Study in Applied Category Theory:
from Categorical Rewriting to
Rule-algebraic Combinatorics**
- II. **The `coreact.wiki` Initiative**

A quick tour of categorical rewriting

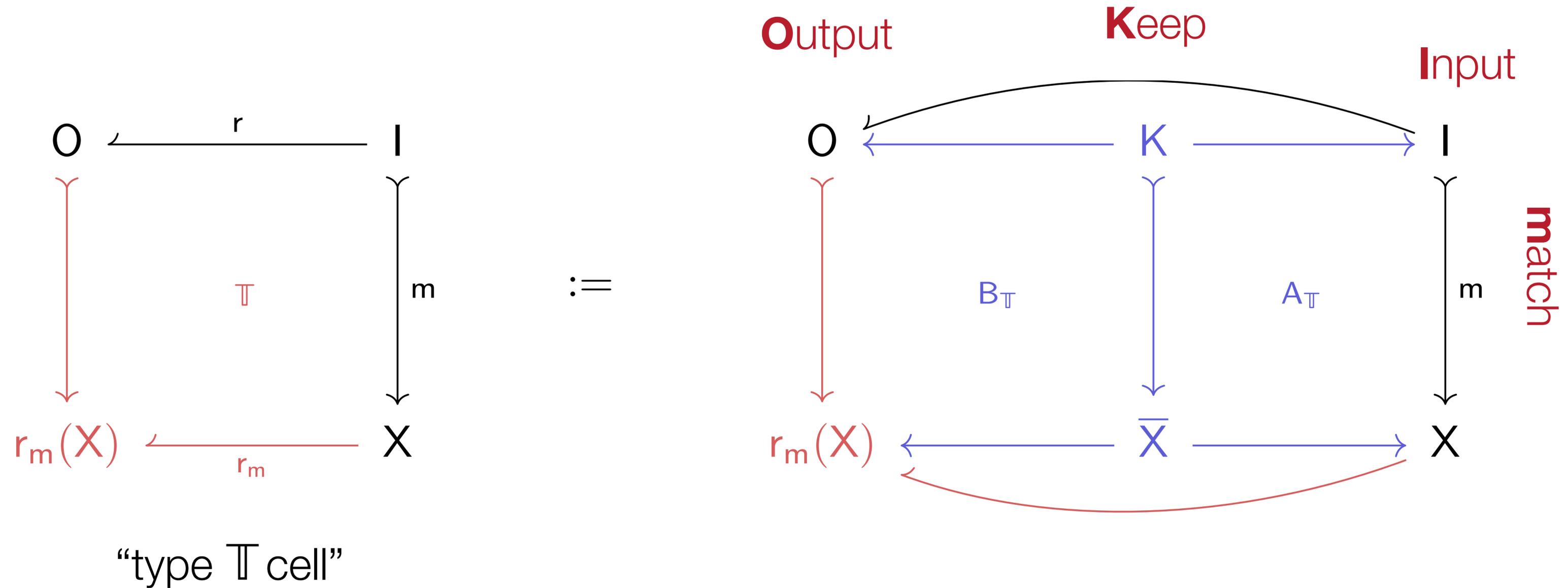


A quick tour of categorical rewriting

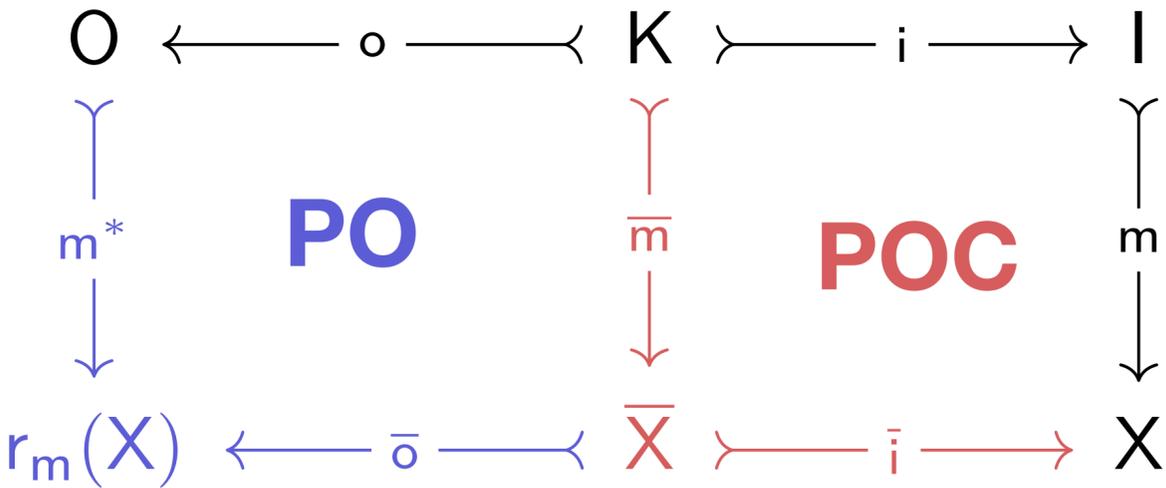


“type Π cell”

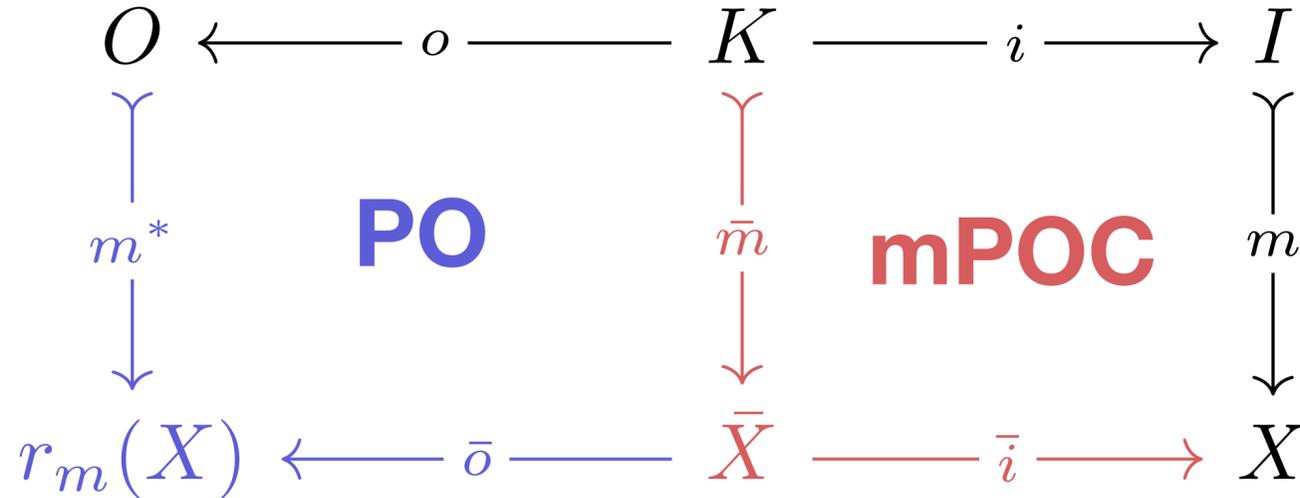
A quick tour of categorical rewriting



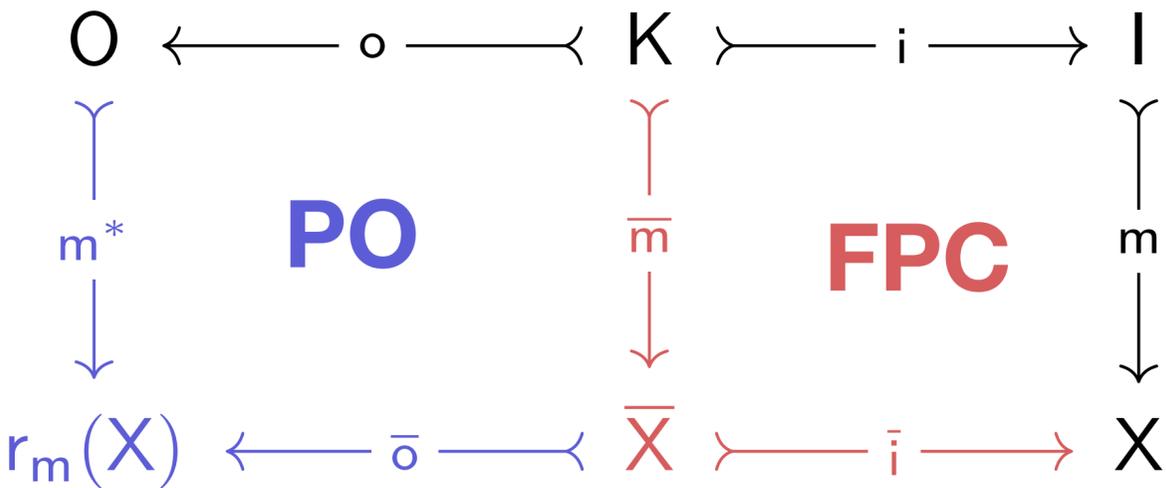
Four flavours of categorical rewriting semantics



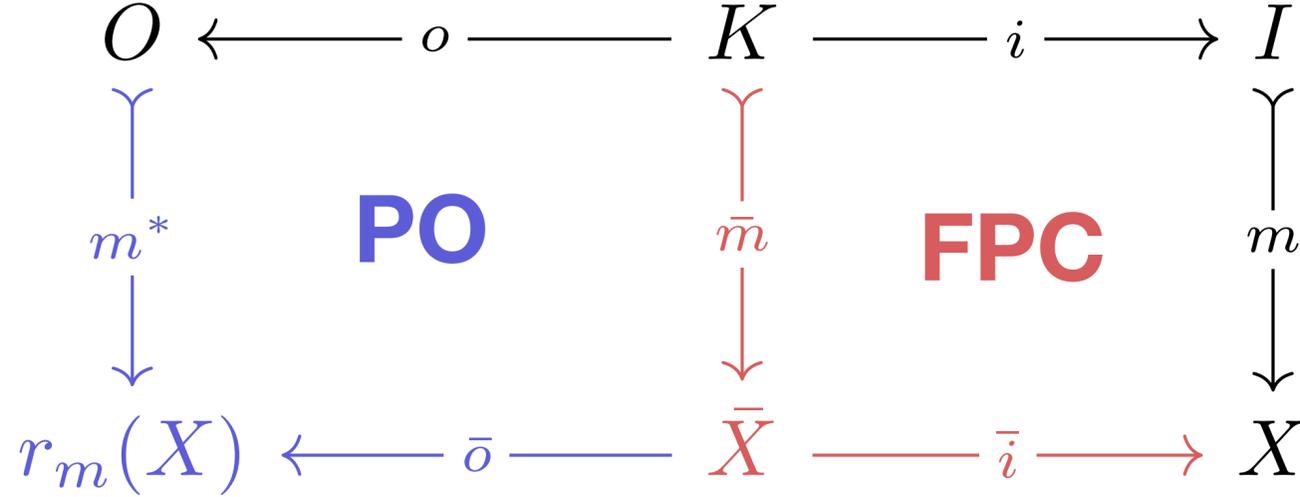
$(\mathcal{M}-)$ linear Double-Pushout (DPO)



non-linear Double-Pushout (DPO)

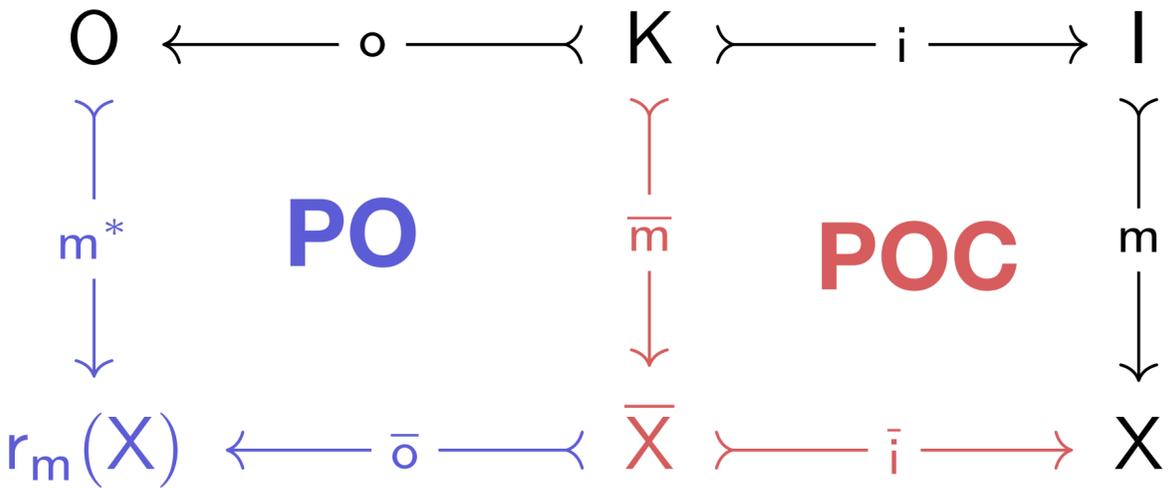


$(\mathcal{M}-)$ linear Sesqui-Pushout (SqPO)

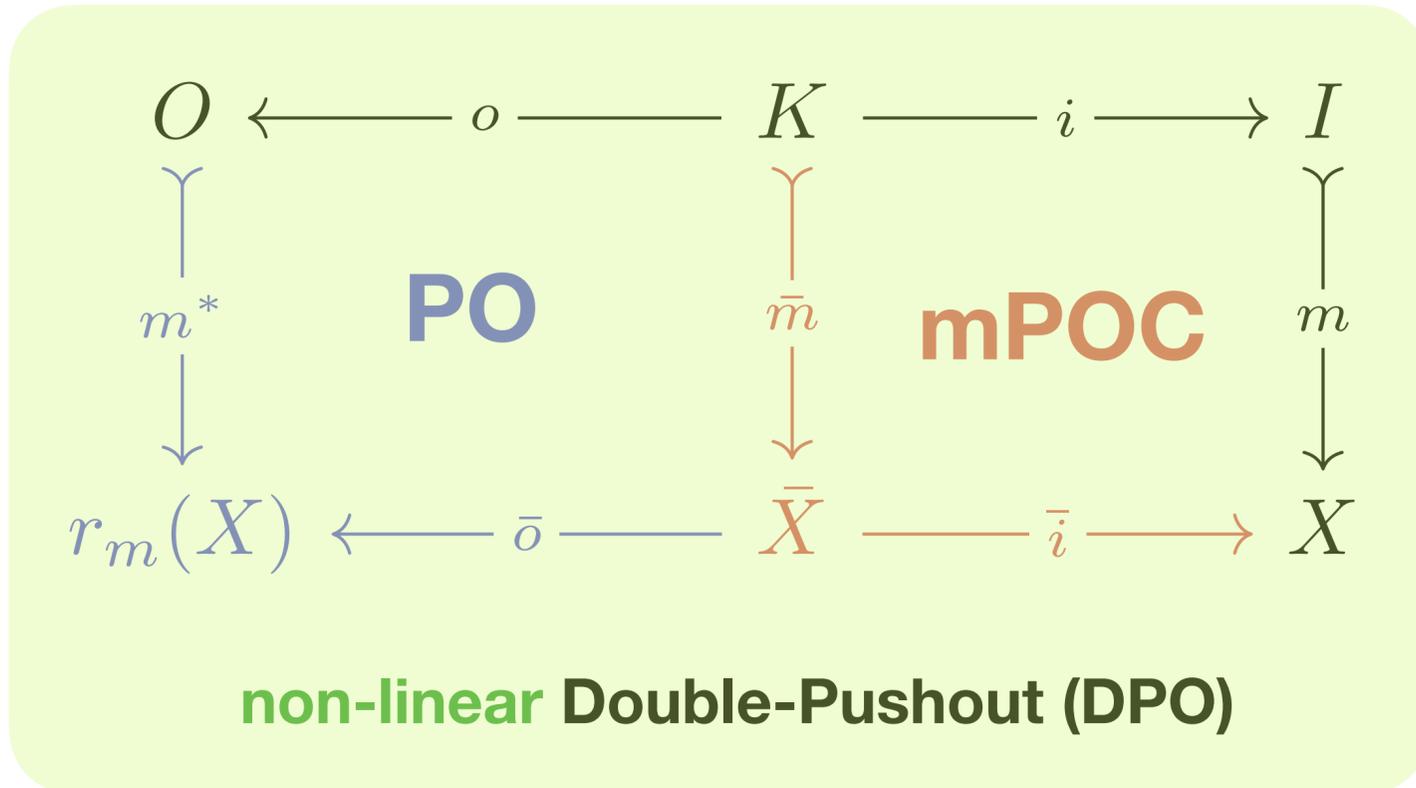


non-linear Sesqui-Pushout (SqPO)

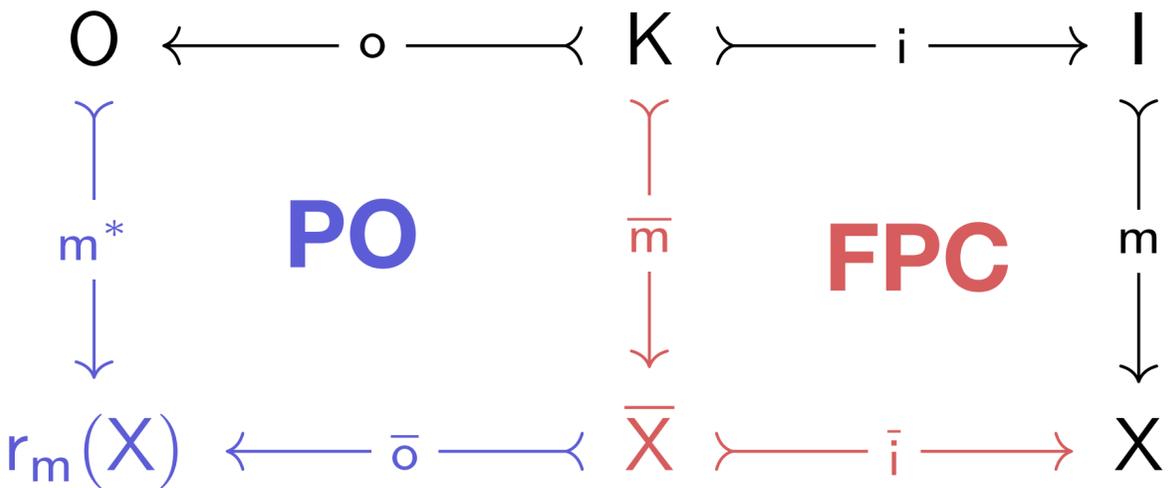
Four flavours of categorical rewriting semantics



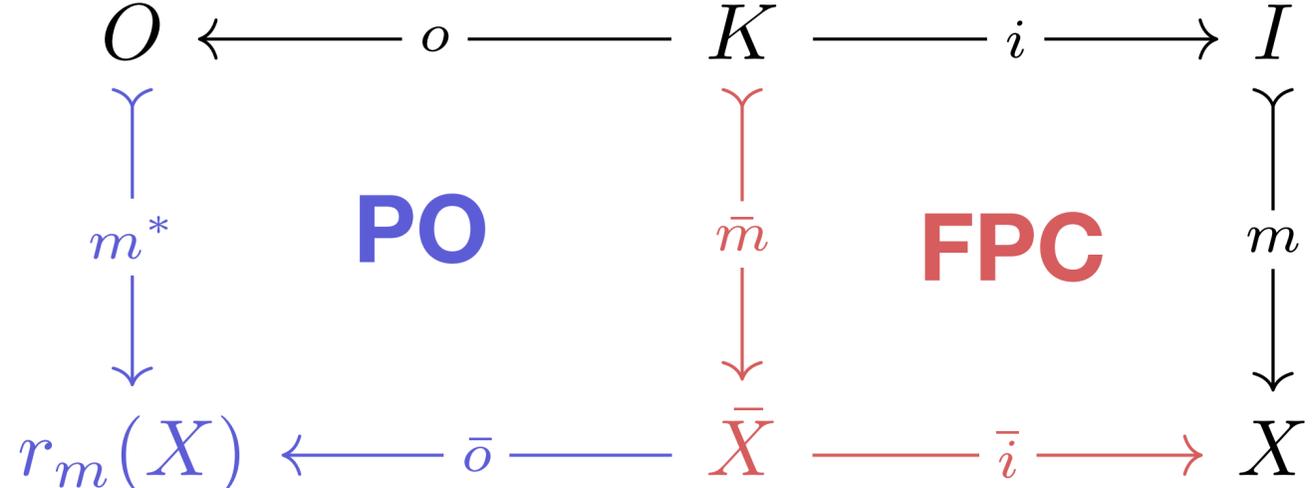
$(\mathcal{M}-)$ linear Double-Pushout (DPO)



non-linear Double-Pushout (DPO)

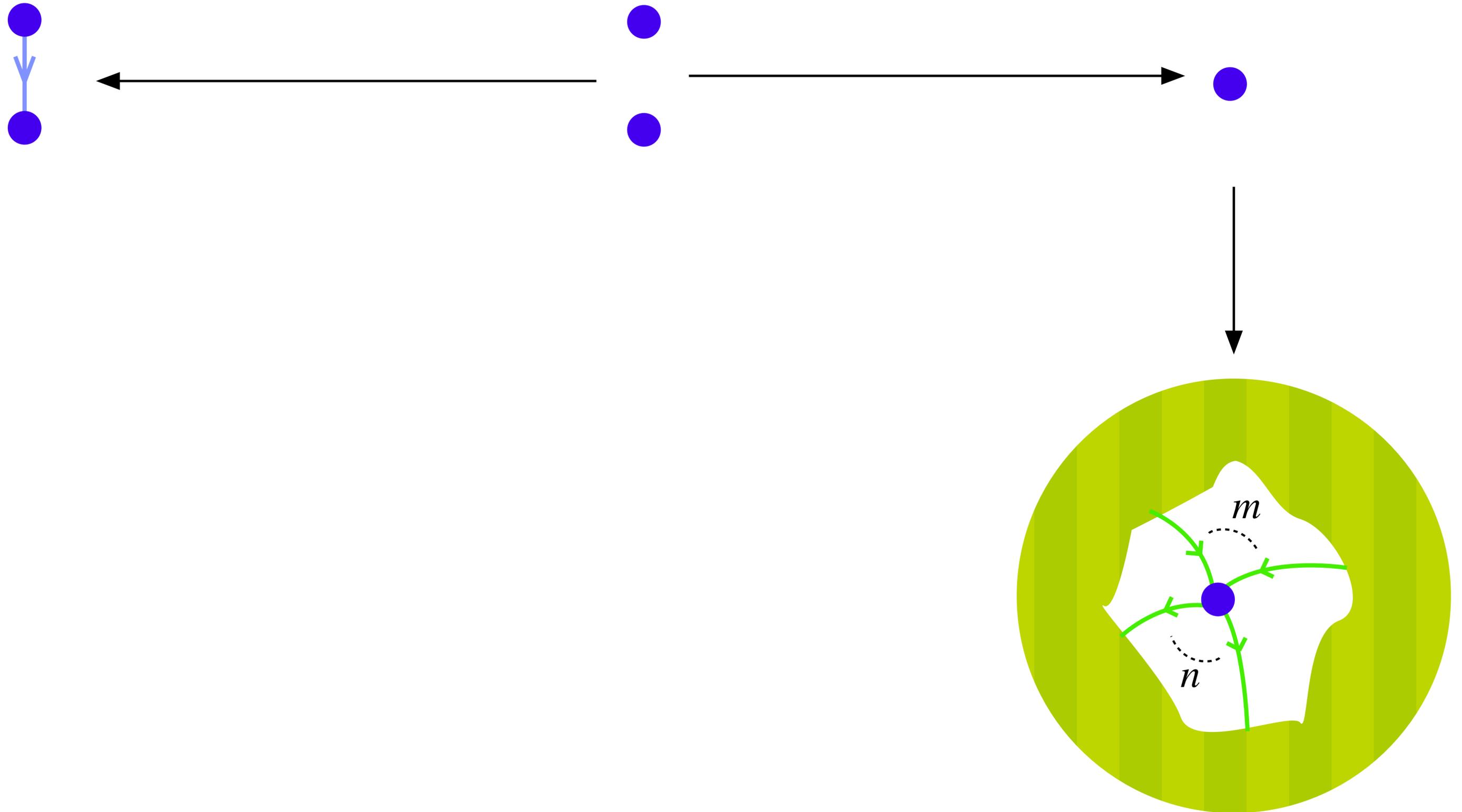


$(\mathcal{M}-)$ linear Sesqui-Pushout (SqPO)

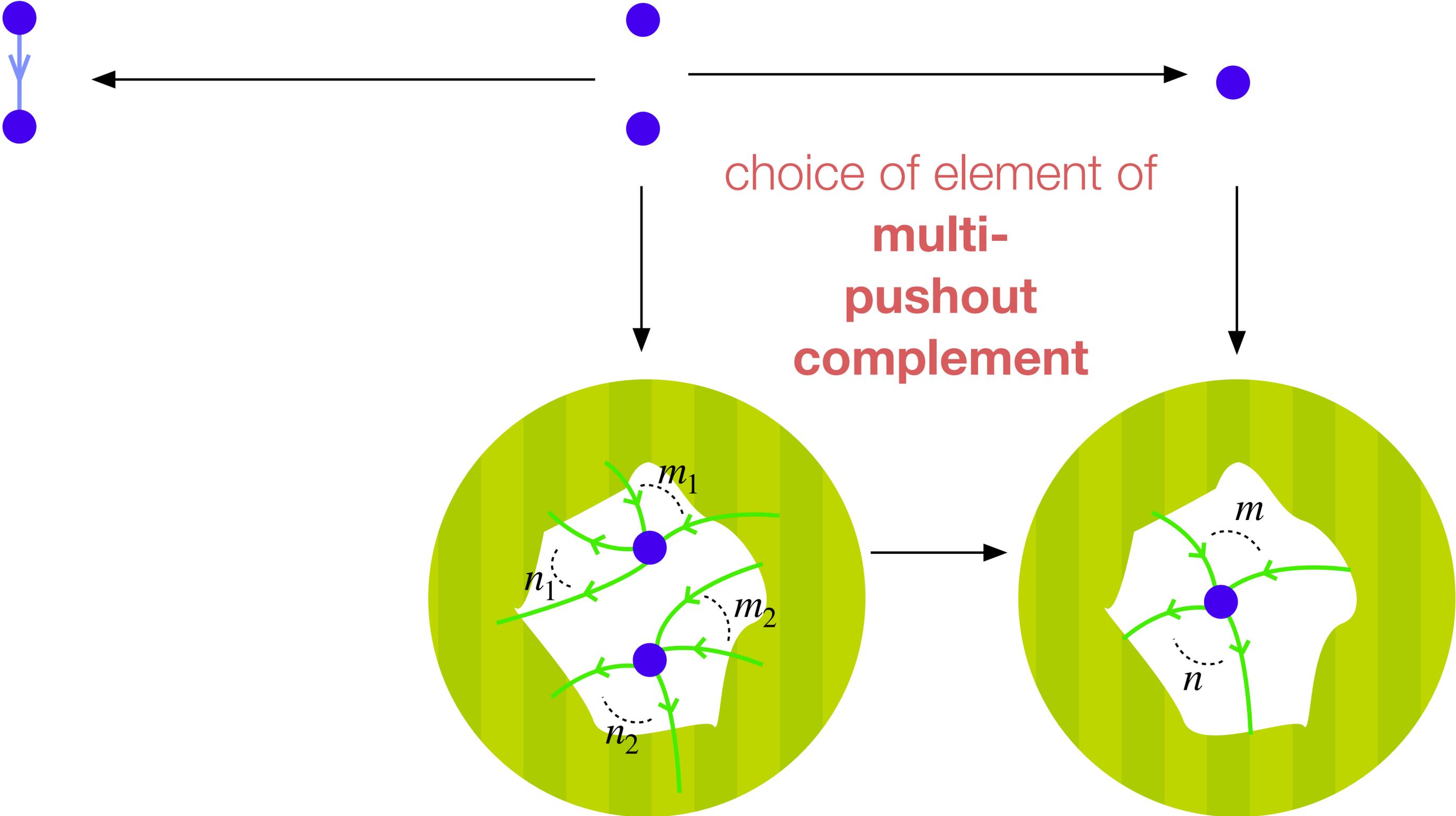


non-linear Sesqui-Pushout (SqPO)

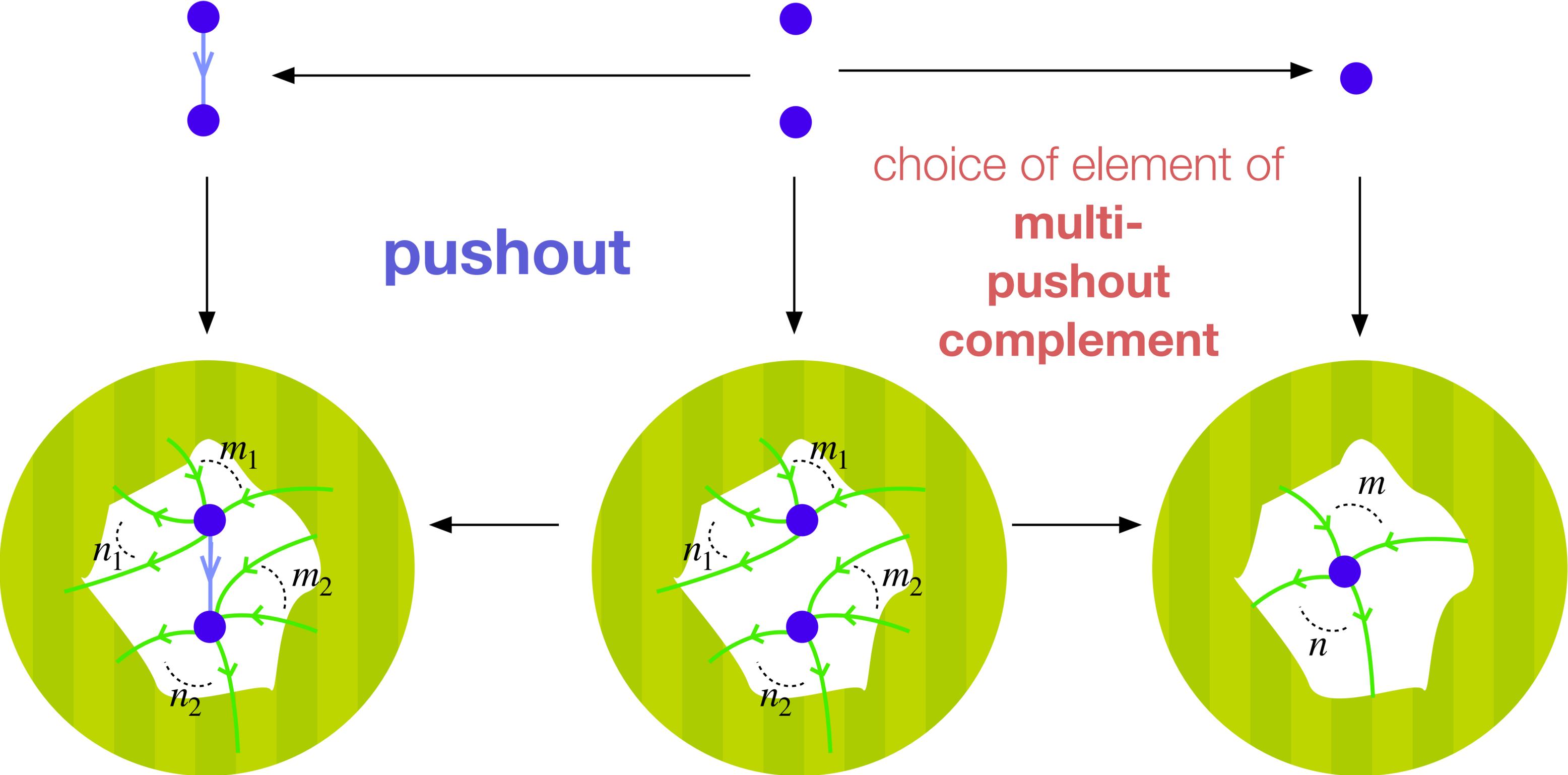
Cloning in **non-linear Double Pushout (DPO)** rewriting



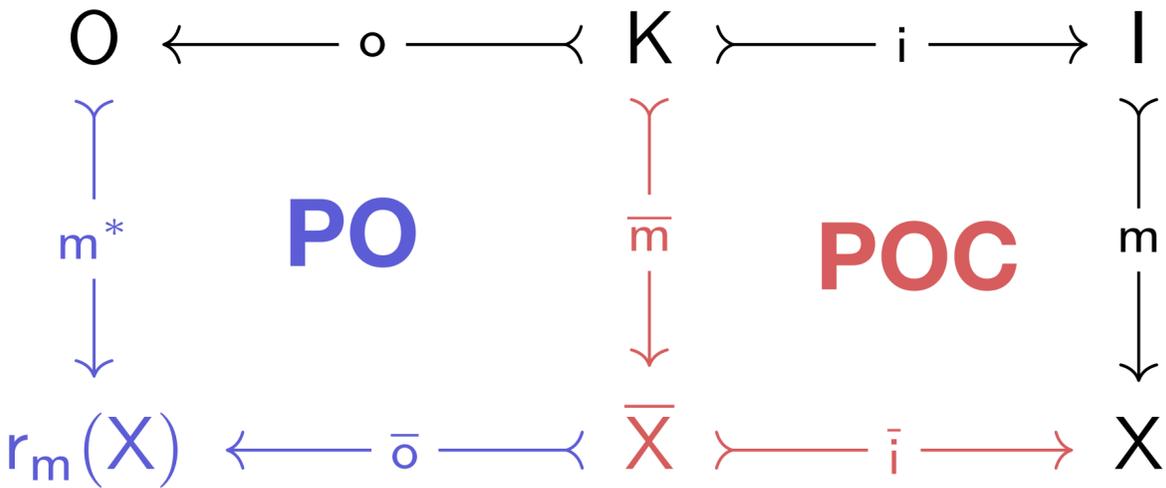
Cloning in **non-linear Double Pushout (DPO)** rewriting



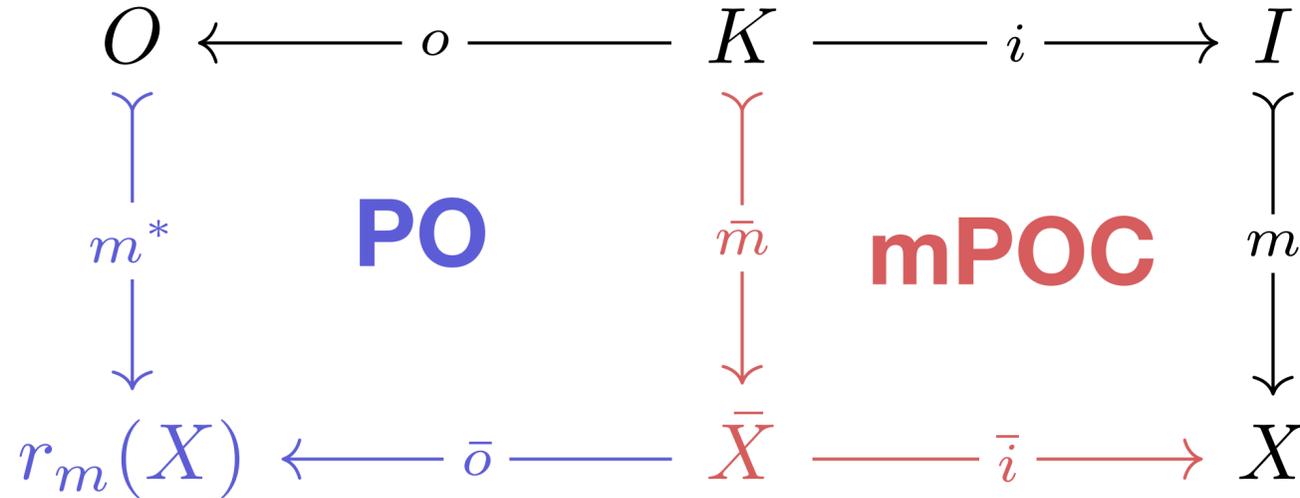
Cloning in **non-linear Double Pushout (DPO)** rewriting



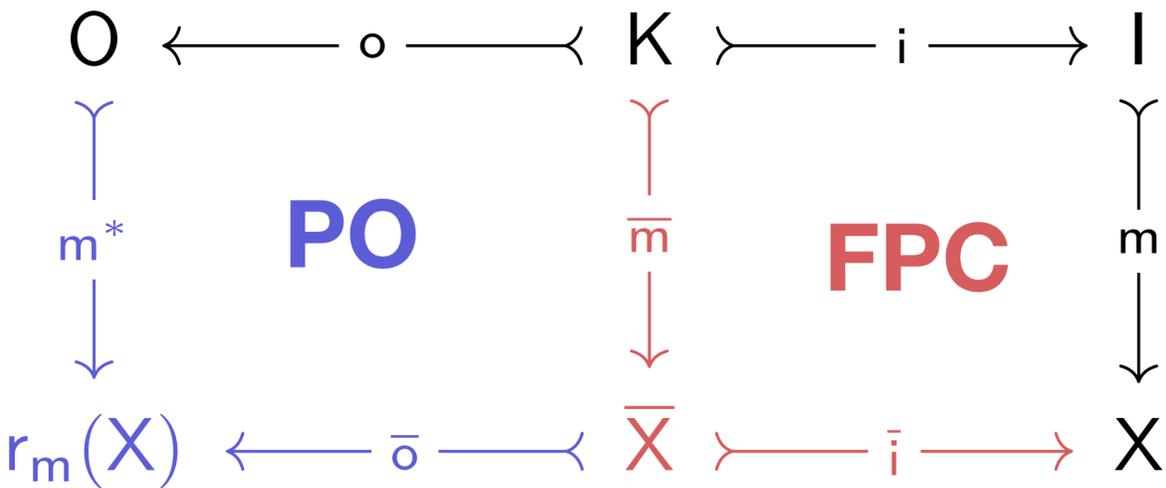
Four flavours of categorical rewriting semantics



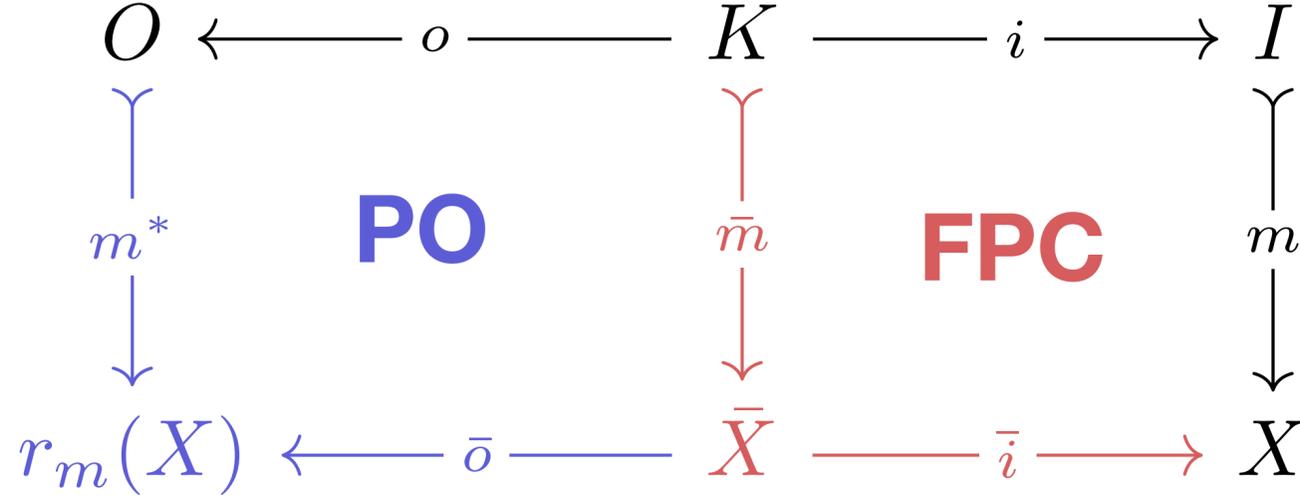
(M-) linear Double-Pushout (DPO)



non-linear Double-Pushout (DPO)



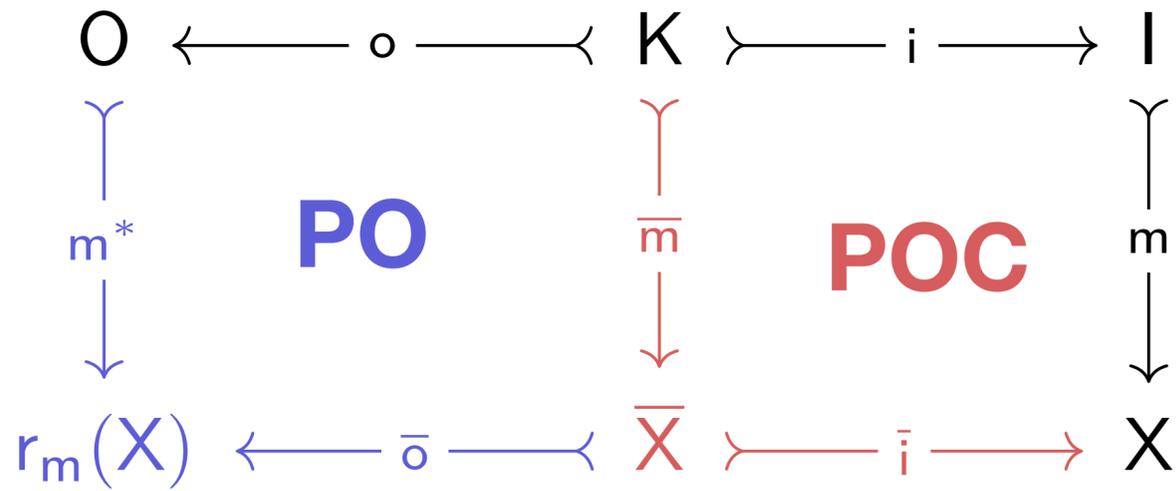
(M-) linear Sesqui-Pushout (SqPO)



non-linear Sesqui-Pushout (SqPO)

Four flavours of categorical rewriting semantics

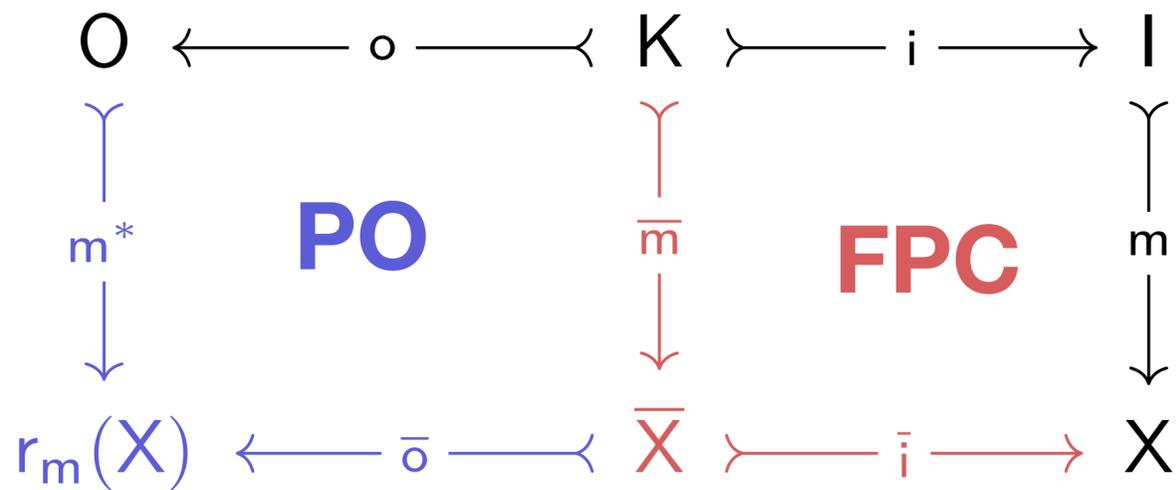
\mathcal{M} -adhesive categories
(+ some extra assumptions...)



PO

POC

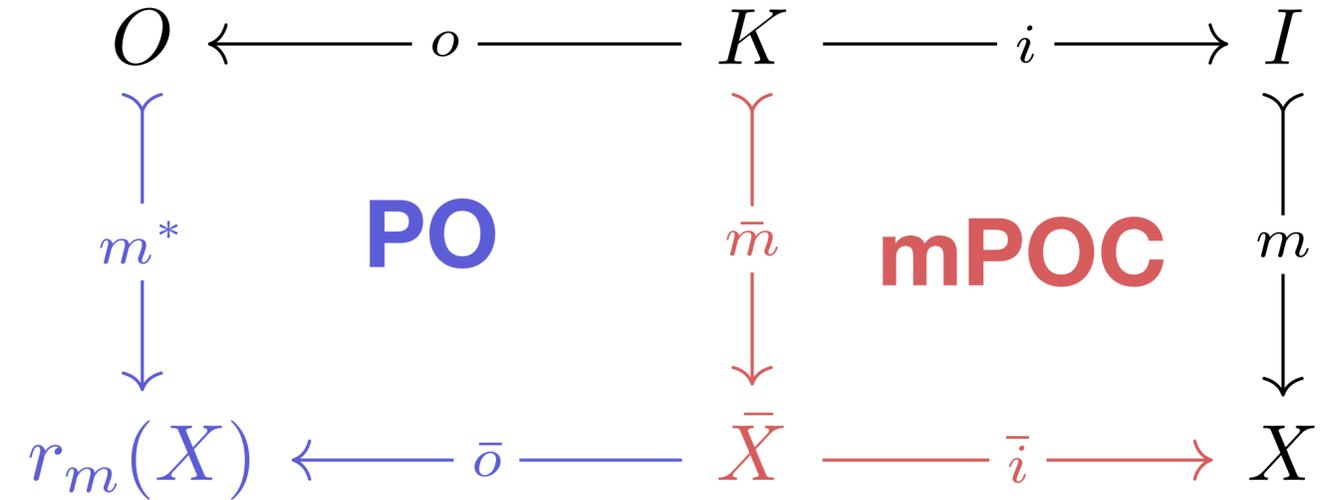
$(\mathcal{M}-)$ linear Double-Pushout (DPO)



PO

FPC

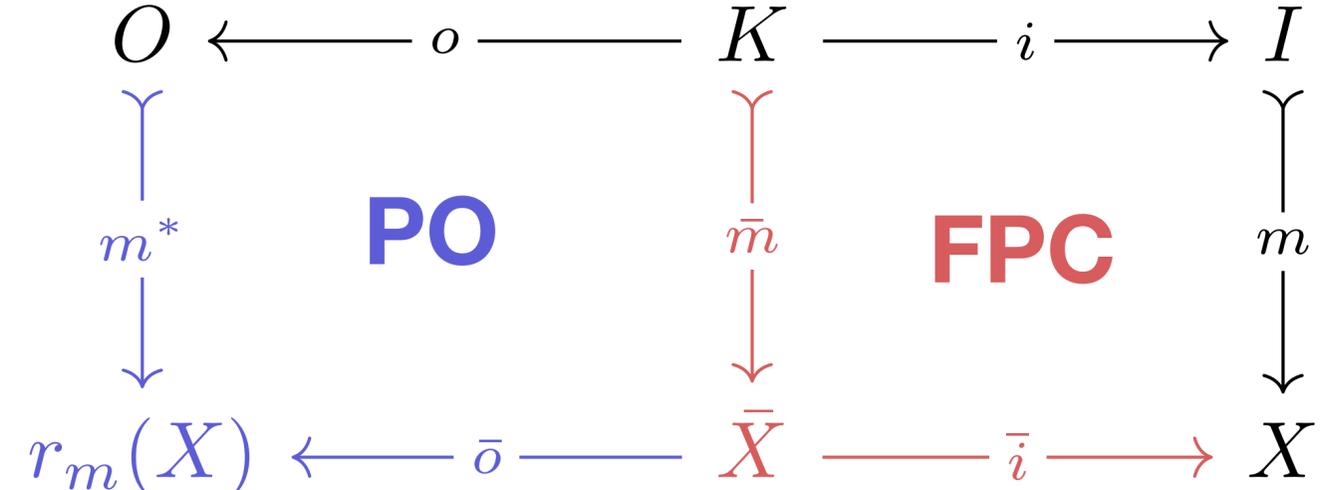
$(\mathcal{M}-)$ linear Sesqui-Pushout (SqPO)



PO

mPOC

non-linear Double-Pushout (DPO)



PO

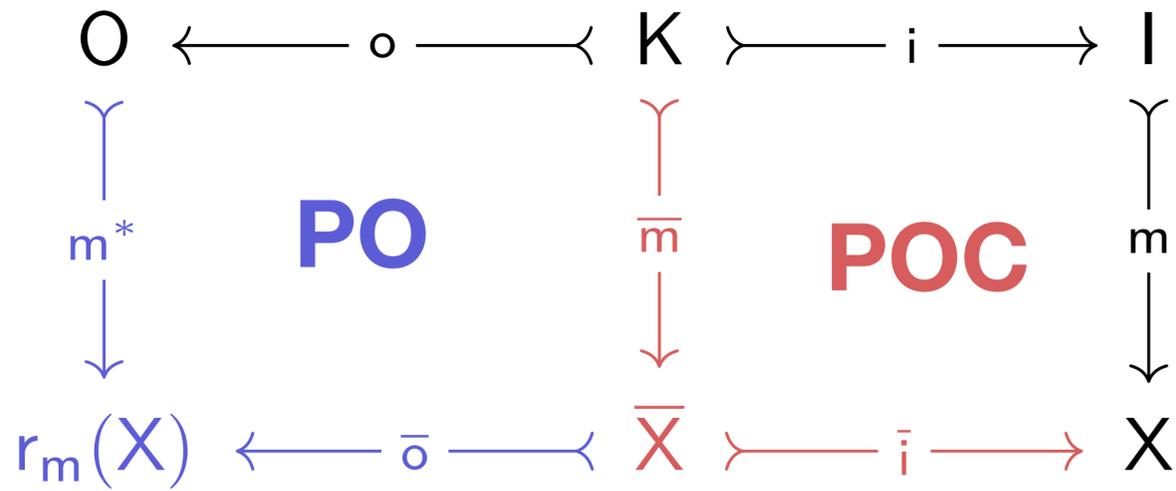
FPC

non-linear Sesqui-Pushout (SqPO)

Four flavours of categorical rewriting semantics

rm-adhesive categories

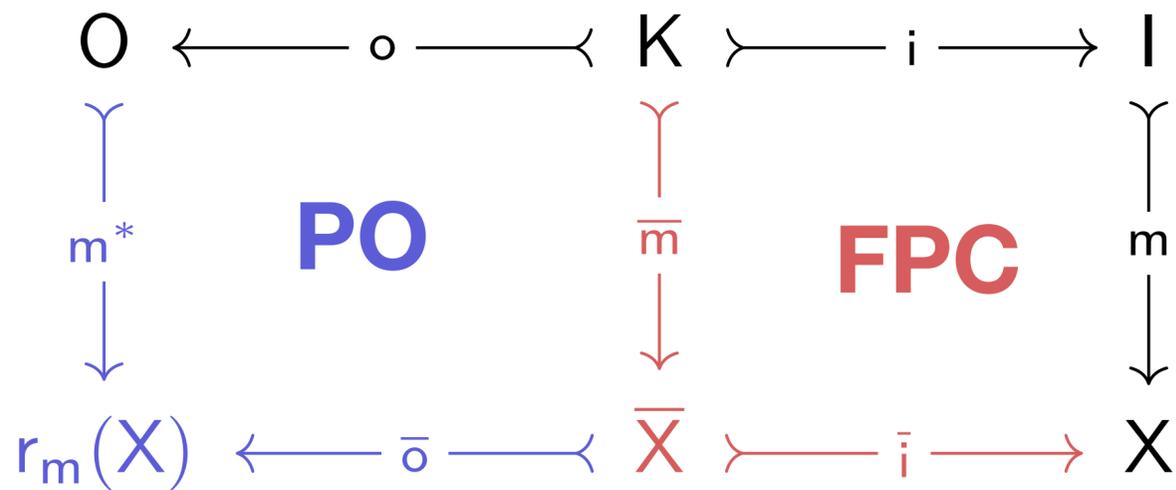
\mathcal{M} -adhesive categories
(+ some extra assumptions...)



PO

POC

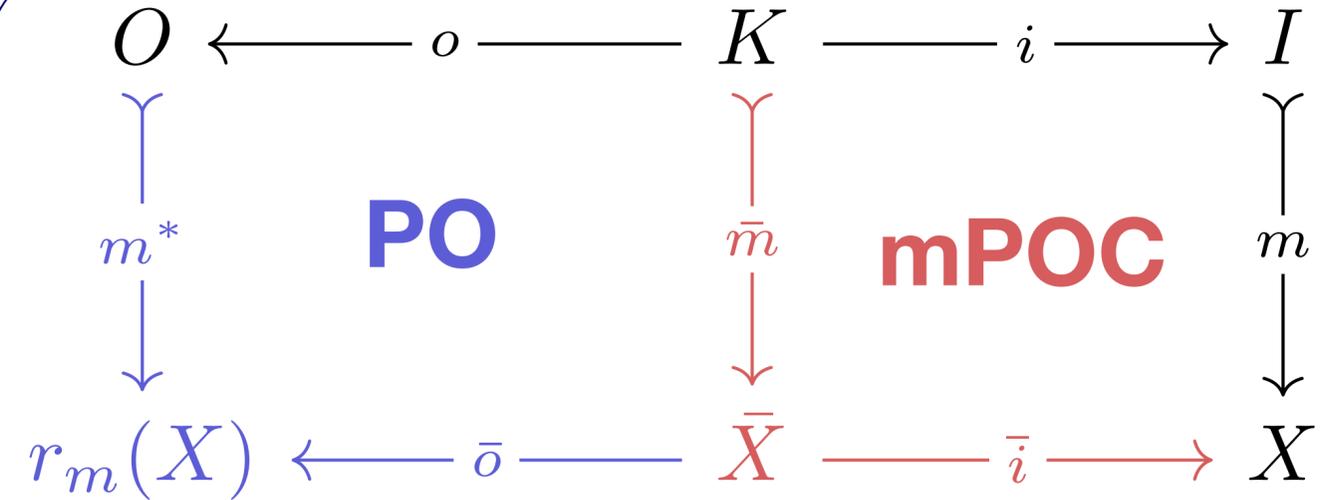
$(\mathcal{M}-)$ linear Double-Pushout (DPO)



PO

FPC

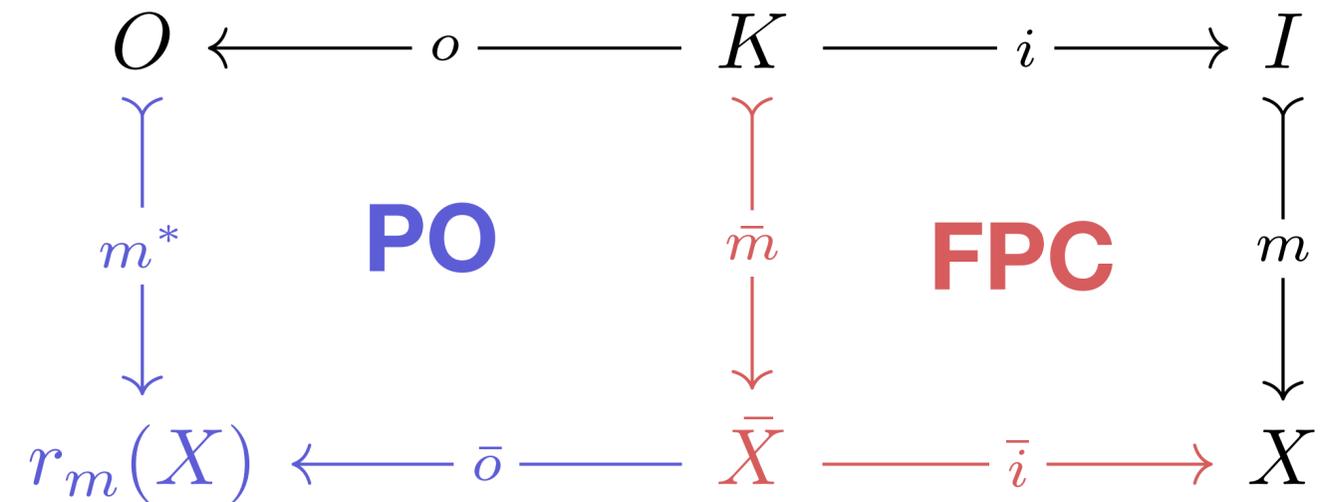
$(\mathcal{M}-)$ linear Sesqui-Pushout (SqPO)



PO

mPOC

non-linear Double-Pushout (DPO)



PO

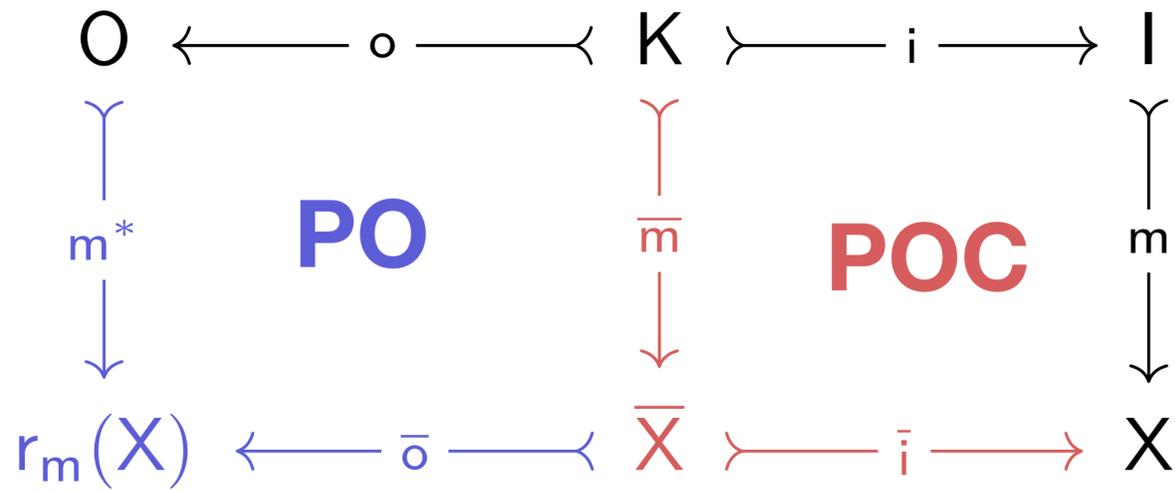
FPC

non-linear Sesqui-Pushout (SqPO)

Four flavours of categorical rewriting semantics

rm-adhesive categories

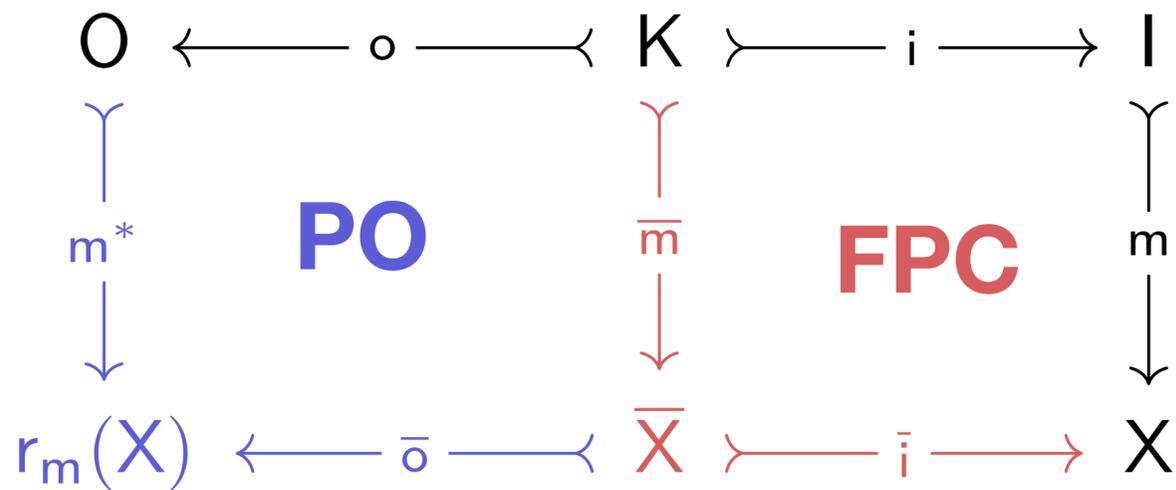
\mathcal{M} -adhesive categories
(+ some extra assumptions...)



PO

POC

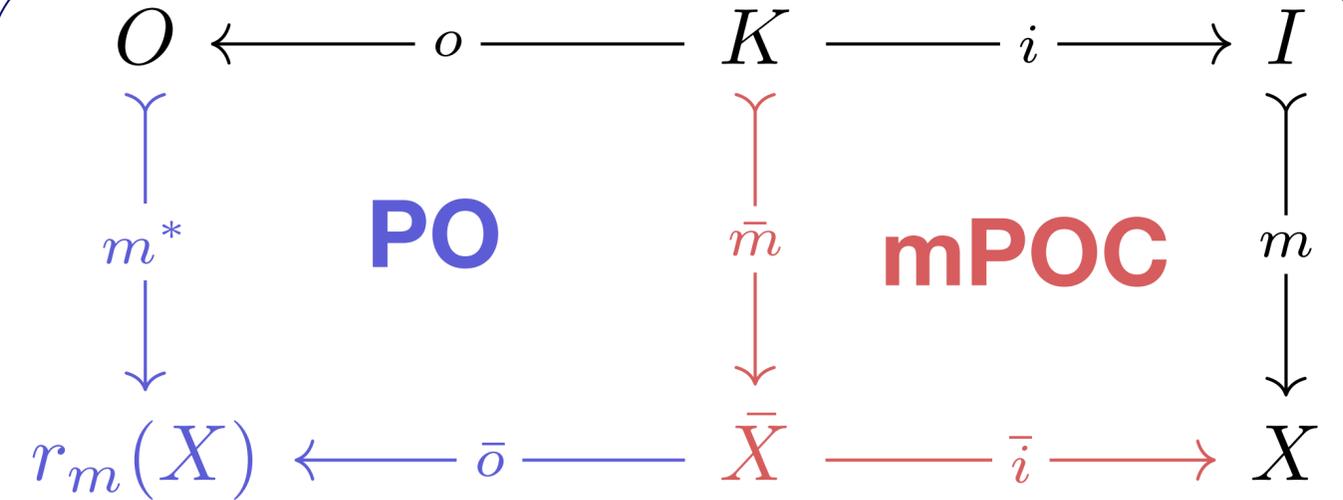
$(\mathcal{M}-)$ linear Double-Pushout (DPO)



PO

FPC

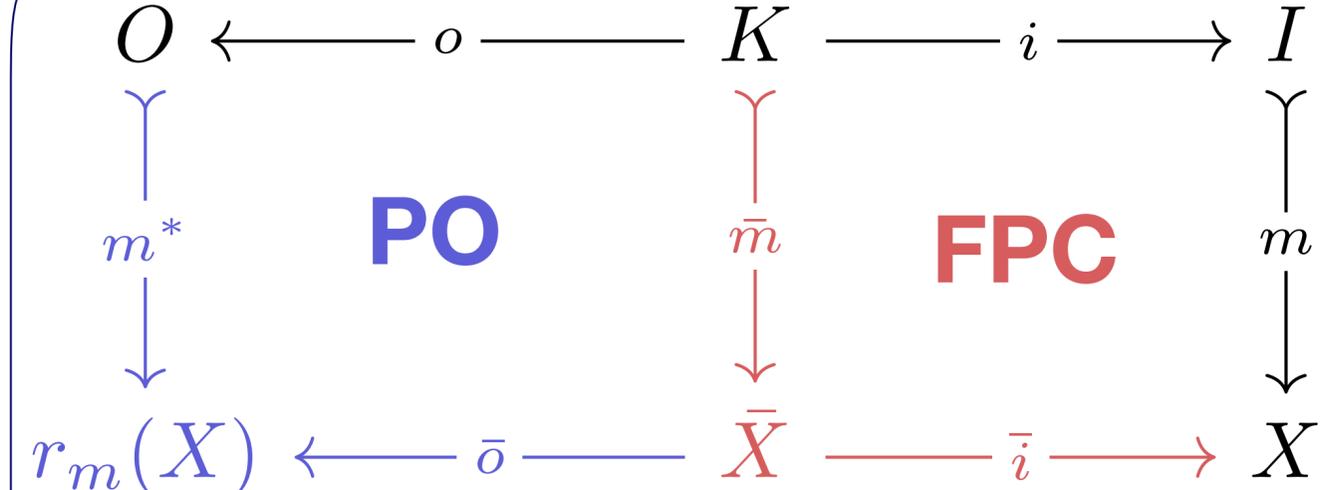
$(\mathcal{M}-)$ linear Sesqui-Pushout (SqPO)



PO

mPOC

non-linear Double-Pushout (DPO)



PO

FPC

non-linear Sesqui-Pushout (SqPO)

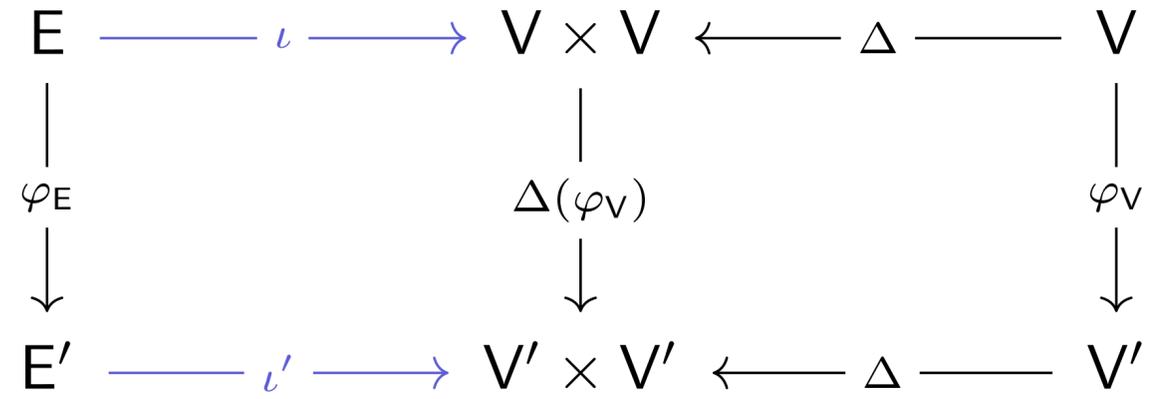
quasi-topoi

directed

undirected

multigraphs

simple graphs



Set / Δ

multigraphs

simple graphs



Set / Δ

directed

undirected

directed

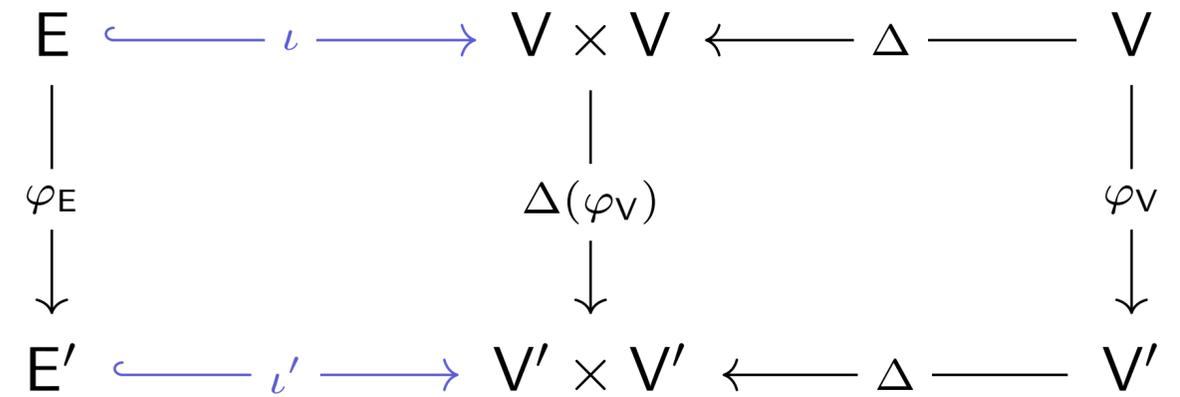
undirected

multigraphs



Set / Δ

simple graphs



Set // Δ

directed

undirected

multigraphs



Set / Δ

simple graphs



Set // Δ

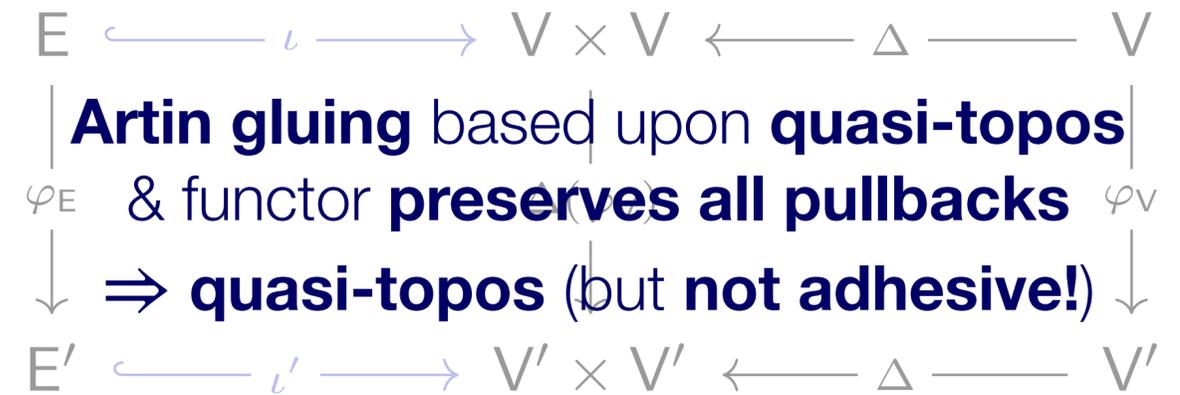
directed

multigraphs



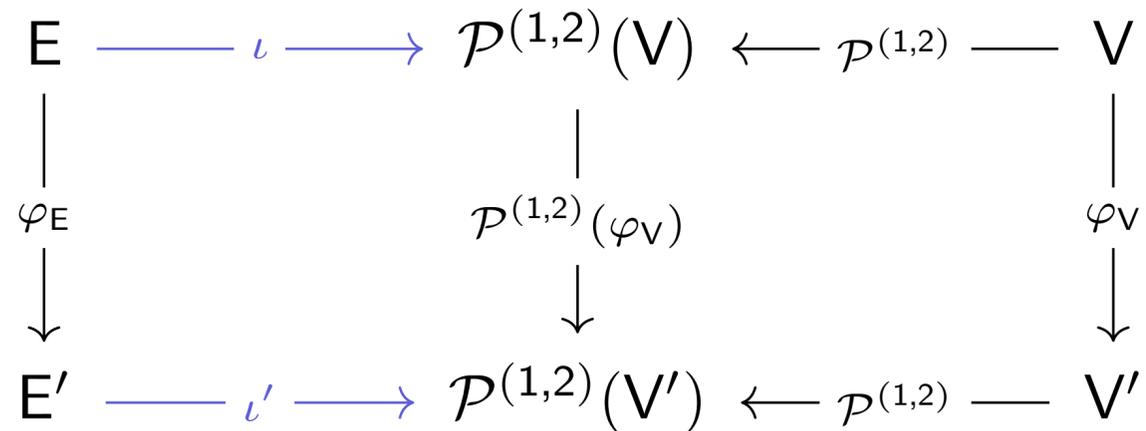
Set / Δ

simple graphs



Set // Δ

undirected



Set / $\mathcal{P}^{(1,2)}$

directed

multigraphs



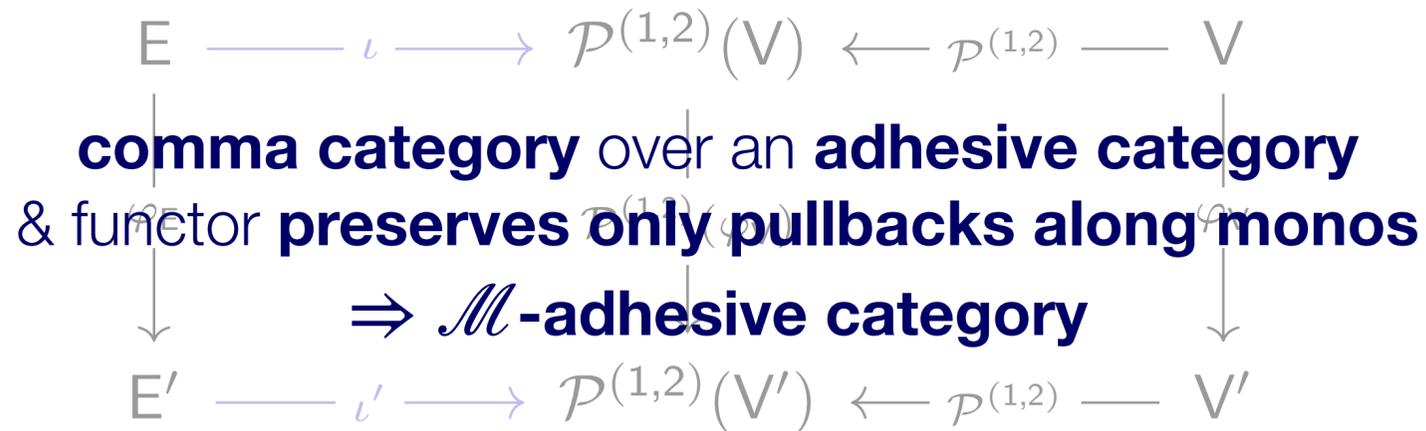
Set / Δ

simple graphs



Set // Δ

undirected



Set / $\mathcal{P}^{(1,2)}$

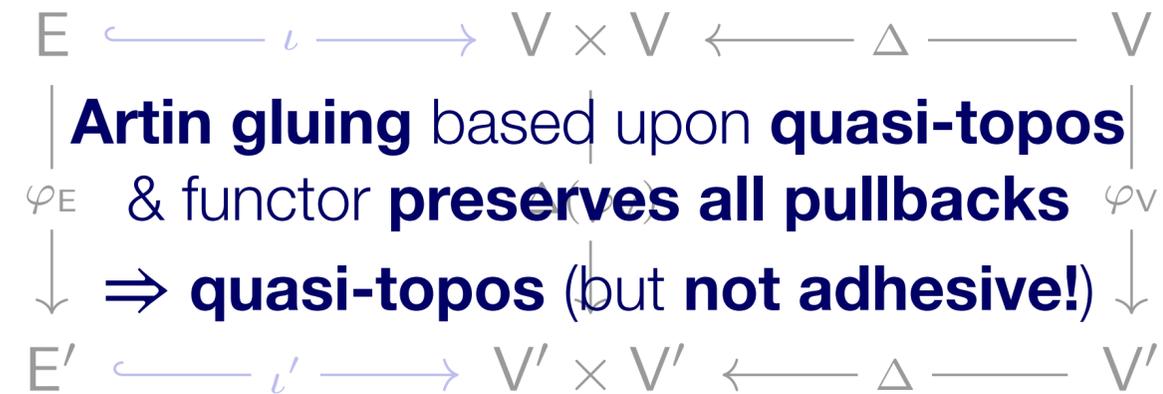
directed

multigraphs



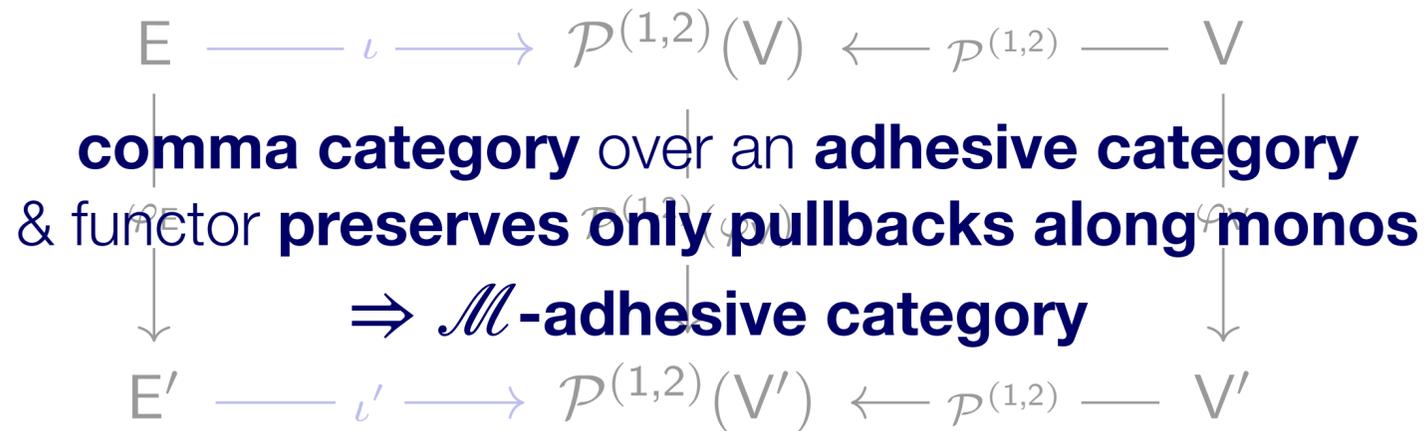
Set / Δ

simple graphs

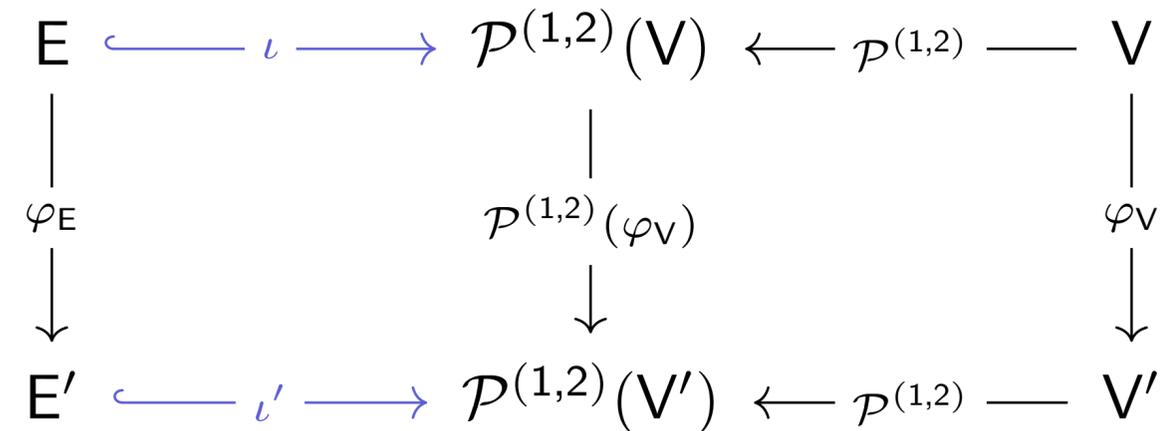


Set // Δ

undirected



Set / $\mathcal{P}^{(1,2)}$



Set // $\mathcal{P}^{(1,2)}$

directed

multigraphs



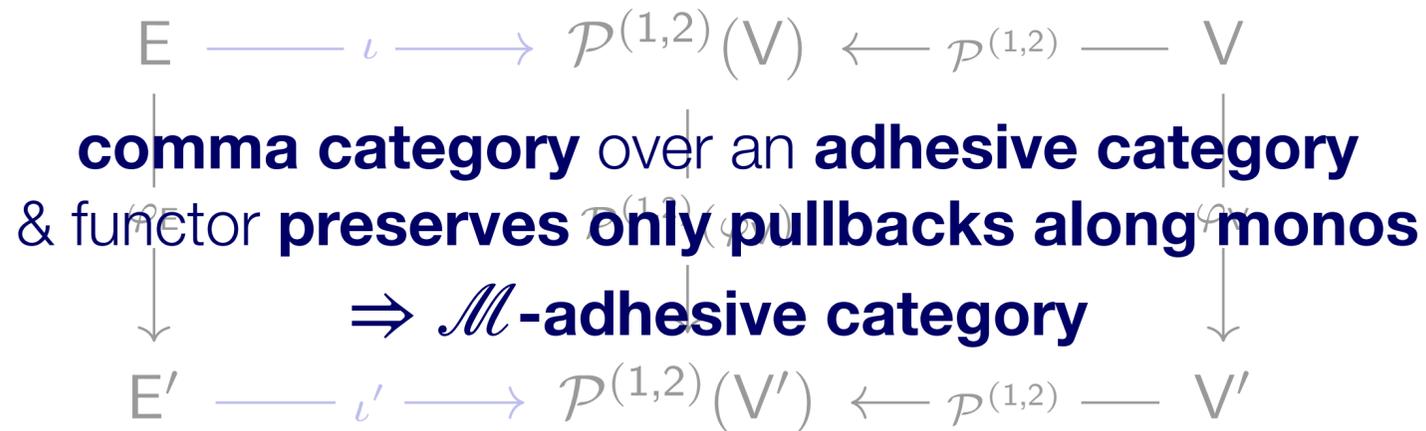
Set / Δ

simple graphs

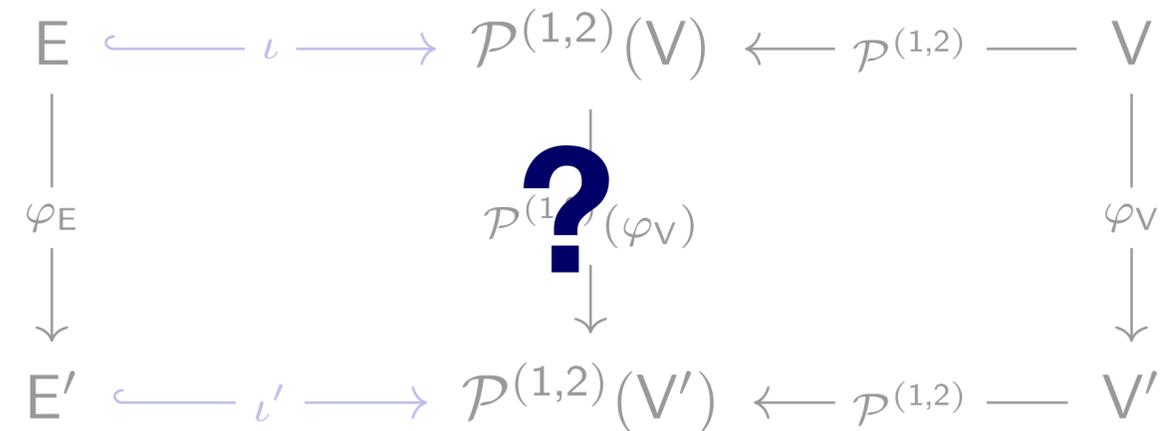


Set // Δ

undirected



Set / $\mathcal{P}^{(1,2)}$



Set // $\mathcal{P}^{(1,2)}$

Quasi-topoi — a natural setting for non-linear rewriting

Definition

A category \mathbf{C} is a **quasi-topos** iff

1. it has finite limits and colimits
2. it is locally Cartesian closed
3. it has a regular-subobject-classifier.

Johnstone, P.T., Lack, S., Sobociński, P.: Quasitoposes, Quasiadhesive Categories and Artin Glueing. In: Algebra and Coalgebra in Computer Science. LNCS, vol. 4624, pp. 312–326 (2007). https://doi.org/10.1007/978-3-540-73859-6_21

Quasi-topoi — a natural setting for non-linear rewriting

Proposition

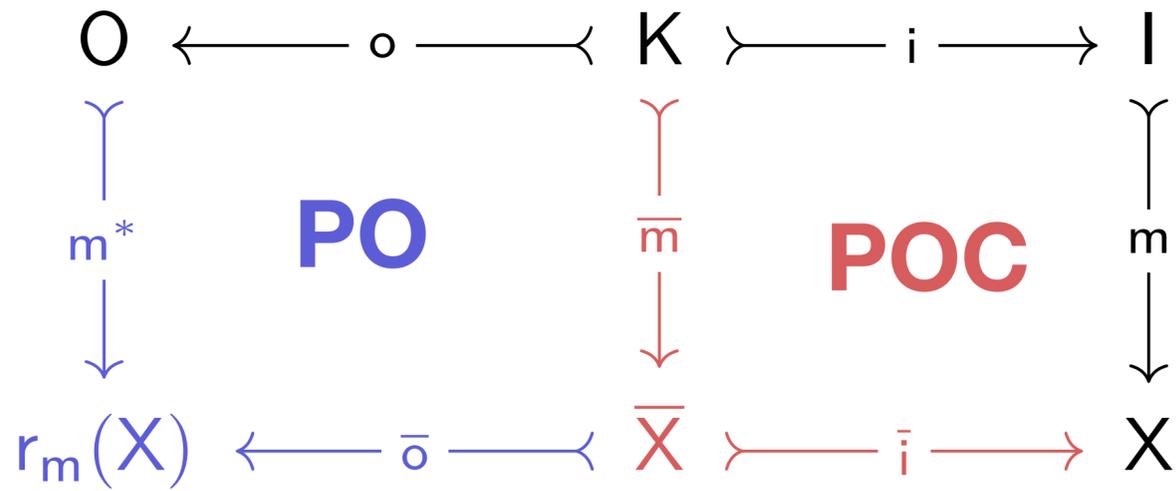
Every **quasi-topos** \mathbf{C} enjoys the following properties:

- It has (by definition) a **stable system of monics** $\mathcal{M} = \text{rm}(\mathbf{C})$ (the class of **regular monos**), which coincides with the class of **extremal monomorphisms**, i.e., if $m = f \circ e$ for $m \in \text{rm}(\mathbf{C})$ and $e \in \text{epi}(\mathbf{C})$, then $e \in \text{iso}(\mathbf{C})$.
- It has (by definition) a **\mathcal{M} -partial map classifier** (T, η) .
- It is **rm-quasi-adhesive**, i.e., it has **pushouts along regular monomorphisms**, these are **stable under pullbacks**, and **pushouts along regular monos are pullbacks**.
- It is **\mathcal{M} -adhesive**.
- For all pairs of composable morphisms $A \xrightarrow{f} B$ and $B \xrightarrow{m} C$ with $m \in \mathcal{M}$, there **exists a final pullback-complement (FPC)** $A \xrightarrow{n} F \xrightarrow{g} C$, and with $n \in \mathcal{M}$.
- It possesses an **epi- \mathcal{M} -factorization**: each morphism $A \xrightarrow{f} B$ factors as $f = m \circ e$, with morphisms $A \xrightarrow{e} B$ in $\text{epi}(\mathbf{C})$ and $B \xrightarrow{m} A$ in \mathcal{M} (uniquely up to isomorphism in B).

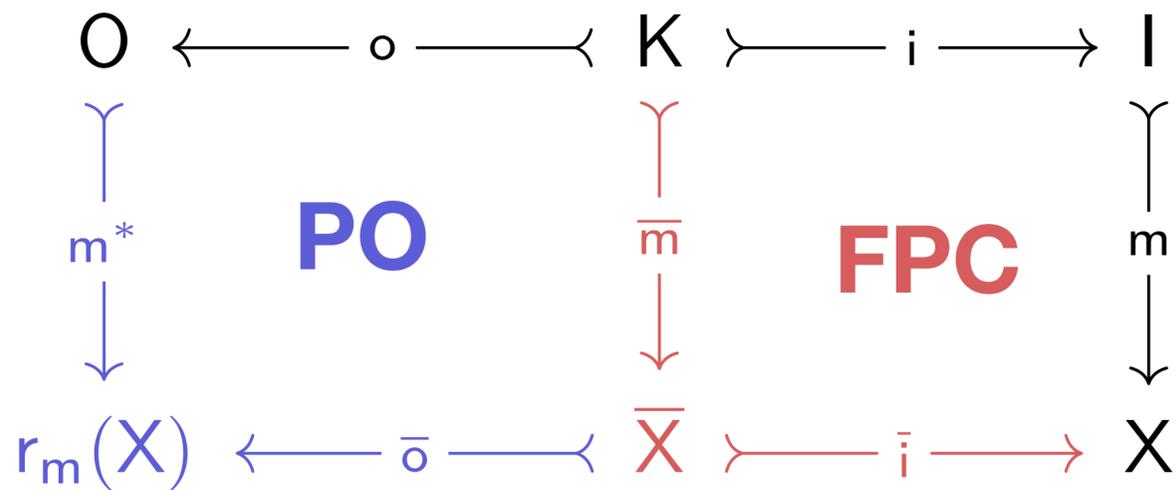
Case of **directed simple graphs** (a **quasi-topos!**)

rm-adhesive categories

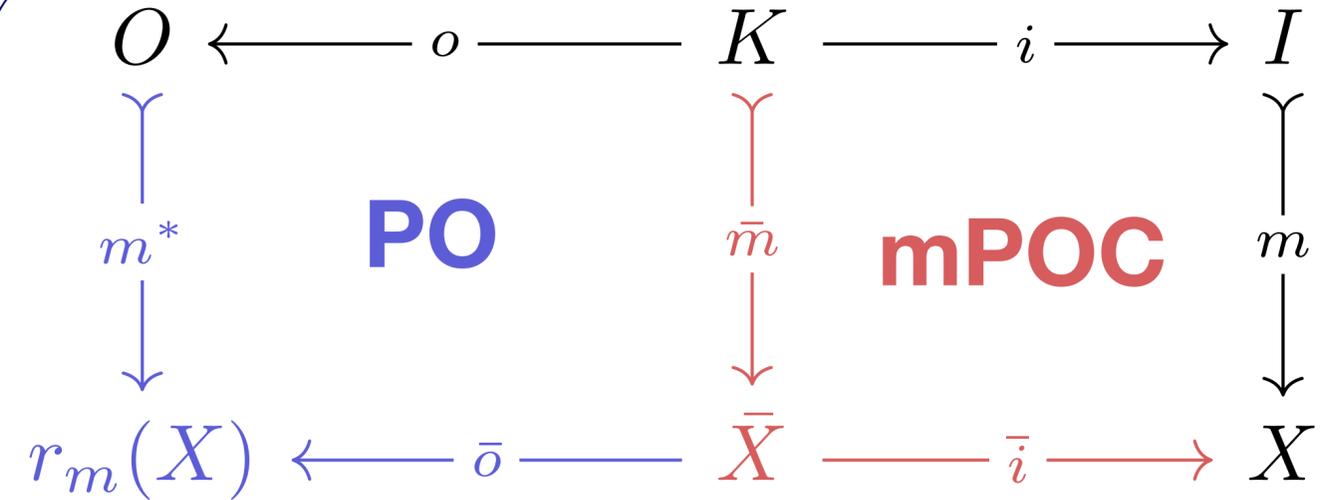
\mathcal{M} -adhesive categories
(+ some extra assumptions...)



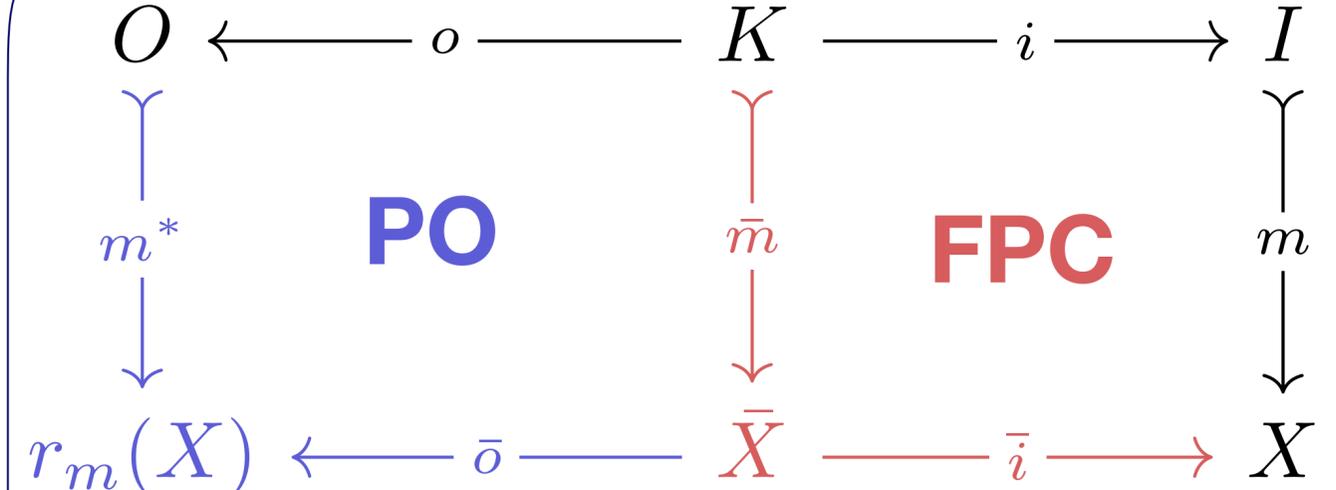
$(\mathcal{M}-)$ **linear** Double-Pushout (DPO)



$(\mathcal{M}-)$ **linear** Sesqui-Pushout (SqPO)



non-linear Double-Pushout (DPO)



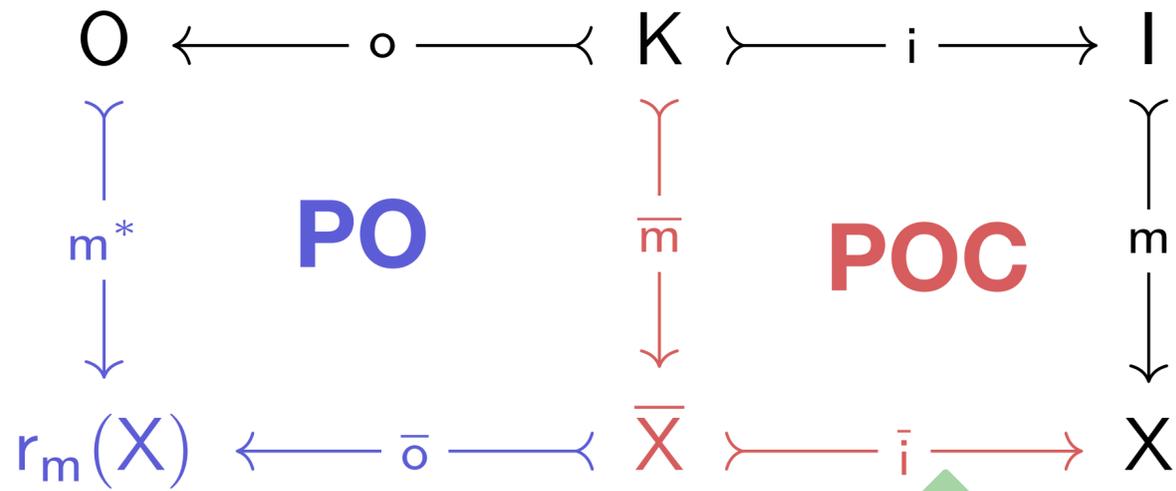
non-linear Sesqui-Pushout (SqPO)

quasi-topoi

Case of **directed simple graphs** (a **quasi-topos!**)

rm-adhesive categories

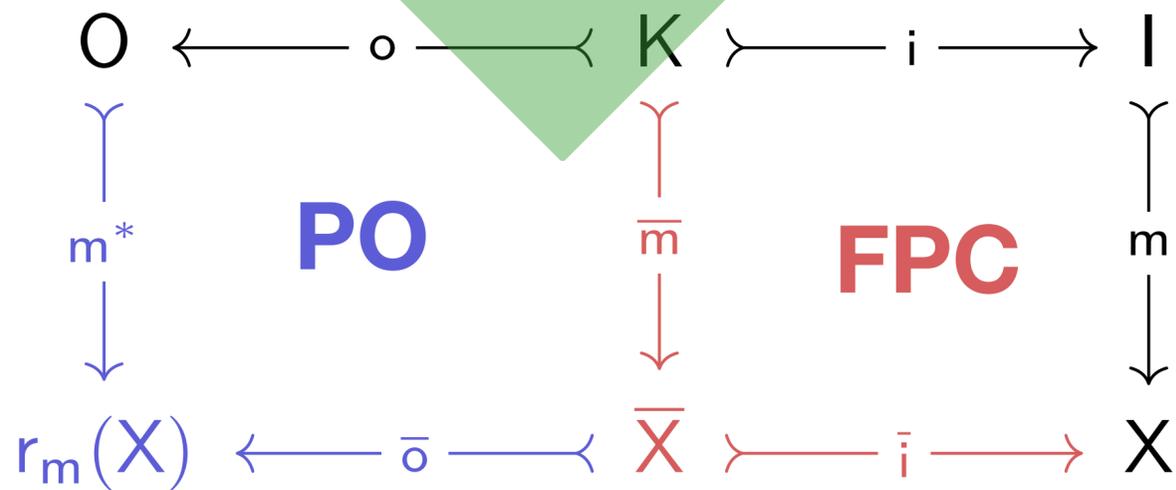
\mathcal{M} -adhesive categories
(+ some extra assumptions...)



PO

POC

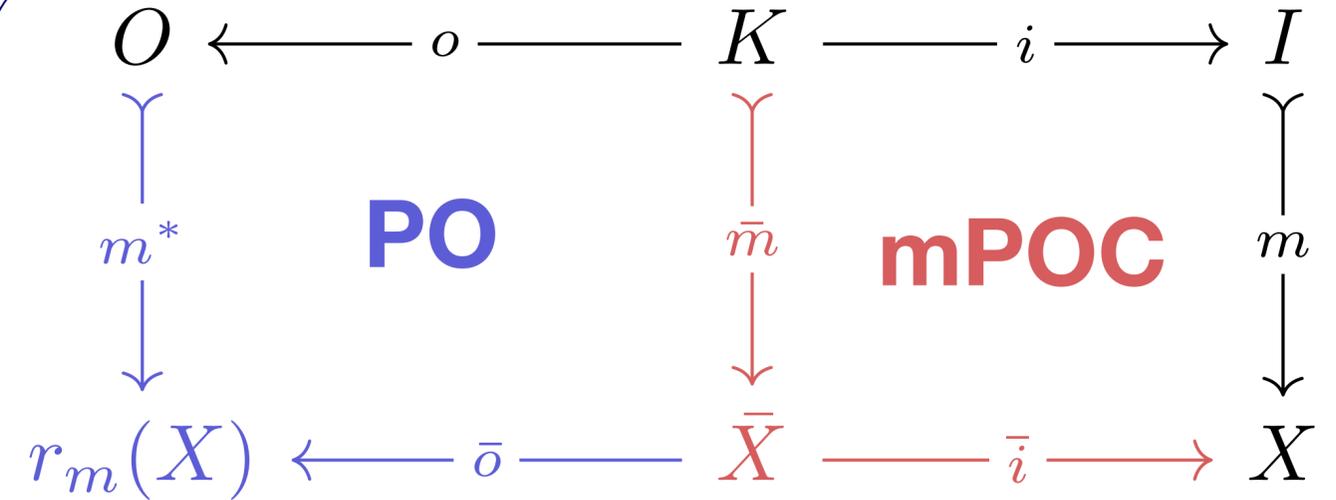
$(\mathcal{M}-)$ **linear Double-Pushout (DPO)**



PO

FPC

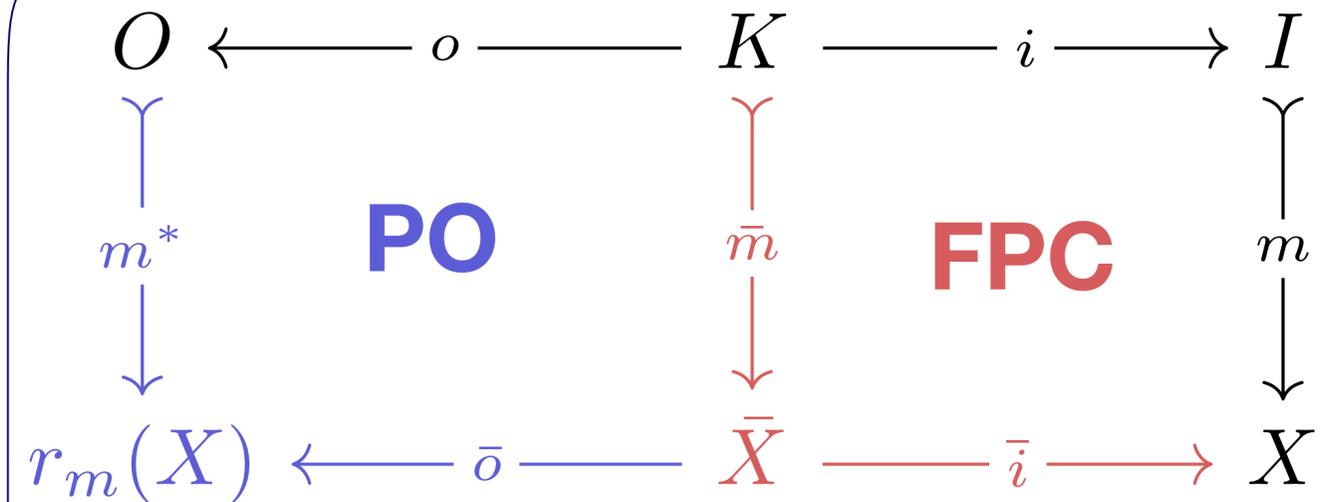
$(\mathcal{M}-)$ **linear Sesqui-Pushout (SqPO)**



PO

mPOC

non-linear Double-Pushout (DPO)



PO

FPC

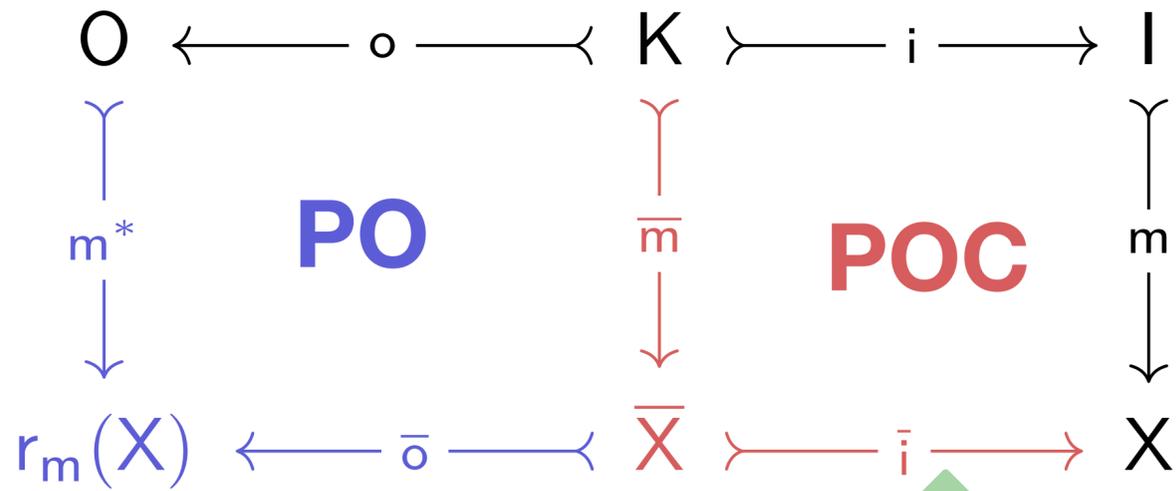
non-linear Sesqui-Pushout (SqPO)

quasi-topoi

Case of **directed simple graphs** (a **quasi-topos!**)

rm-adhesive categories

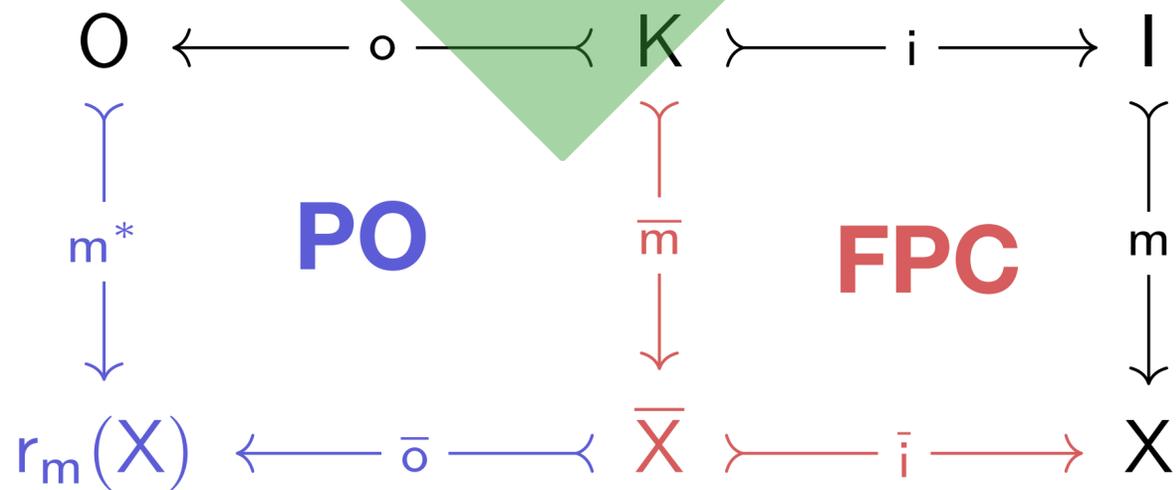
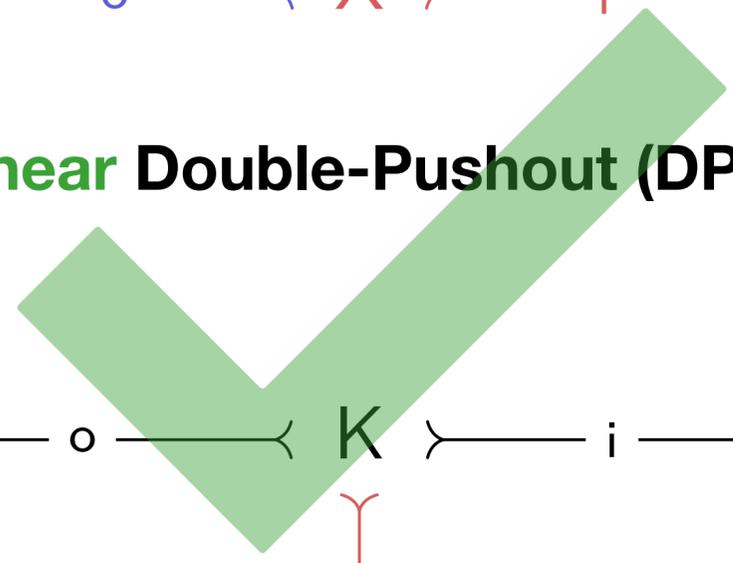
\mathcal{M} -adhesive categories
(+ some extra assumptions...)



PO

POC

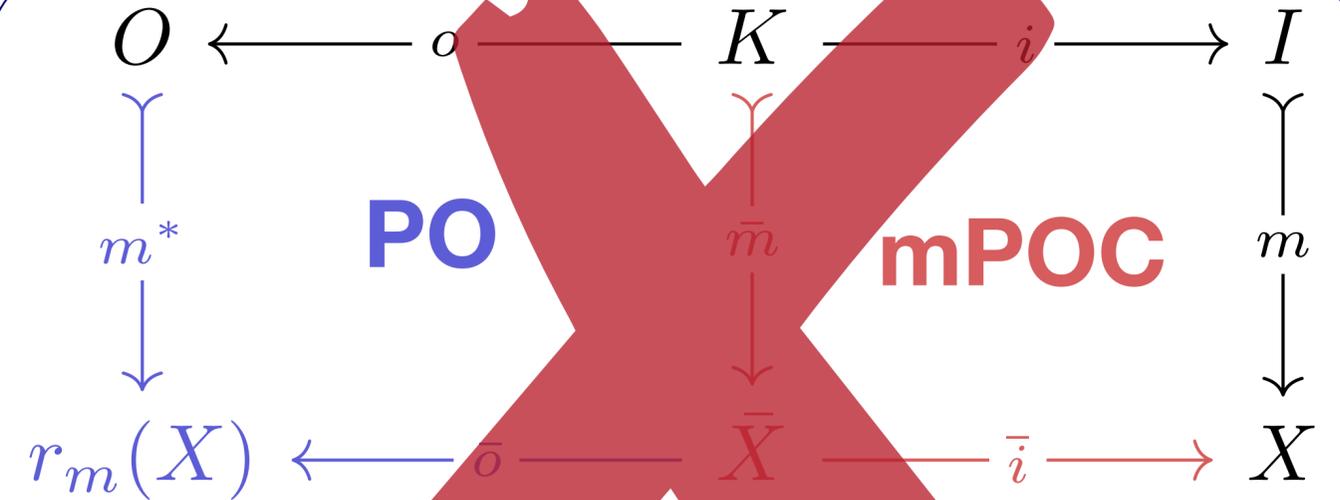
$(\mathcal{M}-)$ **linear Double-Pushout (DPO)**



PO

FPC

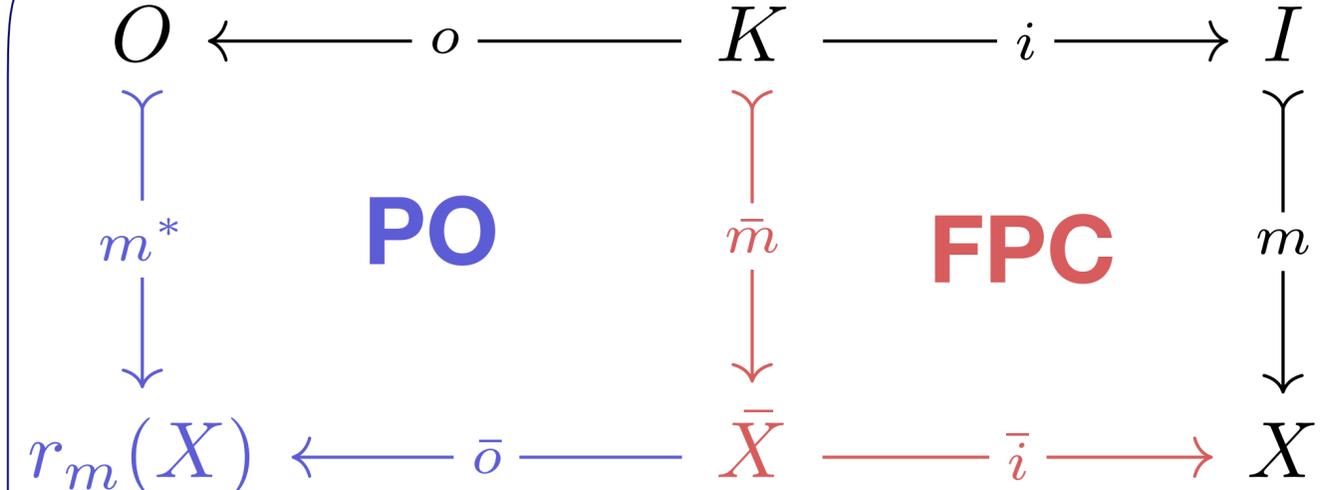
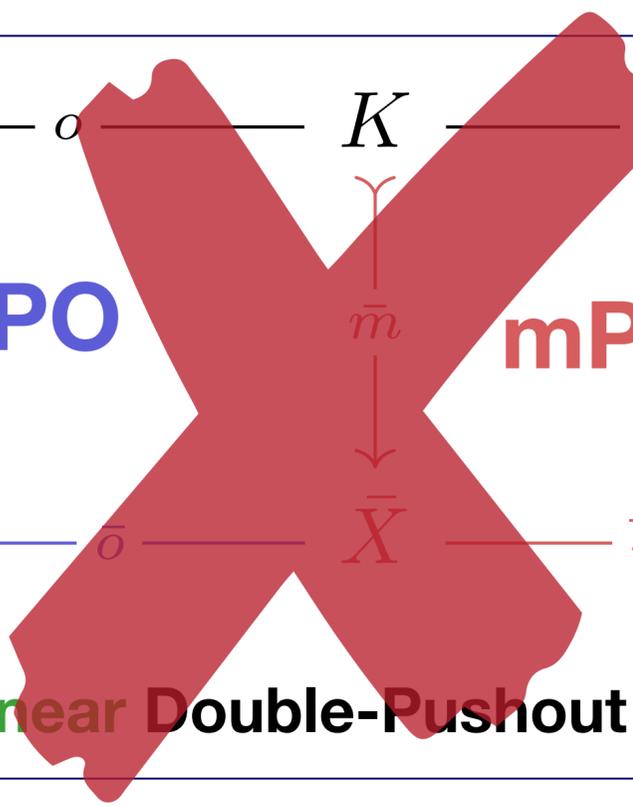
$(\mathcal{M}-)$ **linear Sesqui-Pushout (SqPO)**



PO

mPOC

non-linear Double-Pushout (DPO)



PO

FPC

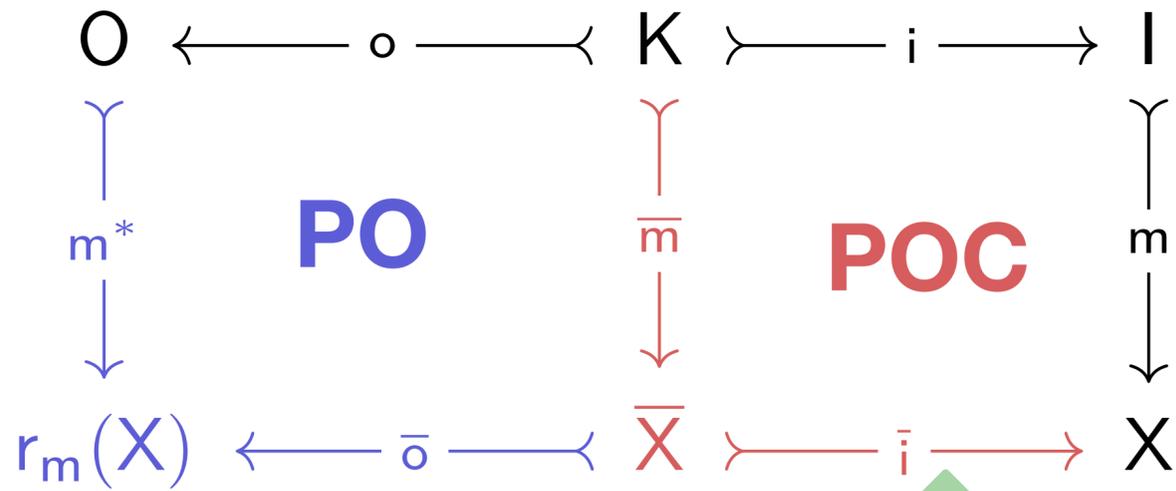
non-linear Sesqui-Pushout (SqPO)

quasi-topoi

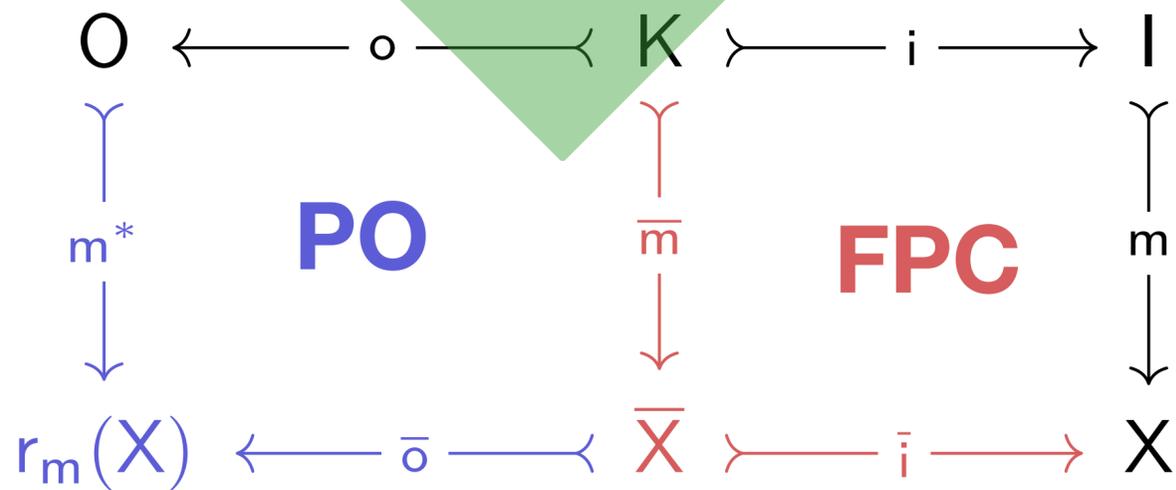
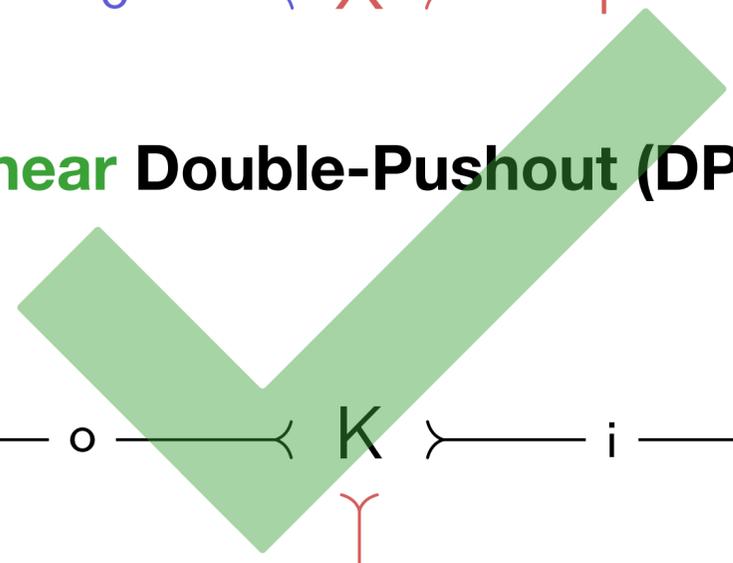
Case of **directed simple graphs** (a **quasi-topos!**)

rm-adhesive categories

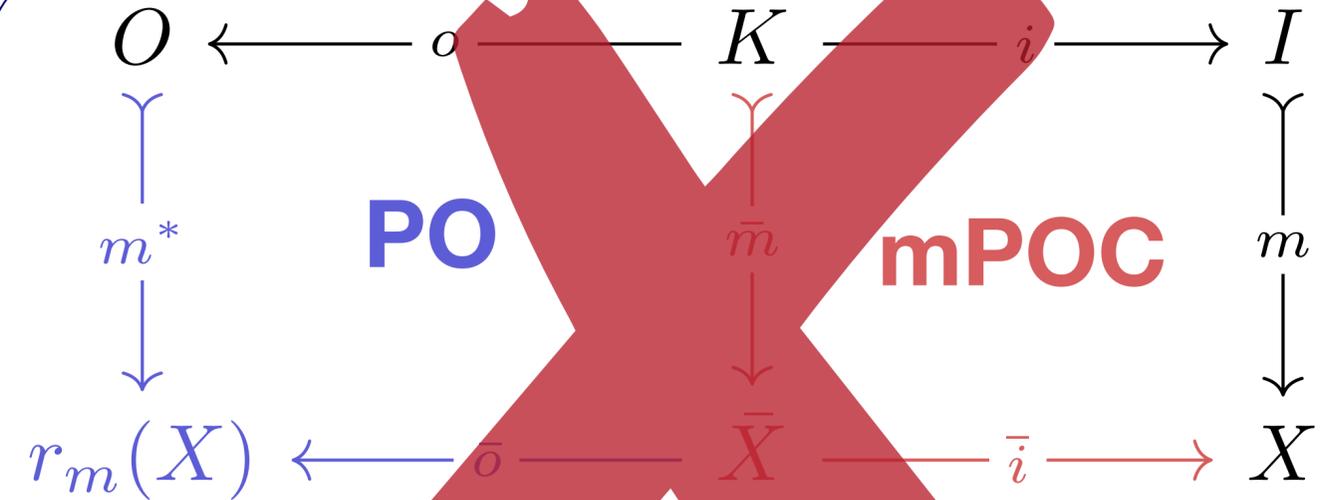
\mathcal{M} -adhesive categories
(+ some extra assumptions...)



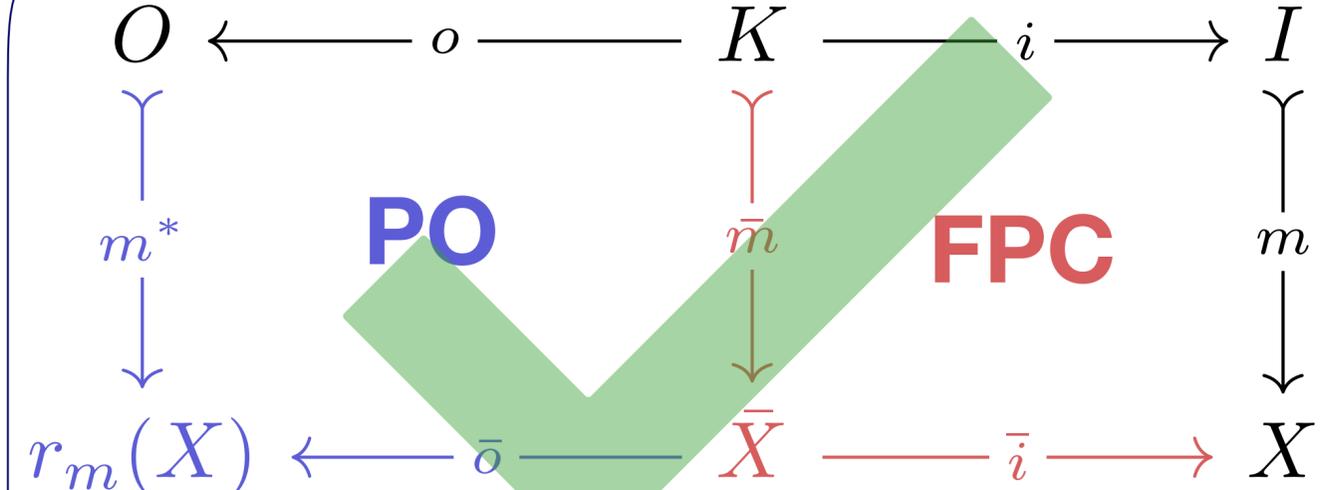
$(\mathcal{M}-)$ **linear Double-Pushout (DPO)**



$(\mathcal{M}-)$ **linear Sesqui-Pushout (SqPO)**



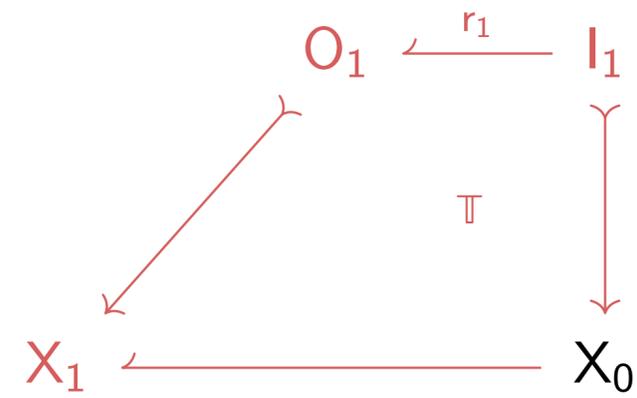
non-linear Double-Pushout (DPO)



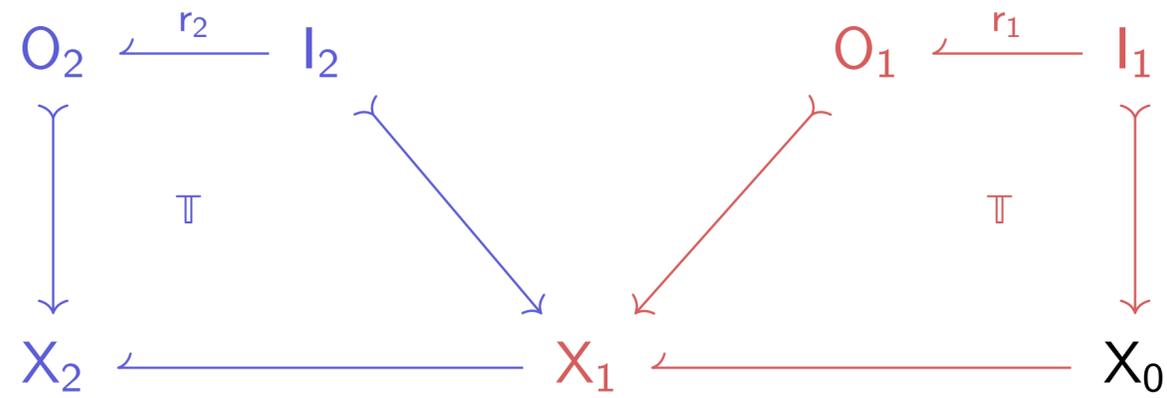
non-linear Sesqui-Pushout (SqPO)

quasi-topoi

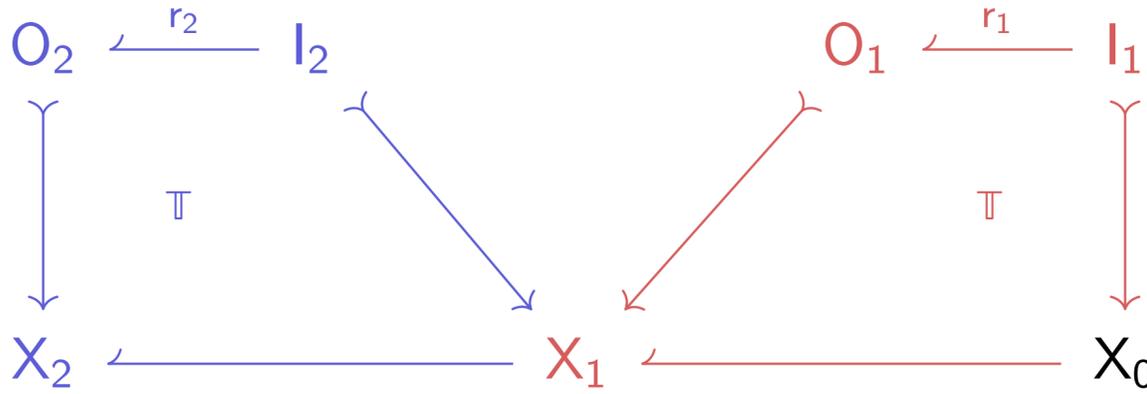
Rule algebras for categorical rewriting systems



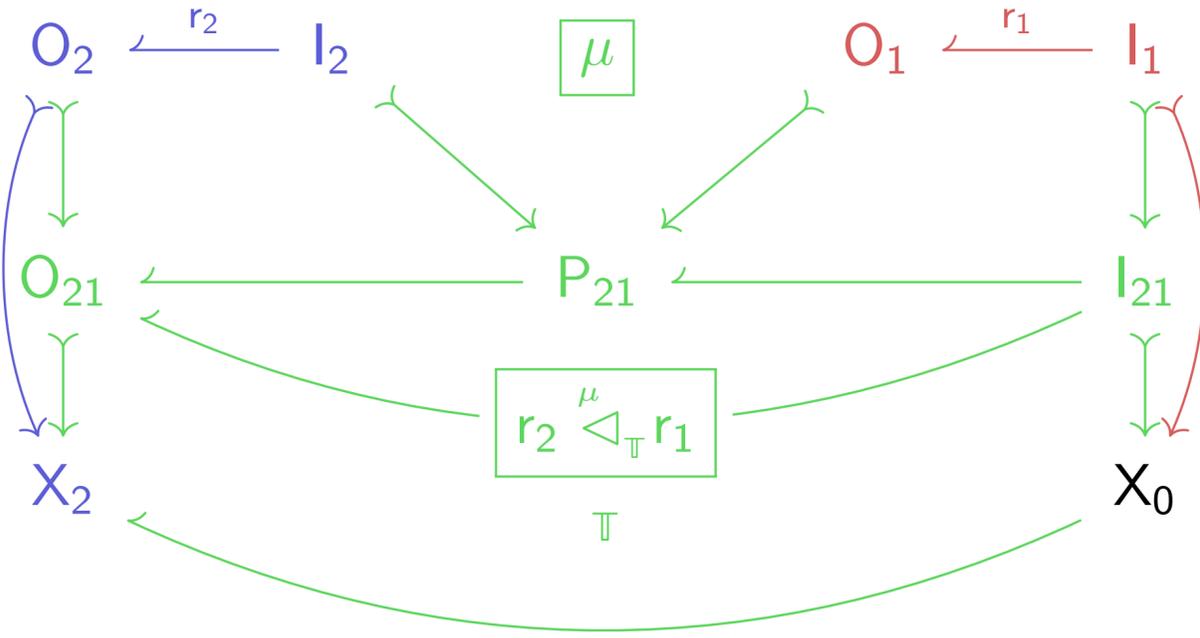
Rule algebras for categorical rewriting systems



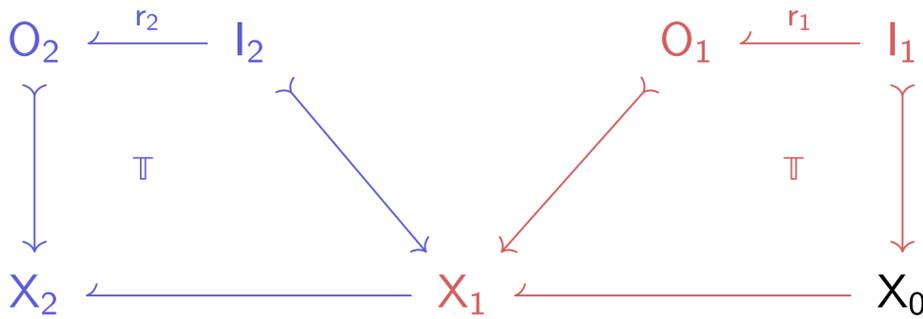
Rule algebras for categorical rewriting systems



\updownarrow 1:1



Rule algebras for categorical rewriting systems



$$\left(O \xleftarrow{r} I \right) \rightsquigarrow \delta \left(O \xleftarrow{r} I \right)$$

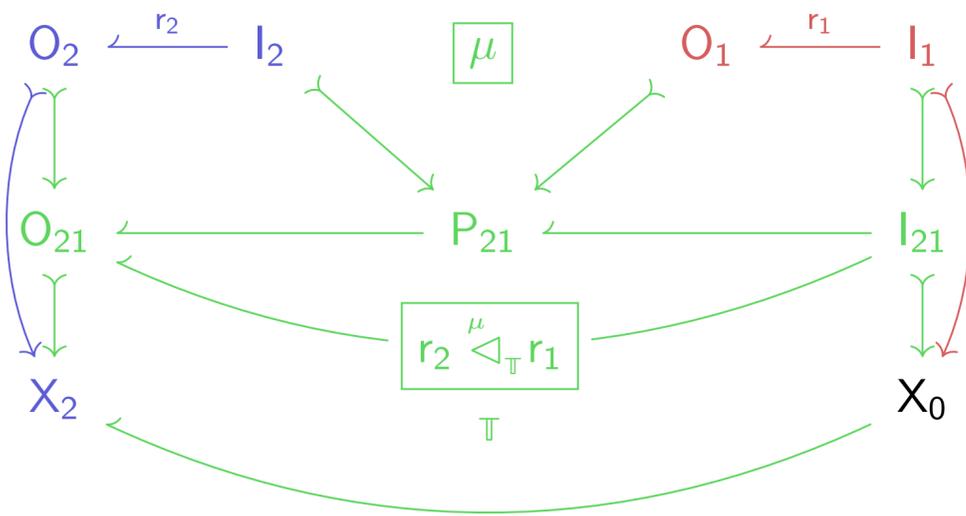
(iso-class of a) **rule** **basis vector**
of a **vector space** \mathcal{R}

\updownarrow 1:1

Definition: the **rule algebra product** $*_{\mathcal{R}} : \mathcal{R} \times \mathcal{R} \rightarrow \mathcal{R}$ is defined via

$$\delta(r_2) *_{\mathcal{R}} \delta(r_1) := \sum_{\mu} \delta \left(r_2 \overset{\mu}{\triangleleft}_{\mathbb{T}} r_1 \right)$$

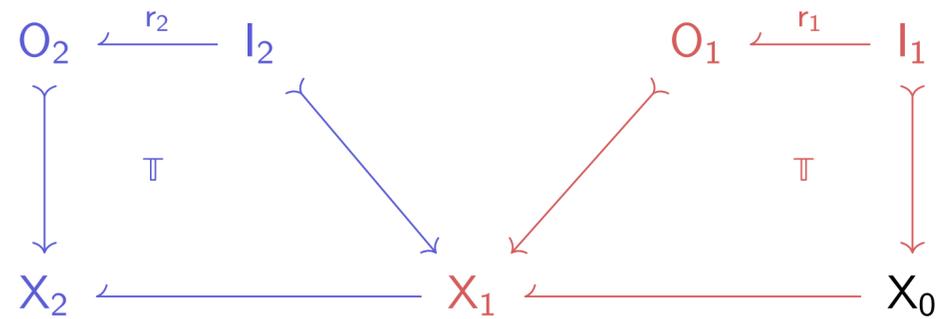
“sum over ways to compose the rules”



Theorem LiCS 2016, CSL 2018, GCM 2019, LMCS 2020, ICGT 2020

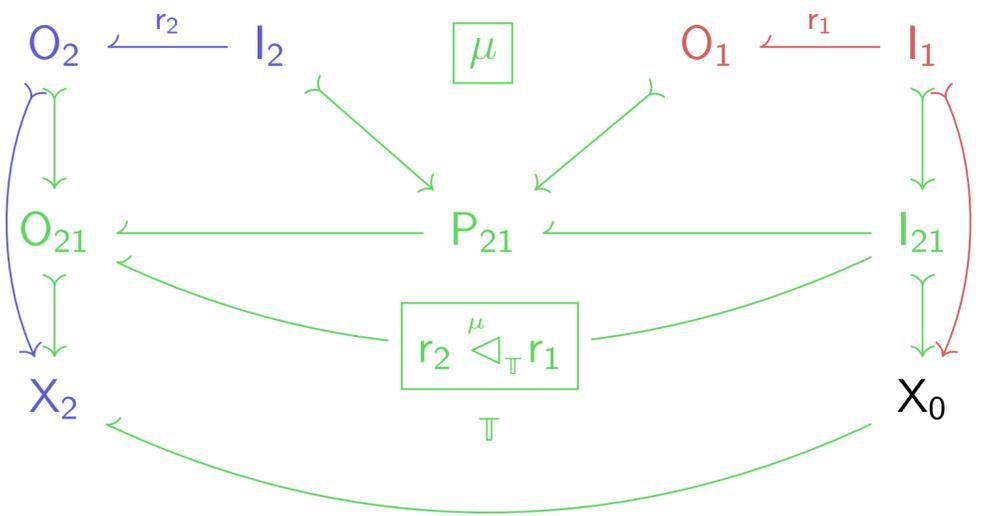
The **rule algebra** $(\mathcal{R}, *_{\mathcal{R}})$ is an **associative unital algebra**,
with **unit element** $\delta(\emptyset \leftarrow \emptyset)$.

Rule algebras for categorical rewriting systems



$$\delta(r_2) *_{\mathcal{R}} \delta(r_1) := \sum_{\mu} \delta \left(r_2 \overset{\mu}{\triangleleft}_{\mathbb{T}} r_1 \right)$$

↕ 1:1



$$\rho_{\mathbb{T}} : \mathcal{R} \rightarrow \text{End}(\hat{\mathbf{C}})$$

$$\rho_{\mathbb{T}}(\delta(r)) |X\rangle := \sum_m |r_m(X)\rangle$$

Theorem

LICS 2016, CSL 2018, GCM 2019, LMCS 2020, ICGT 2020

$\rho_{\mathbb{T}} : \mathcal{R} \rightarrow \text{End}(\hat{\mathbf{C}})$ is a **representation** of the **rule algebra** $(\mathcal{R}, *_{\mathcal{R}})$, i.e.

$$\rho_{\mathbb{T}}(\delta(r_2)) \rho_{\mathbb{T}}(\delta(r_1)) |X\rangle = \rho_{\mathbb{T}}(\delta(r_2) *_{\mathcal{R}} \delta(r_1)) |X\rangle$$

On Stochastic Rewriting and Combinatorics via Rule-Algebraic Methods*

Nicolas Behr

Université de Paris, CNRS, IRIF

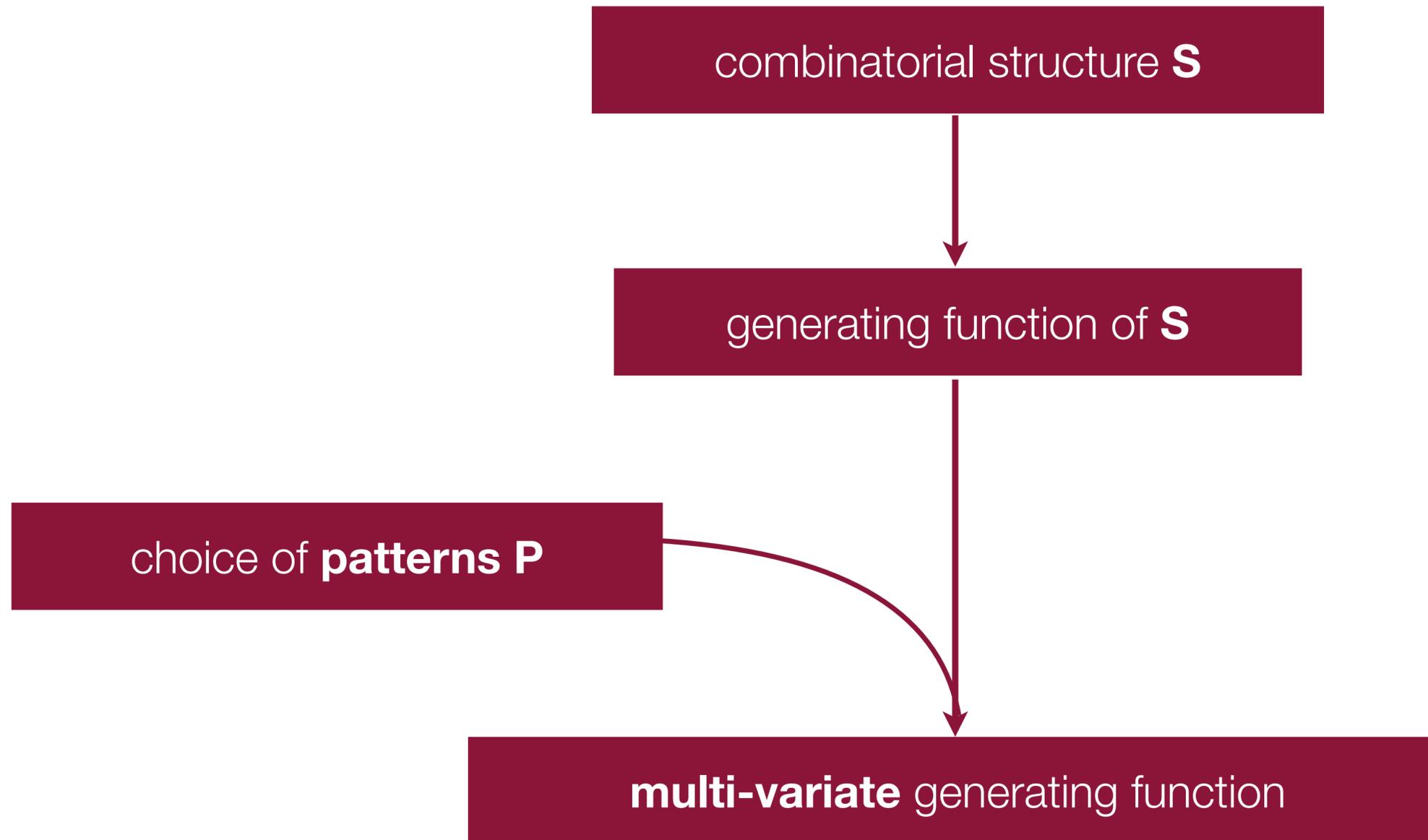
F-75006, Paris, France

nicolas.behr@irif.fr

Building upon the rule-algebraic stochastic mechanics framework, we present new results on the relationship of stochastic rewriting systems described in terms of continuous-time Markov chains, their embedded discrete-time Markov chains and certain types of generating function expressions in combinatorics. We introduce a number of generating function techniques that permit a novel form of static analysis for rewriting systems based upon marginalizing distributions over the states of the rewriting systems via pattern-counting observables.

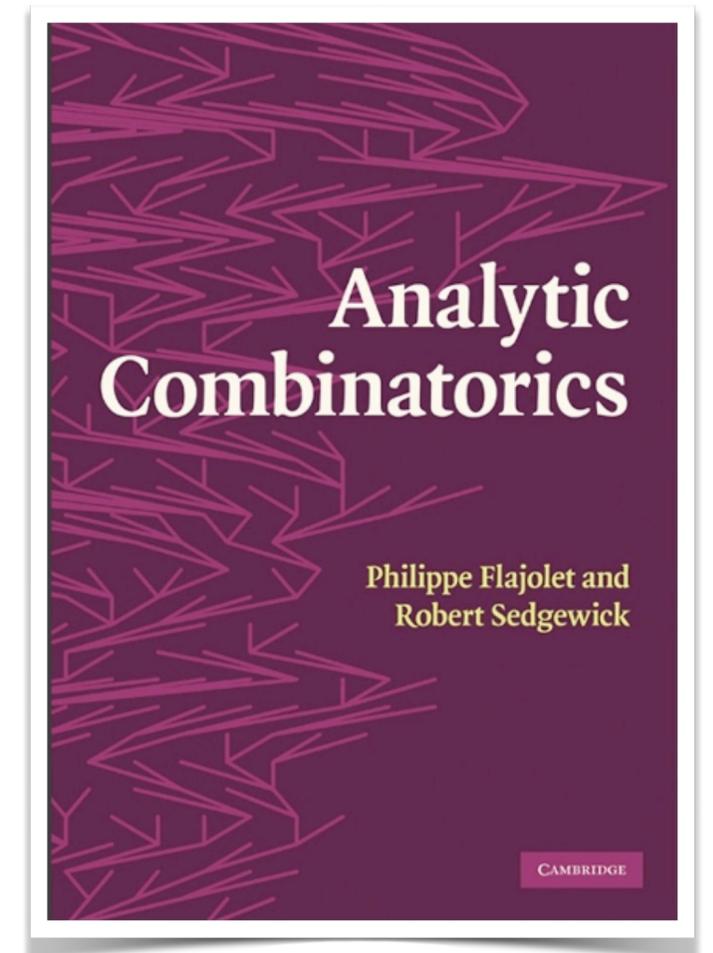
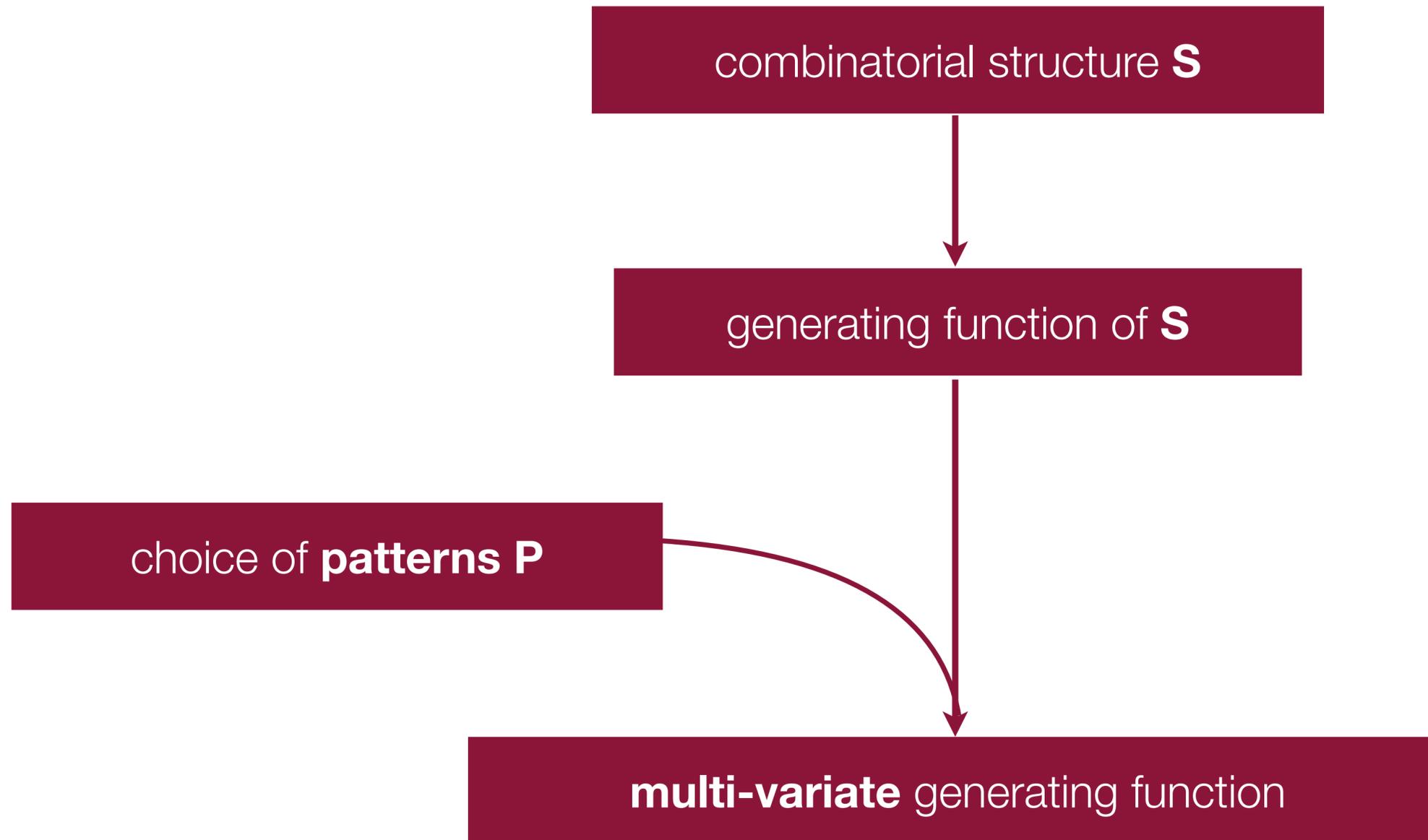
Motivation

The enumerative combinatorics "workflow" (à la Flajolet):



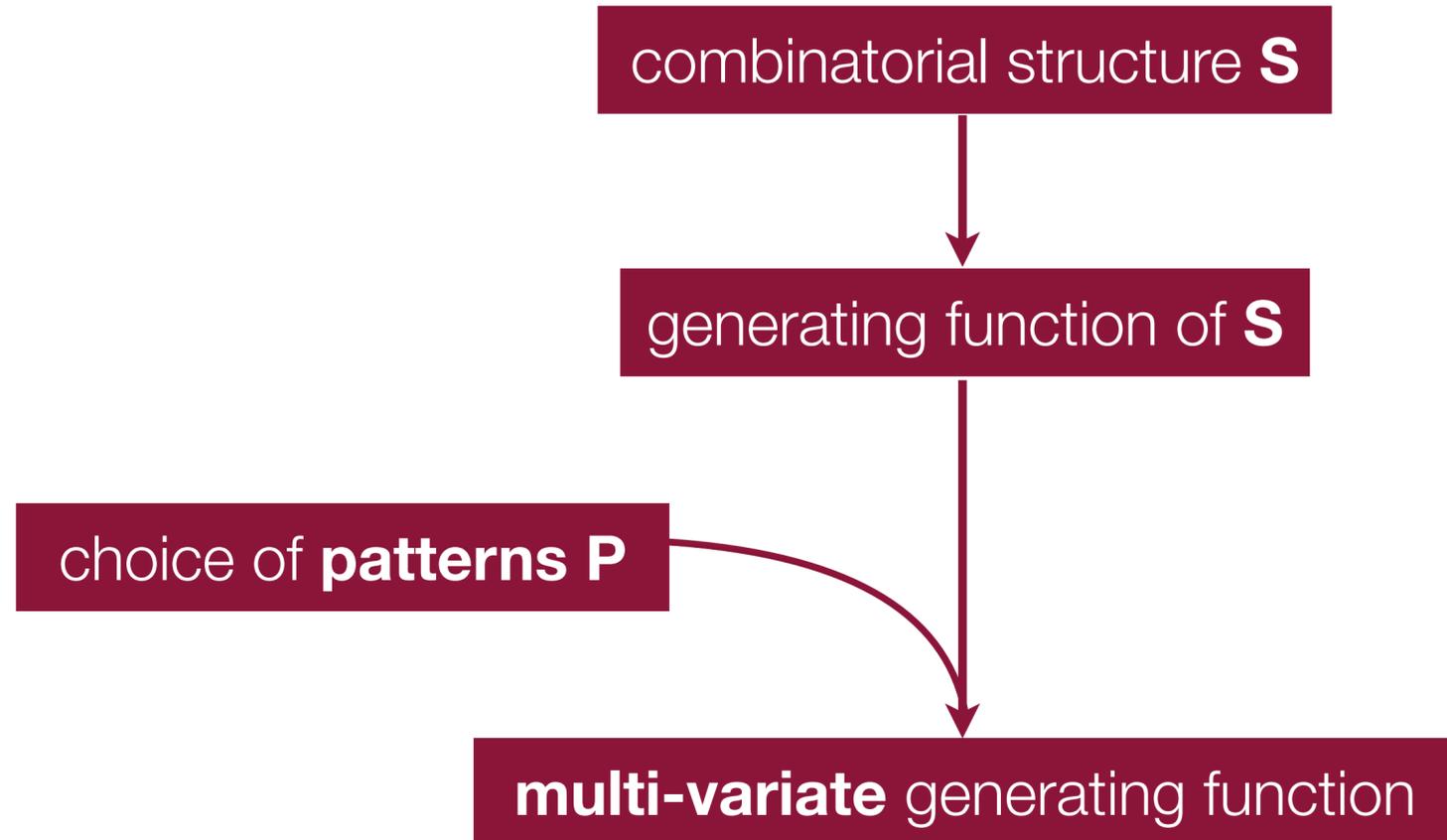
Motivation

The enumerative combinatorics "workflow" (à la Flajolet):

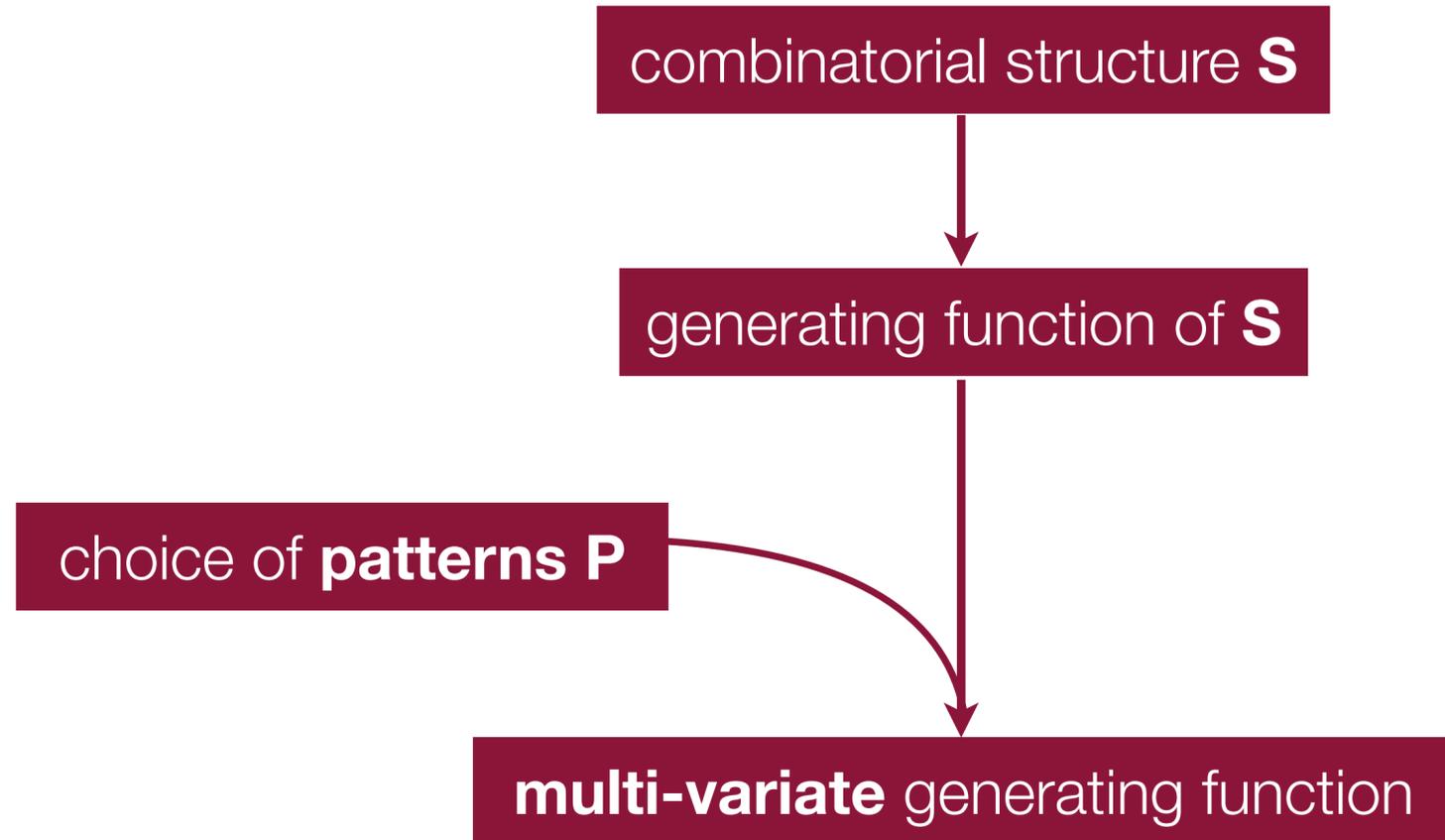


Motivation

Example: planar rooted binary trees (PRBTs)



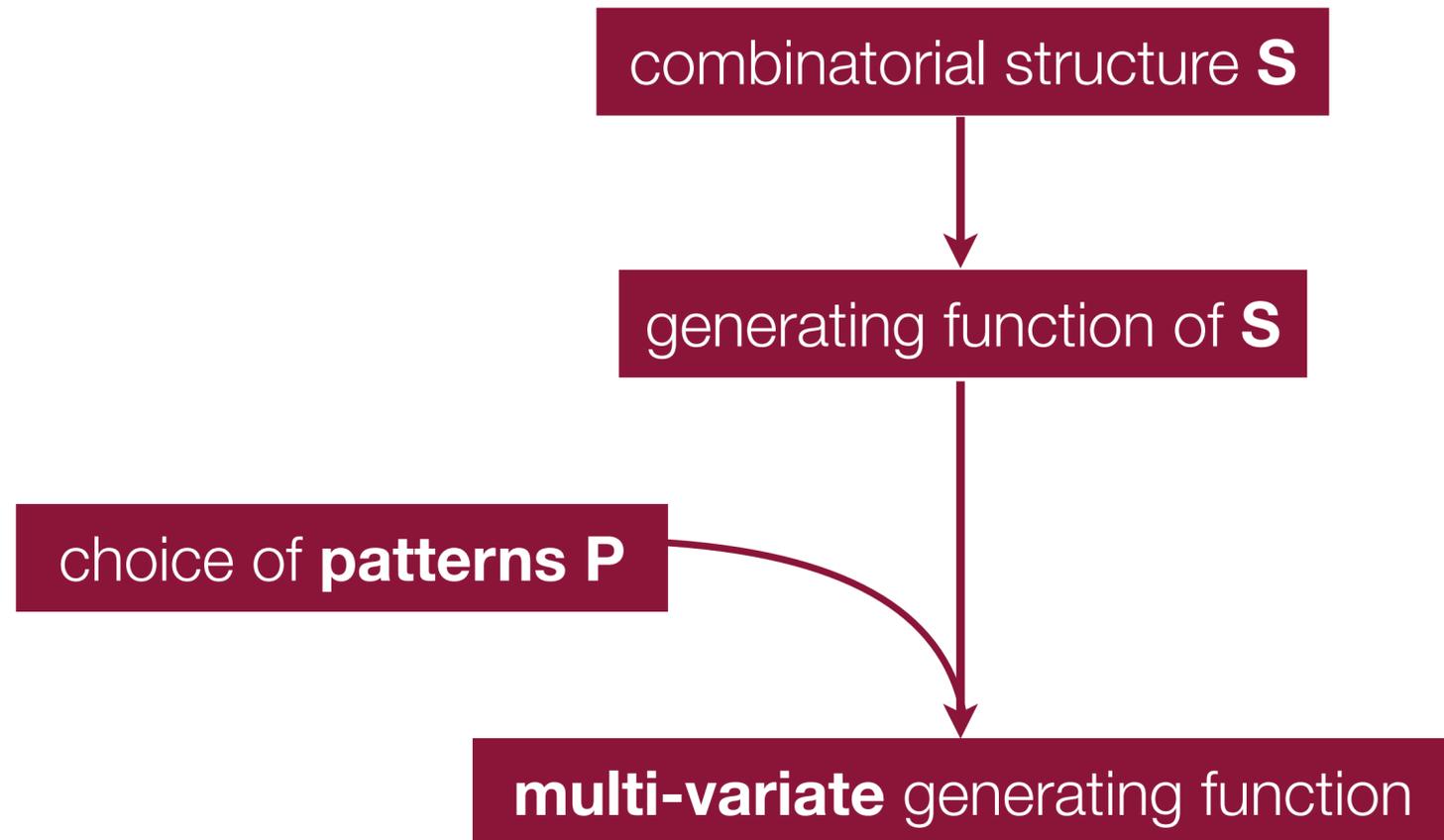
Motivation



Example: planar rooted binary trees (PRBTs)

$$\mathcal{T}_0 := \left\{ \begin{array}{c} \bullet \\ | \\ \bullet \end{array} \right\}, \quad \mathcal{T}_1 := \left\{ \begin{array}{c} \bullet & \bullet \\ / & \backslash \\ \bullet & \bullet \\ | \\ \bullet \end{array} \right\}, \quad \mathcal{T}_2 := \left\{ \begin{array}{c} \bullet & \bullet \\ / & \backslash \\ \bullet & \bullet \\ / & \backslash \\ \bullet & \bullet \\ | \\ \bullet \end{array} \right\}, \quad \dots$$

Motivation

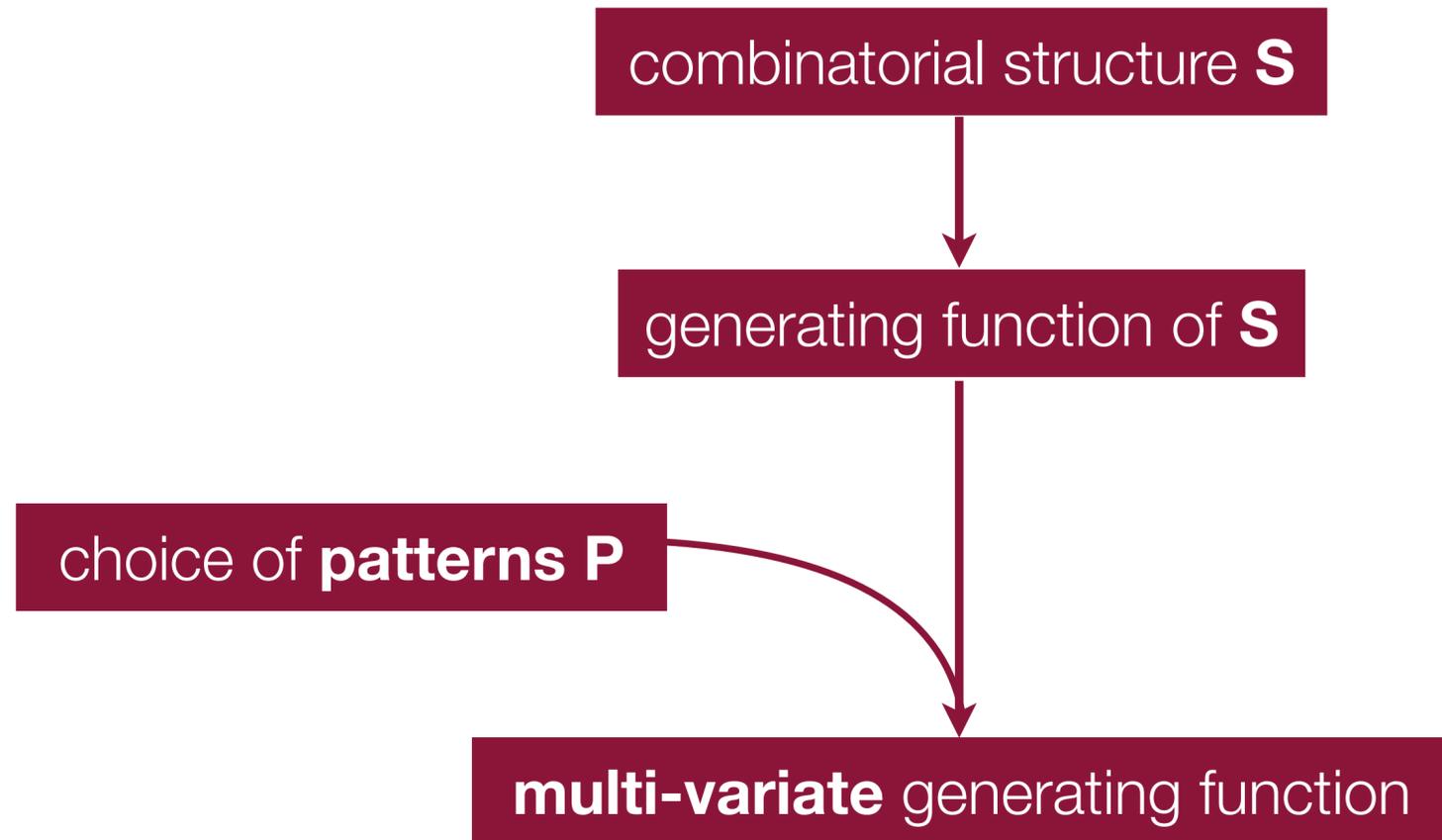


Example: planar rooted binary trees (PRBTs)

$$\mathcal{T}_0 := \left\{ \begin{array}{c} \bullet \\ | \\ \bullet \end{array} \right\}, \quad \mathcal{T}_1 := \left\{ \begin{array}{c} \bullet & \bullet \\ / & \backslash \\ \bullet & \bullet \\ | \\ \bullet \end{array} \right\}, \quad \mathcal{T}_2 := \left\{ \begin{array}{c} \bullet & \bullet \\ / & \backslash \\ \bullet & \bullet \\ / & \backslash \\ \bullet & \bullet \\ | \\ \bullet \end{array} \right\}, \dots$$

$$\mathcal{G}(\lambda) := \sum_{n \geq 0} \frac{\lambda^n}{n!} (\# \text{ of structures of size } n)$$

Motivation



Example: planar rooted binary trees (PRBTs)

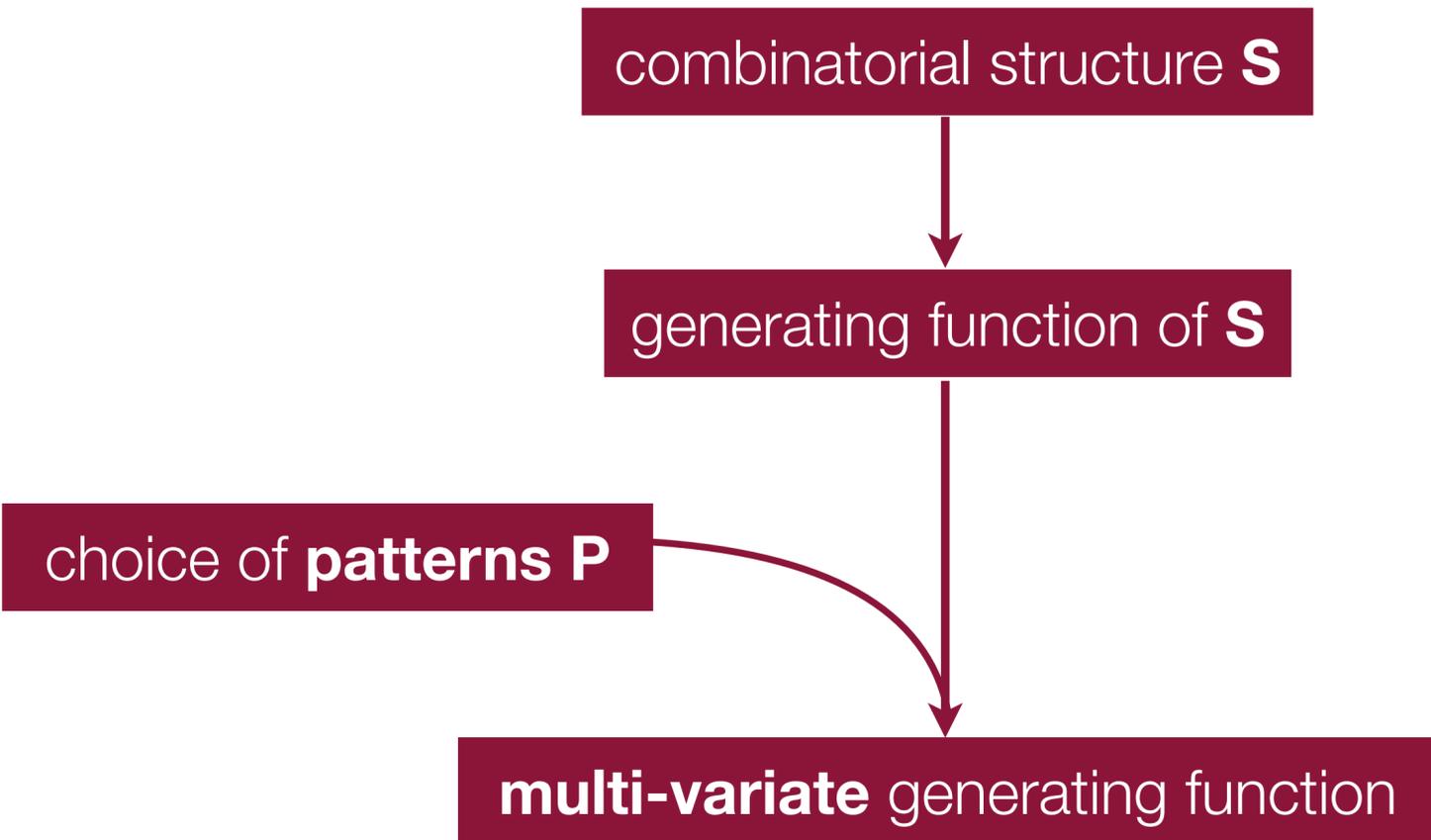
$$\mathcal{T}_0 := \left\{ \begin{array}{c} \bullet \\ | \\ \bullet \end{array} \right\}, \quad \mathcal{T}_1 := \left\{ \begin{array}{c} \bullet & \bullet \\ / & \backslash \\ \bullet & \bullet \\ | \\ \bullet \end{array} \right\}, \quad \mathcal{T}_2 := \left\{ \begin{array}{c} \bullet & \bullet \\ / & \backslash \\ \bullet & \bullet \\ / & \backslash \\ \bullet & \bullet \\ | \\ \bullet \end{array} \right\}, \dots$$

$$\mathcal{G}(\lambda) := \sum_{n \geq 0} \frac{\lambda^n}{n!} (\# \text{ of structures of size } n)$$

Choose some **patterns**:

$$P_1 := \begin{array}{c} | \\ * \end{array}, \quad P_2 := \begin{array}{c} \diagup \quad \diagdown \\ \quad \quad | \\ \quad \quad * \end{array}, \quad P_3 := \begin{array}{c} \diagup \quad \diagdown \\ / \quad \backslash \\ \quad \quad | \\ \quad \quad * \end{array}, \quad P_4 := \begin{array}{c} \diagup \quad \diagdown \\ / \quad \backslash \\ / \quad \backslash \\ \quad \quad | \\ \quad \quad * \end{array}$$

Motivation



Example: planar rooted binary trees (PRBTs)

$$\mathcal{T}_0 := \left\{ \begin{array}{c} \bullet \\ | \\ \bullet \end{array} \right\}, \quad \mathcal{T}_1 := \left\{ \begin{array}{c} \bullet & \bullet \\ / & \backslash \\ \bullet & \bullet \\ | \\ \bullet \end{array} \right\}, \quad \mathcal{T}_2 := \left\{ \begin{array}{c} \bullet & \bullet \\ / & \backslash \\ \bullet & \bullet \\ / & \backslash \\ \bullet & \bullet \\ | \\ \bullet \end{array} \right\}, \dots$$

$$\mathcal{G}(\lambda) := \sum_{n \geq 0} \frac{\lambda^n}{n!} \left(\# \text{ of structures of size } n \right)$$

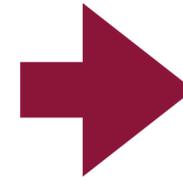
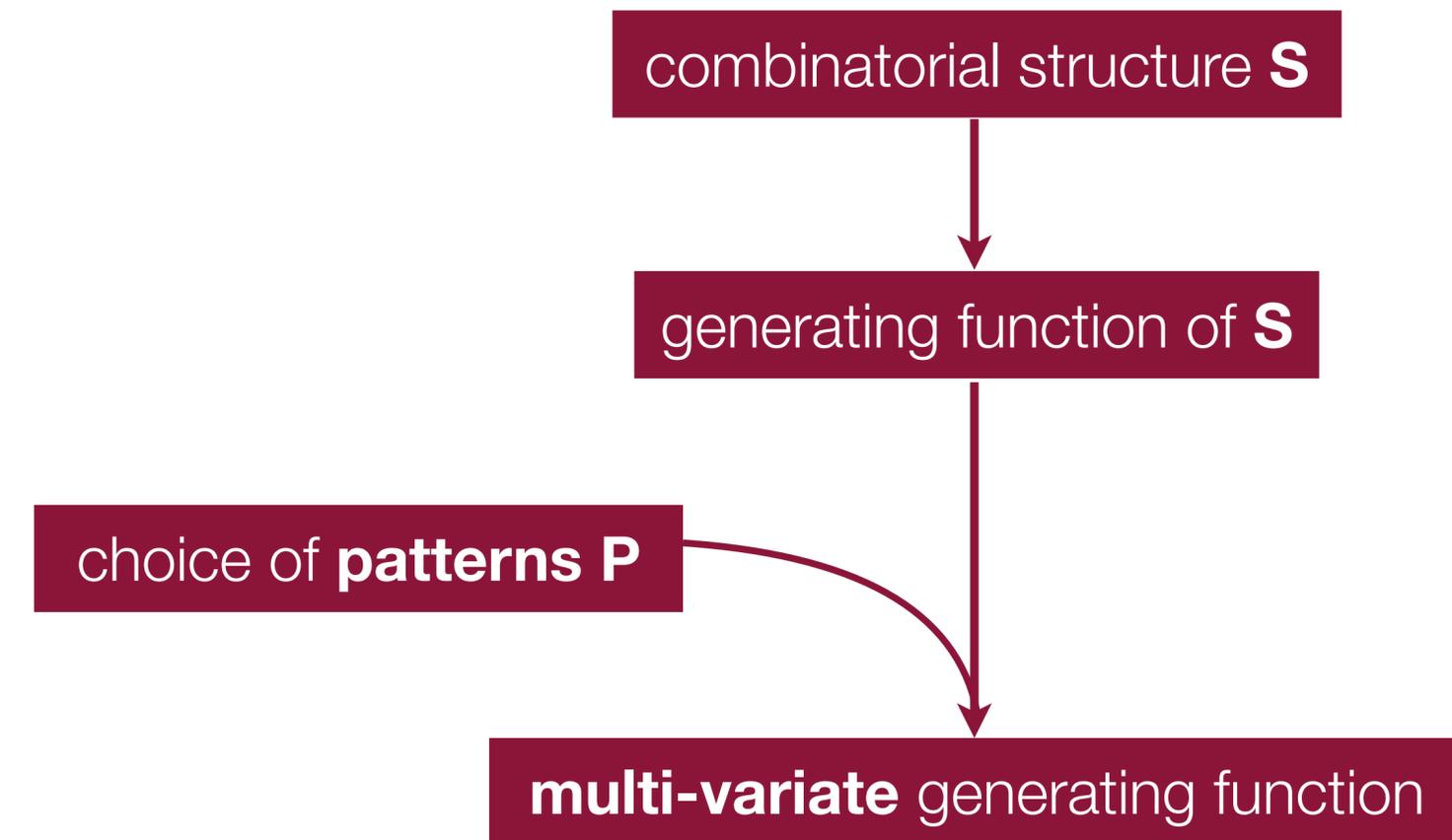
Choose some **patterns**:

$$P_1 := \begin{array}{c} | \\ * \end{array}, \quad P_2 := \begin{array}{c} \diagup \quad \diagdown \\ \quad \quad | \\ \quad \quad * \end{array}, \quad P_3 := \begin{array}{c} \diagup \quad \diagdown \\ \diagup \quad \diagdown \\ \quad \quad | \\ \quad \quad * \end{array}, \quad P_4 := \begin{array}{c} \diagup \quad \diagdown \\ \diagup \quad \diagdown \\ \diagup \quad \diagdown \\ \quad \quad | \\ \quad \quad * \end{array}$$

$$\mathcal{G}(\lambda; \omega_1, \dots, \omega_k) := \sum_{n \geq 0} \frac{\lambda^n}{n!} \sum_{p_1, \dots, p_k \geq 0} \frac{\omega_1^{p_1} \cdots \omega_k^{p_k}}{p_1! \cdots p_k!} \left(\# \text{ of structures of size } n \text{ and with } p_i \text{ occurrences of pattern } P_i \text{ (for } 1 \leq i \leq k) \right)$$

This talk

An **alternative approach** to enumerative combinatorics based upon **rewriting theory**:

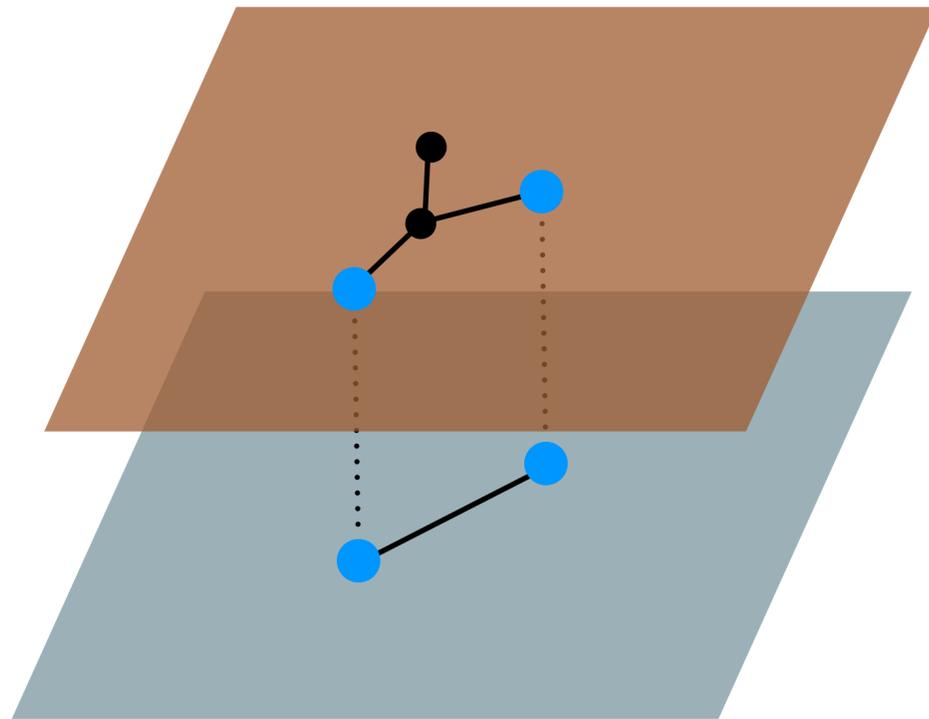


- **generate** structure **S** via applying **rewriting rules** to some **initial configuration** “in all possible ways”
- **count patterns** via applying special types of rewriting rules
- formulate **generating functions** via **linear operators** associated to rewriting rules

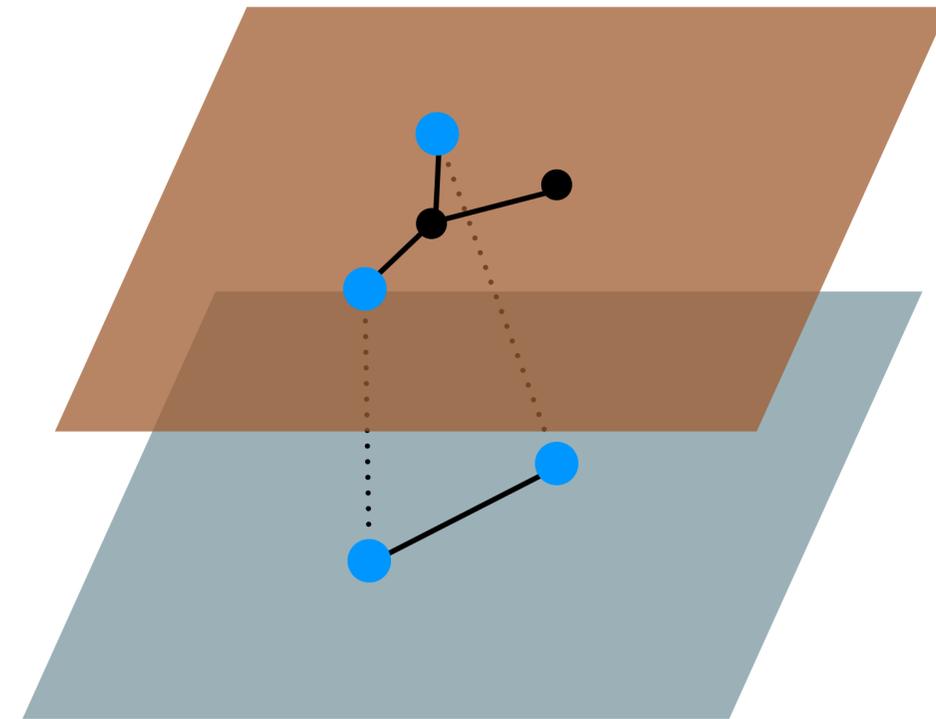
Key tool: the **rule-algebra** formalism!

Example: generating **planar rooted binary trees (PRBTs)** uniformly

The Rémy uniform generator (heuristics)



or



Example: generating **planar rooted binary trees (PRBTs)** uniformly

⇒ **combinatorics of partial observations**: rather than trying to reason about the full structure of the combinatorial species, we instead **pick** a (finite) **set of patterns** P_1, \dots, P_k and try to reason about their combinatorics within the species via **EGFs**

$$\mathcal{G}(\lambda; \omega_1, \dots, \omega_k) := \sum_{n \geq 0} \frac{\lambda^n}{n!} \sum_{p_1, \dots, p_k \geq 0} \frac{\omega_1^{p_1} \cdots \omega_k^{p_k}}{p_1! \cdots p_k!} \left(\begin{array}{l} \# \text{ of structures of size } n \\ \text{and with } p_i \text{ occurrences of pattern } P_i \text{ (for } 1 \leq i \leq k) \end{array} \right)$$

Example: generating **planar rooted binary trees (PRBTs)** uniformly

⇒ **combinatorics of partial observations**: rather than trying to reason about the full structure of the combinatorial species, we instead **pick** a (finite) **set of patterns** P_1, \dots, P_k and try to reason about their combinatorics within the species via **EGFs**

$$\mathcal{G}(\lambda; \omega_1, \dots, \omega_k) := \sum_{n \geq 0} \frac{\lambda^n}{n!} \sum_{p_1, \dots, p_k \geq 0} \frac{\omega_1^{p_1} \cdots \omega_k^{p_k}}{p_1! \cdots p_k!} \left(\begin{array}{l} \# \text{ of structures of size } n \\ \text{and with } p_i \text{ occurrences of pattern } P_i \text{ (for } 1 \leq i \leq k) \end{array} \right)$$

Insight from **stochastic mechanics**: introduce so-called **observables** \hat{O}_P

$$\hat{O}_P |t\rangle := (\#_P(t)) \cdot |t\rangle \quad P \text{ — a PBRT pattern}$$

of occurrences
of P in the PBRT t

A rule-algebraic **generating-functionology**

Definition Let $\langle |$ be defined via $\langle | t \rangle := 1_{\mathbb{R}}$ for arbitrary PRBT iso-class t . (*Note:* this permits to implement the operation of **summation over coefficients**)

A rule-algebraic **generating-functionology**

Definition Let $\langle |$ be defined via $\langle | t \rangle := 1_{\mathbb{R}}$ for arbitrary PRBT iso-class t . (Note: this permits to implement the operation of **summation over coefficients**)

Definition

(12)

A rule-algebraic **generating-functionology**

Definition Let $\langle |$ be defined via $\langle | t \rangle := 1_{\mathbb{R}}$ for arbitrary PRBT iso-class t . (Note: this permits to implement the operation of **summation over coefficients**)

Definition

let $|X_0\rangle \in \hat{\mathbf{C}}$ denote the **initial state**

$$|X_0\rangle \tag{12}$$

A rule-algebraic **generating-functionology**

Definition Let $\langle |$ be defined via $\langle | t \rangle := \mathbf{1}_{\mathbb{R}}$ for arbitrary PRBT iso-class t . (Note: this permits to implement the operation of **summation over coefficients**)

Definition Let \hat{G} be a linear operator (the **generator**),
let $|X_0\rangle \in \hat{\mathbf{C}}$ denote the **initial state**

$$e^{\lambda \hat{G}} |X_0\rangle \tag{12}$$

A rule-algebraic **generating-functionology**

Definition Let $\langle |$ be defined via $\langle | t \rangle := 1_{\mathbb{R}}$ for arbitrary PRBT iso-class t . (Note: this permits to implement the operation of **summation over coefficients**)

Definition Let \hat{G} be a linear operator (the **generator**), let $\hat{O}_1, \dots, \hat{O}_m$ be a choice of (finitely many) **pattern observables**, and let $|X_0\rangle \in \hat{\mathbf{C}}$ denote the **initial state**

$$e^{\omega \cdot \hat{O}} e^{\lambda \hat{G}} |X_0\rangle \tag{12}$$

A rule-algebraic **generating-functionology**

Definition Let $\langle |$ be defined via $\langle | t \rangle := 1_{\mathbb{R}}$ for arbitrary PRBT iso-class t . (Note: this permits to implement the operation of **summation over coefficients**)

Definition Let \hat{G} be a linear operator (the **generator**), let $\hat{O}_1, \dots, \hat{O}_m$ be a choice of (finitely many) **pattern observables**, and let $|X_0\rangle \in \hat{\mathbf{C}}$ denote the **initial state**. Then the **exponential moment-generating function (EMGF)** $\mathcal{G}(\lambda; \underline{\omega})$ is defined as

$$\mathcal{G}(\lambda; \underline{\omega}) := \langle | e^{\underline{\omega} \cdot \hat{O}} e^{\lambda \hat{G}} | X_0 \rangle \quad (12)$$

Here, we employed the shorthand notation $\underline{\omega} \cdot \hat{O} := \sum_{j=1}^m \omega_j \hat{O}_j$, and λ as well as $\omega_1, \dots, \omega_m$ are **formal variables**.

Combinatorial evolution equations

The **formal EMGF evolution equation** for $\mathcal{G}(\lambda; \underline{\omega})$ reads as follows:

$$\frac{\partial}{\partial \lambda} \mathcal{G}(\lambda; \underline{\omega}) = \langle | \left(e^{ad_{\underline{\omega} \cdot \hat{O}} \hat{G}} \right) e^{\underline{\omega} \cdot \hat{O}} e^{\lambda \hat{G}} | X_0 \rangle \quad (ad_A(B) := AB - BA) \quad (15)$$

Combinatorial evolution equations

The **formal EMGF evolution equation** for $\mathcal{G}(\lambda; \underline{\omega})$ reads as follows:

$$\frac{\partial}{\partial \lambda} \mathcal{G}(\lambda; \underline{\omega}) = \langle | \left(e^{ad_{\underline{\omega} \cdot \hat{O}} \hat{G}} \right) e^{\underline{\omega} \cdot \hat{O}} e^{\lambda \hat{G}} | X_0 \rangle \quad (ad_A(B) := AB - BA) \quad (15)$$

Applying the version of the **jump-closure theorem** appropriate for the chosen rewriting semantics (DPO or SqPO), the above formal evolution equation may be converted into a proper **evolution equation on formal power series** if the following **polynomial jump-closure** holds:

$$(PJC') \quad \forall q \in \mathbb{Z}_{\geq 0} : \exists \underline{N}(n) \in \mathbb{Z}_{\geq 0}^m, \gamma_q(\underline{\omega}, \underline{k}) \in \mathbb{R} : \langle | ad_{\underline{\omega} \cdot \hat{O}}^{\circ q}(\hat{G}) = \sum_{\underline{k}=0}^{\underline{N}(q)} \gamma_{\underline{k}}(\underline{\omega}, \underline{k}) \langle | \hat{O}^{\underline{k}} \quad (16)$$

Combinatorial evolution equations

The **formal EMGF evolution equation** for $\mathcal{G}(\lambda; \underline{\omega})$ reads as follows:

$$\frac{\partial}{\partial \lambda} \mathcal{G}(\lambda; \underline{\omega}) = \langle | \left(e^{ad_{\underline{\omega} \cdot \hat{O}} \hat{G}} \right) e^{\underline{\omega} \cdot \hat{O}} e^{\lambda \hat{G}} | X_0 \rangle \quad (ad_A(B) := AB - BA) \quad (15)$$

Applying the version of the **jump-closure theorem** appropriate for the chosen rewriting semantics (DPO or SqPO), the above formal evolution equation may be converted into a proper **evolution equation on formal power series** if the following **polynomial jump-closure** holds:

$$(PJC') \quad \forall q \in \mathbb{Z}_{\geq 0} : \exists \underline{N}(n) \in \mathbb{Z}_{\geq 0}^m, \gamma_q(\underline{\omega}, \underline{k}) \in \mathbb{R} : \langle | ad_{\underline{\omega} \cdot \hat{O}}^{\circ q}(\hat{G}) = \sum_{\underline{k}=0}^{\underline{N}(q)} \gamma_{\underline{k}}(\underline{\omega}, \underline{k}) \langle | \hat{O}^{\underline{k}} \quad (16)$$

Combinatorial evolution equations

The **formal EMGF evolution equation** for $\mathcal{G}(\lambda; \underline{\omega})$ reads as follows:

$$\frac{\partial}{\partial \lambda} \mathcal{G}(\lambda; \underline{\omega}) = \langle | \left(e^{ad_{\underline{\omega} \cdot \hat{O}} \hat{G}} \right) e^{\underline{\omega} \cdot \hat{O}} e^{\lambda \hat{G}} | X_0 \rangle \quad (ad_A(B) := AB - BA) \quad (15)$$

Applying the version of the **jump-closure theorem** appropriate for the chosen rewriting semantics (DPO or SqPO), the above formal evolution equation may be converted into a proper **evolution equation on formal power series** if the following **polynomial jump-closure** holds:

$$(PJC') \quad \forall q \in \mathbb{Z}_{\geq 0} : \exists \underline{N}(n) \in \mathbb{Z}_{\geq 0}^m, \gamma_q(\underline{\omega}, \underline{k}) \in \mathbb{R} : \langle | ad_{\underline{\omega} \cdot \hat{O}}^{\circ q}(\hat{G}) = \sum_{k=0}^{\underline{N}(q)} \gamma_{\underline{k}}(\underline{\omega}, \underline{k}) \langle | \hat{O}^{\underline{k}} \quad (16)$$

If a given set of observables satisfies (PJC'), the **formal evolution equation** (12) for the EMGF $\mathcal{G}(\lambda; \underline{\omega})$ may be refined into

$$\frac{\partial}{\partial \lambda} \mathcal{G}(\lambda; \underline{\omega}) = \mathbb{G}(\underline{\omega}, \underline{\partial \omega}) \mathcal{G}(\lambda; \underline{\omega}), \quad \mathbb{G}(\underline{\omega}, \underline{\partial \omega}) = \left(\langle | e^{ad_{\underline{\omega} \cdot \hat{O}}(\hat{G})} \right) \Big|_{\hat{O} \mapsto \underline{\partial \omega}}. \quad (17)$$

Example: generating **planar rooted binary trees (PRBTs)** uniformly

$\hat{O}_E := \text{---}^*$ Pattern E : an edge of any type

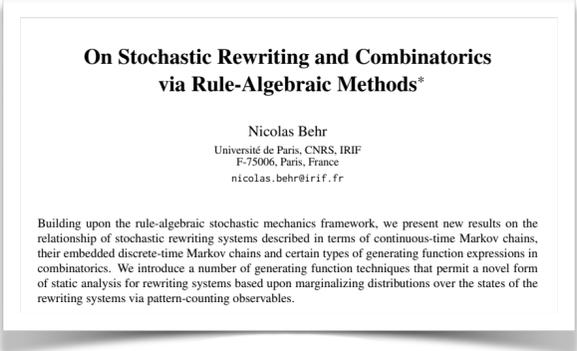
On Stochastic Rewriting and Combinatorics via Rule-Algebraic Methods*

Nicolas Behr
Université de Paris, CNRS, IRIF
F-75006, Paris, France
nicolas.behr@irif.fr

Building upon the rule-algebraic stochastic mechanics framework, we present new results on the relationship of stochastic rewriting systems described in terms of continuous-time Markov chains, their embedded discrete-time Markov chains and certain types of generating function expressions in combinatorics. We introduce a number of generating function techniques that permit a novel form of static analysis for rewriting systems based upon marginalizing distributions over the states of the rewriting systems via pattern-counting observables.

Example: generating **planar rooted binary trees (PRBTs)** uniformly

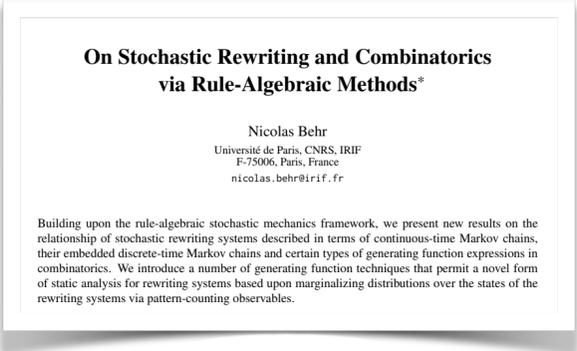
$\hat{O}_E := \text{---}^*$ Pattern E : an edge of any type



$$[\hat{O}_E, \hat{G}] = \left[\text{---} + \diagdown + \diagup, \text{---}^* + \text{---}^* \right] = \text{---}^* + \text{---}^* + \text{---}^* + \text{---}^* + \dots - \dots = 2\hat{G}$$

Example: generating **planar rooted binary trees (PRBTs)** uniformly

$\hat{O}_E := \text{---}^*$ Pattern E : an edge of any type

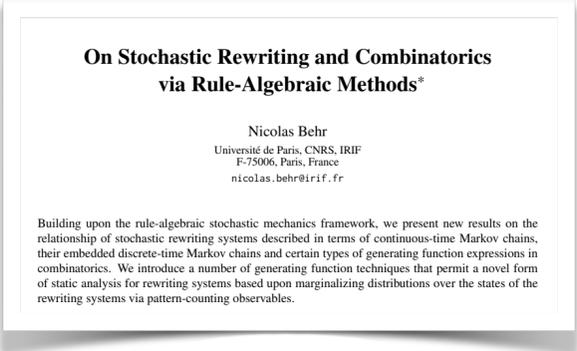


$$[\hat{O}_E, \hat{G}] = \left[\text{---} + \backslash + / , \text{---}^* + \text{---}^* \right] = \text{---}^* + \text{---}^* + \text{---}^* + \text{---}^* + \dots - \dots = 2\hat{G}$$

$$\langle | \hat{G} = 2 \langle | \hat{O}_E$$

Example: generating **planar rooted binary trees (PRBTs)** uniformly

$\hat{O}_E := \text{---}^*$ Pattern E : an edge of any type



$$[\hat{O}_E, \hat{G}] = \left[\text{---} + \diagdown + \diagup, \begin{array}{c} \text{---}^* \\ \diagdown \quad \diagup \\ \text{---}^* \end{array} + \begin{array}{c} \text{---}^* \\ \diagup \quad \diagdown \\ \text{---}^* \end{array} \right] = \begin{array}{c} \text{---}^* \\ \diagdown \quad \diagup \\ \text{---}^* \end{array} + \begin{array}{c} \text{---}^* \\ \diagup \quad \diagdown \\ \text{---}^* \end{array} + \begin{array}{c} \text{---}^* \\ \diagdown \quad \diagup \\ \text{---}^* \end{array} + \begin{array}{c} \text{---}^* \\ \diagup \quad \diagdown \\ \text{---}^* \end{array} + \dots - \dots = 2\hat{G}$$

$$\langle | \hat{G} = 2 \langle | \hat{O}_E$$

$$\begin{cases} \frac{\partial}{\partial \lambda} \mathcal{G}(\lambda; \varepsilon) = 2e^{2\varepsilon} \frac{\partial}{\partial \varepsilon} \mathcal{G}(\lambda; \varepsilon) \\ \mathcal{G}(0; \varepsilon) = \langle | e^{\varepsilon \hat{O}_E} | | \rangle = e^\varepsilon \end{cases} \Rightarrow \mathcal{G}(\lambda; \varepsilon) = \frac{1}{\sqrt{e^{-2\varepsilon} - 4\lambda}} = \sum_{n \geq 0} \frac{\lambda^n}{n!} \left(\frac{(2n)!}{n!} e^{\varepsilon(2n+1)} \right)$$

Example: generating **planar rooted binary trees (PRBTs)** uniformly

$$\hat{O}_{P1} := \begin{array}{c} \diagup \quad \diagdown \\ | \\ * \end{array} \equiv \sum_{T \in \{I, L, R\}} \begin{array}{c} \diagup \quad \diagdown \\ | \\ T \end{array},$$

$$\hat{O}_{P2} := \begin{array}{c} \diagup \quad \diagdown \\ \diagup \quad \diagdown \\ | \\ * \end{array} \equiv \sum_{T \in \{I, L, R\}} \begin{array}{c} \diagup \quad \diagdown \\ \diagup \quad \diagdown \\ | \\ T \end{array},$$

$$\hat{O}_{P3} := \begin{array}{c} \diagup \quad \diagdown \\ \diagup \quad \diagdown \\ \diagup \quad \diagdown \\ | \\ * \end{array} \equiv \sum_{T \in \{I, L, R\}} \begin{array}{c} \diagup \quad \diagdown \\ \diagup \quad \diagdown \\ \diagup \quad \diagdown \\ | \\ T \end{array}$$

Example: generating **planar rooted binary trees (PRBTs)** uniformly

$$\hat{O}_{P1} := \begin{array}{c} \diagup \\ | \\ * \\ \diagdown \end{array} \equiv \sum_{T \in \{I, L, R\}} \begin{array}{c} \diagup \\ | \\ T \\ \diagdown \end{array}, \quad \hat{O}_{P2} := \begin{array}{c} \diagup \\ \diagdown \\ | \\ * \end{array} \equiv \sum_{T \in \{I, L, R\}} \begin{array}{c} \diagup \\ \diagdown \\ | \\ T \end{array}, \quad \hat{O}_{P3} := \begin{array}{c} \diagup \\ \diagdown \\ \diagup \\ \diagdown \\ | \\ * \end{array} \equiv \sum_{T \in \{I, L, R\}} \begin{array}{c} \diagup \\ \diagdown \\ \diagup \\ \diagdown \\ | \\ T \end{array}$$

$$[\hat{O}_{P2}, \hat{G}] = \begin{array}{c} \bullet \\ \diagup \\ \diagdown \\ | \\ * \end{array} + \begin{array}{c} \bullet \\ \diagdown \\ \diagup \\ | \\ * \end{array} + \begin{array}{c} \bullet \\ \diagup \\ \diagdown \\ | \\ * \end{array} - \begin{array}{c} \bullet \\ \diagdown \\ \diagdown \\ | \\ * \end{array} - \begin{array}{c} \bullet \\ \diagup \\ \diagup \\ | \\ * \end{array}, \quad \hat{R}_{P3'} := \begin{array}{c} \bullet \\ \diagup \\ \diagdown \\ \diagup \\ \diagdown \\ | \\ * \end{array}$$

$$[\hat{O}_{P3}, \hat{G}] = \begin{array}{c} \bullet \\ \diagdown \\ \diagup \\ \diagdown \\ | \\ * \end{array} + \begin{array}{c} \bullet \\ \diagup \\ \diagdown \\ \diagdown \\ | \\ * \end{array} + \begin{array}{c} \bullet \\ \diagdown \\ \diagup \\ \diagup \\ | \\ * \end{array} + \begin{array}{c} \bullet \\ \diagup \\ \diagdown \\ \diagup \\ | \\ * \end{array} - \begin{array}{c} \bullet \\ \diagdown \\ \diagdown \\ \diagdown \\ | \\ * \end{array} - \begin{array}{c} \bullet \\ \diagup \\ \diagdown \\ \diagdown \\ | \\ * \end{array} - \begin{array}{c} \bullet \\ \diagup \\ \diagdown \\ \diagup \\ | \\ * \end{array} - \hat{R}_{P3'}$$

$$[\hat{O}_{P2}, [\hat{O}_{P2}, \hat{G}]] = [\hat{O}_{P2}, \hat{G}], \quad [\hat{O}_{P2}, [\hat{O}_{P3}, \hat{G}]] = [\hat{O}_{P3}, \hat{G}] + \hat{R}_{P3}$$

$$[\hat{O}_{P3}, [\hat{O}_{P3}, \hat{G}]] = [\hat{O}_{P3}, \hat{G}] + 2\hat{R}_{P3'}, \quad [\hat{O}_{P2}, \hat{R}_{P3'}] = 0, \quad [\hat{O}_{P3}, \hat{R}_{P3'}] = -\hat{R}_{P3'}$$

$$\langle | [\hat{O}_{P2}, \hat{G}] = \langle | (3\hat{O}_{P1} - 2\hat{O}_{P2}), \quad \langle | [\hat{O}_{P3}, \hat{G}] = \langle | (4\hat{O}_{P2} - 3\hat{O}_{P3}), \quad \langle | \hat{R}_{P3'} = \langle | \hat{O}_{P3}$$

Example: generating **planar rooted binary trees (PRBTs)** uniformly

$$\hat{O}_{P1} := \text{Y-shape with root } * \equiv \sum_{T \in \{I, L, R\}} \text{Y-shape with root } T,$$

$$\hat{O}_{P2} := \text{Y-shape with root } * \equiv \sum_{T \in \{I, L, R\}} \text{Y-shape with root } T,$$

$$\hat{O}_{P3} := \text{Y-shape with root } * \equiv \sum_{T \in \{I, L, R\}} \text{Y-shape with root } T,$$

$[\hat{O}_{P2}, \hat{G}] =$

$[\hat{O}_{P3}, \hat{G}] =$

$\hat{R}_{P3'} :=$

$[\hat{O}_{P2}, [\hat{O}_{P2}, \hat{G}]] = [\hat{O}_{P2}, \hat{G}], \quad [\hat{O}_{P2}, [\hat{O}_{P3}, \hat{G}]] = [\hat{O}_{P3}, \hat{G}] + \hat{R}_{P3}$

$[\hat{O}_{P3}, [\hat{O}_{P3}, \hat{G}]] = [\hat{O}_{P3}, \hat{G}] + 2\hat{R}_{P3'}, \quad [\hat{O}_{P2}, \hat{R}_{P3'}] = 0, \quad [\hat{O}_{P3}, \hat{R}_{P3'}] = -\hat{R}_{P3'}$

$\langle | [\hat{O}_{P2}, \hat{G}] = \langle | (3\hat{O}_{P1} - 2\hat{O}_{P2}), \quad \langle | [\hat{O}_{P3}, \hat{G}] = \langle | (4\hat{O}_{P2} - 3\hat{O}_{P3}), \quad \langle | \hat{R}_{P3'} = \langle | \hat{O}_{P3}$

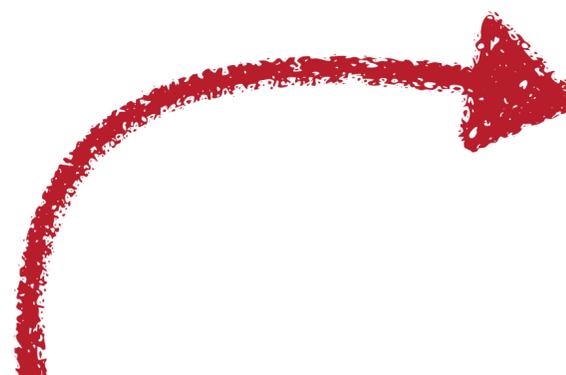


Example: generating **planar rooted binary trees (PRBTs)** uniformly

$$\hat{O}_{P1} := \begin{array}{c} \diagup \\ | \\ * \\ \diagdown \end{array} \equiv \sum_{T \in \{I, L, R\}} \begin{array}{c} \diagup \\ | \\ T \\ \diagdown \end{array},$$

$$\hat{O}_{P2} := \begin{array}{c} \diagup \\ \diagdown \\ | \\ * \end{array} \equiv \sum_{T \in \{I, L, R\}} \begin{array}{c} \diagup \\ \diagdown \\ | \\ T \end{array},$$

$$\hat{O}_{P3} := \begin{array}{c} \diagup \\ \diagdown \\ \diagup \\ \diagdown \\ | \\ * \end{array} \equiv \sum_{T \in \{I, L, R\}} \begin{array}{c} \diagup \\ \diagdown \\ \diagup \\ \diagdown \\ | \\ T \end{array}$$



$$\mathcal{G}(\lambda; \underline{\omega}) := \langle | e^{\underline{\omega} \cdot \hat{O}} e^{\lambda \hat{G}} | | \rangle, \quad \underline{\omega} \cdot \hat{O} := \varepsilon \hat{O}_E + \gamma \hat{O}_{P1} + \mu \hat{O}_{P2} + \nu \hat{O}_{P3}$$

$$\frac{\partial}{\partial \lambda} \mathcal{G}(\lambda; \underline{\omega}) = \langle | \left(e^{ad_{\underline{\omega} \cdot \hat{O}}}(\hat{G}) \right) e^{\underline{\omega} \cdot \hat{O}} e^{\lambda \hat{G}} | | \rangle \stackrel{(*)}{=} \langle | \left(e^{ad_{\nu \hat{O}_{P3}}} \left(e^{ad_{\mu \hat{O}_{P2}}} \left(e^{ad_{\varepsilon \hat{O}_E + \gamma \hat{O}_{P1}}}(\hat{G}) \right) \right) \right) e^{\underline{\omega} \cdot \hat{O}} e^{\lambda \hat{G}} | | \rangle$$

$$= e^{2\varepsilon + \gamma} \langle | \left(e^{ad_{\nu \hat{O}_{P3}}} \left(e^{ad_{\mu \hat{O}_{P2}}}(\hat{G}) \right) \right) e^{\underline{\omega} \cdot \hat{O}} e^{\lambda \hat{G}} | | \rangle$$

$$= e^{2\varepsilon + \gamma} \langle | \left(e^{ad_{\nu \hat{O}_{P3}}} \left(\hat{G} + (e^\mu - 1)[\hat{O}_{P2}, \hat{G}] \right) \right) e^{\underline{\omega} \cdot \hat{O}} e^{\lambda \hat{G}} | | \rangle$$

$$= e^{2\varepsilon + \gamma} \langle | \left(\hat{G} + (e^\mu - 1)[\hat{O}_{P2}, \hat{G}] + e^\mu (e^\nu - 1)[\hat{O}_{P3}, \hat{G}] + (e^\nu - 1)(e^\mu - e^{-\nu})\hat{R}_{P3'} \right) e^{\underline{\omega} \cdot \hat{O}} e^{\lambda \hat{G}} | | \rangle$$

$$= e^{2\varepsilon + \gamma} \langle | \left(2\hat{O}_E + 3(e^\mu - 1)\hat{O}_{P1} + (4e^{\mu+\nu} - 6e^\mu + 2)\hat{O}_{P2} + (3e^\mu + e^{-\nu} - 3e^{\mu+\nu} - 1)\hat{O}_{P3} \right) e^{\underline{\omega} \cdot \hat{O}} e^{\lambda \hat{G}} | | \rangle$$

$$= e^{2\varepsilon + \gamma} \langle | \left(2\frac{\partial}{\partial \varepsilon} + 3(e^\mu - 1)\frac{\partial}{\partial \gamma} + (4e^{\mu+\nu} - 6e^\mu + 2)\frac{\partial}{\partial \mu} + (3e^\mu + e^{-\nu} - 3e^{\mu+\nu} - 1)\frac{\partial}{\partial \nu} \right) e^{\underline{\omega} \cdot \hat{O}} e^{\lambda \hat{G}} | | \rangle$$

$$[\hat{O}_{P2}, \hat{G}] = \begin{array}{c} \diagup \\ \diagdown \\ | \\ * \end{array} + \begin{array}{c} \diagup \\ | \\ * \\ \diagdown \end{array} + \begin{array}{c} \diagup \\ | \\ * \\ \diagdown \end{array} - \begin{array}{c} \diagup \\ | \\ * \\ \diagdown \end{array} - \begin{array}{c} \diagup \\ | \\ * \\ \diagdown \end{array}$$

$$[\hat{O}_{P3}, \hat{G}] = \begin{array}{c} \diagup \\ \diagdown \\ | \\ * \end{array} + \begin{array}{c} \diagup \\ \diagdown \\ | \\ * \end{array} + \begin{array}{c} \diagup \\ \diagdown \\ | \\ * \end{array} + \begin{array}{c} \diagup \\ \diagdown \\ | \\ * \end{array} - \begin{array}{c} \diagup \\ \diagdown \\ | \\ * \end{array} - \begin{array}{c} \diagup \\ \diagdown \\ | \\ * \end{array} - \begin{array}{c} \diagup \\ \diagdown \\ | \\ * \end{array} - \hat{R}_{P3'}$$

$$\hat{R}_{P3'} := \begin{array}{c} \diagup \\ \diagdown \\ \diagup \\ \diagdown \\ | \\ * \end{array}$$

$$[\hat{O}_{P2}, [\hat{O}_{P2}, \hat{G}]] = [\hat{O}_{P2}, \hat{G}], \quad [\hat{O}_{P2}, [\hat{O}_{P3}, \hat{G}]] = [\hat{O}_{P3}, \hat{G}] + \hat{R}_{P3'}$$

$$[\hat{O}_{P3}, [\hat{O}_{P3}, \hat{G}]] = [\hat{O}_{P3}, \hat{G}] + 2\hat{R}_{P3'}, \quad [\hat{O}_{P2}, \hat{R}_{P3'}] = 0, \quad [\hat{O}_{P3}, \hat{R}_{P3'}] = -\hat{R}_{P3'}$$

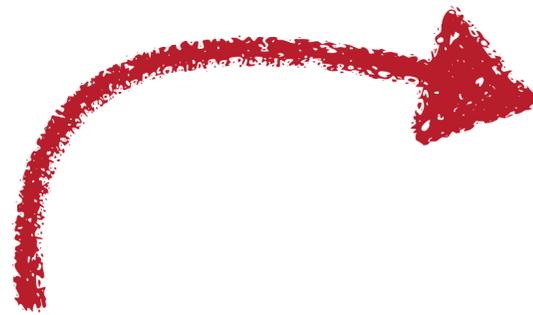
$$\langle | [\hat{O}_{P2}, \hat{G}] \rangle = \langle | (3\hat{O}_{P1} - 2\hat{O}_{P2}) \rangle, \quad \langle | [\hat{O}_{P3}, \hat{G}] \rangle = \langle | (4\hat{O}_{P2} - 3\hat{O}_{P3}) \rangle, \quad \langle | \hat{R}_{P3'} \rangle = \langle | \hat{O}_{P3} \rangle$$

Example: generating **planar rooted binary trees (PRBTs)** uniformly

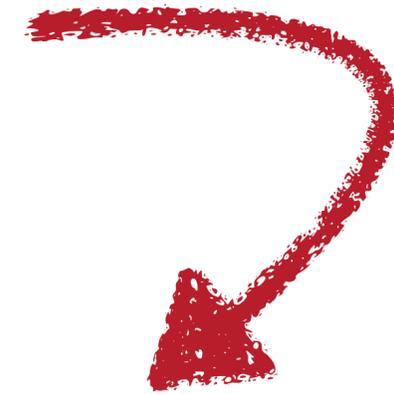
$$\hat{O}_{P1} := \begin{array}{c} \diagup \\ | \\ * \\ \diagdown \end{array} \equiv \sum_{T \in \{I, L, R\}} \begin{array}{c} \diagup \\ | \\ T \\ \diagdown \end{array},$$

$$\hat{O}_{P2} := \begin{array}{c} \diagup \\ \diagdown \\ | \\ * \end{array} \equiv \sum_{T \in \{I, L, R\}} \begin{array}{c} \diagup \\ \diagdown \\ | \\ T \end{array},$$

$$\hat{O}_{P3} := \begin{array}{c} \diagup \\ \diagdown \\ \diagup \\ | \\ * \\ \diagdown \end{array} \equiv \sum_{T \in \{I, L, R\}} \begin{array}{c} \diagup \\ \diagdown \\ \diagup \\ | \\ T \\ \diagdown \end{array}$$



$$\begin{aligned} \mathcal{G}(\lambda; \underline{\omega}) &:= \langle | e^{\underline{\omega} \cdot \hat{O}} e^{\lambda \hat{G}} | | \rangle, \quad \underline{\omega} \cdot \hat{O} := \varepsilon \hat{O}_E + \gamma \hat{O}_{P1} + \mu \hat{O}_{P2} + \nu \hat{O}_{P3} \\ \frac{\partial}{\partial \lambda} \mathcal{G}(\lambda; \underline{\omega}) &= \langle | (e^{ad_{\underline{\omega} \cdot \hat{O}}}(\hat{G})) e^{\underline{\omega} \cdot \hat{O}} e^{\lambda \hat{G}} | | \rangle \stackrel{(*)}{=} \langle | (e^{ad_{\nu \hat{O}_{P3}}} (e^{ad_{\mu \hat{O}_{P2}}} (e^{ad_{\varepsilon \hat{O}_E + \gamma \hat{O}_{P1}}}(\hat{G})))) e^{\underline{\omega} \cdot \hat{O}} e^{\lambda \hat{G}} | | \rangle \\ &= e^{2\varepsilon + \gamma} \langle | (e^{ad_{\nu \hat{O}_{P3}}} (e^{ad_{\mu \hat{O}_{P2}}}(\hat{G}))) e^{\underline{\omega} \cdot \hat{O}} e^{\lambda \hat{G}} | | \rangle \\ &= e^{2\varepsilon + \gamma} \langle | (e^{ad_{\nu \hat{O}_{P3}}}(\hat{G} + (e^\mu - 1)[\hat{O}_{P2}, \hat{G}])) e^{\underline{\omega} \cdot \hat{O}} e^{\lambda \hat{G}} | | \rangle \\ &= e^{2\varepsilon + \gamma} \langle | (\hat{G} + (e^\mu - 1)[\hat{O}_{P2}, \hat{G}] \\ &\quad + e^\mu (e^\nu - 1)[\hat{O}_{P3}, \hat{G}] + (e^\nu - 1)(e^\mu - e^{-\nu})\hat{R}_{P3'}) e^{\underline{\omega} \cdot \hat{O}} e^{\lambda \hat{G}} | | \rangle \\ &= e^{2\varepsilon + \gamma} \langle | (2\hat{O}_E + 3(e^\mu - 1)\hat{O}_{P1} + (4e^{\mu+\nu} - 6e^\mu + 2)\hat{O}_{P2} \\ &\quad + (3e^\mu + e^{-\nu} - 3e^{\mu+\nu} - 1)\hat{O}_{P3}) e^{\underline{\omega} \cdot \hat{O}} e^{\lambda \hat{G}} | | \rangle \\ &= e^{2\varepsilon + \gamma} \langle | (2\frac{\partial}{\partial \varepsilon} + 3(e^\mu - 1)\frac{\partial}{\partial \gamma} + (4e^{\mu+\nu} - 6e^\mu + 2)\frac{\partial}{\partial \mu} \\ &\quad + (3e^\mu + e^{-\nu} - 3e^{\mu+\nu} - 1)\frac{\partial}{\partial \nu}) e^{\underline{\omega} \cdot \hat{O}} e^{\lambda \hat{G}} | | \rangle \end{aligned}$$



$$\begin{aligned} [\hat{O}_{P2}, \hat{G}] &= \begin{array}{c} \diagup \\ | \\ \cdot \\ \diagdown \end{array} + \begin{array}{c} \cdot \\ \diagup \\ | \\ \cdot \\ \diagdown \end{array} + \begin{array}{c} \cdot \\ \diagup \\ \cdot \\ | \\ \cdot \\ \diagdown \end{array} - \begin{array}{c} \cdot \\ \diagup \\ \cdot \\ | \\ \cdot \\ \diagdown \end{array} - \begin{array}{c} \cdot \\ \diagup \\ \cdot \\ | \\ \cdot \\ \diagdown \end{array} \\ [\hat{O}_{P3}, \hat{G}] &= \begin{array}{c} \cdot \\ \diagup \\ \cdot \\ | \\ \cdot \\ \diagdown \end{array} + \begin{array}{c} \cdot \\ \diagup \\ \cdot \\ | \\ \cdot \\ \diagdown \end{array} + \begin{array}{c} \cdot \\ \diagup \\ \cdot \\ | \\ \cdot \\ \diagdown \end{array} + \begin{array}{c} \cdot \\ \diagup \\ \cdot \\ | \\ \cdot \\ \diagdown \end{array} - \begin{array}{c} \cdot \\ \diagup \\ \cdot \\ | \\ \cdot \\ \diagdown \end{array} - \begin{array}{c} \cdot \\ \diagup \\ \cdot \\ | \\ \cdot \\ \diagdown \end{array} - \begin{array}{c} \cdot \\ \diagup \\ \cdot \\ | \\ \cdot \\ \diagdown \end{array} - \hat{R}_{P3'} \\ \hat{R}_{P3'} &:= \begin{array}{c} \cdot \\ \diagup \\ \cdot \\ | \\ \cdot \\ \diagdown \end{array} \end{aligned}$$

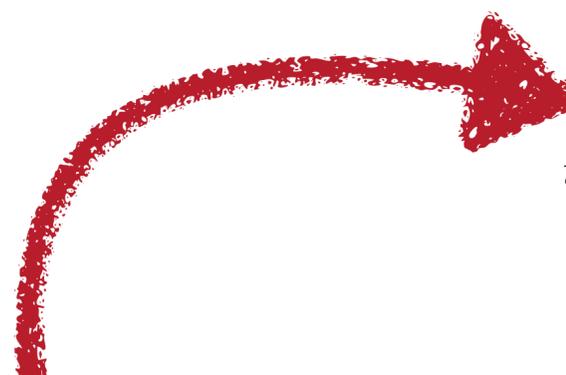
$$\begin{aligned} [\hat{O}_{P2}, [\hat{O}_{P2}, \hat{G}]] &= [\hat{O}_{P2}, \hat{G}], \quad [\hat{O}_{P2}, [\hat{O}_{P3}, \hat{G}]] = [\hat{O}_{P3}, \hat{G}] + \hat{R}_{P3} \\ [\hat{O}_{P3}, [\hat{O}_{P3}, \hat{G}]] &= [\hat{O}_{P3}, \hat{G}] + 2\hat{R}_{P3'}, \quad [\hat{O}_{P2}, \hat{R}_{P3'}] = 0, \quad [\hat{O}_{P3}, \hat{R}_{P3'}] = -\hat{R}_{P3'} \\ \langle | [\hat{O}_{P2}, \hat{G}] | | \rangle &= \langle | (3\hat{O}_{P1} - 2\hat{O}_{P2}) | | \rangle, \quad \langle | [\hat{O}_{P3}, \hat{G}] | | \rangle = \langle | (4\hat{O}_{P2} - 3\hat{O}_{P3}) | | \rangle, \quad \langle | \hat{R}_{P3'} | | \rangle = \langle | \hat{O}_{P3} | | \rangle \end{aligned}$$

Granted that the derivation of the evolution equation for $\mathcal{G}(\lambda; \underline{\omega})$ is somewhat involved, one may extract from it a very interesting insight via a transformation of variables $\omega_i \rightarrow \ln x_i$ (which entails that $\frac{\partial}{\partial \omega_i} \rightarrow x_i \frac{\partial}{\partial x_i}$), and collecting coefficients for the operators $\hat{n}_i := x_i \frac{\partial}{\partial x_i}$:

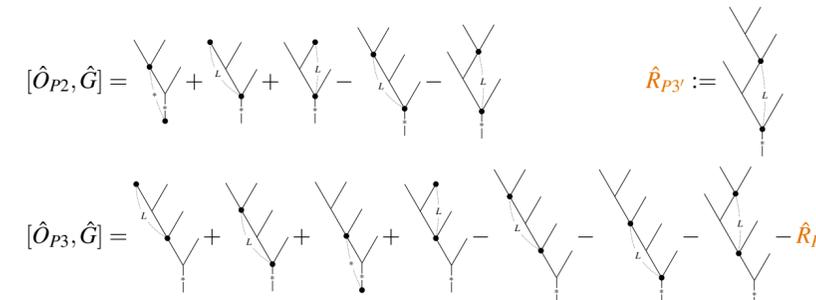
$$\begin{aligned} \frac{\partial}{\partial \lambda} \mathcal{G}(\lambda; \underline{\ln x}) &= \hat{D} \mathcal{G}(\lambda; \underline{\ln x}) \\ \hat{D} &= x_\varepsilon^2 x_\nu (2\hat{n}_\varepsilon - 3\hat{n}_\gamma + 2\hat{n}_\mu - \hat{n}_\nu) + x_\varepsilon^2 x_\nu x_\mu (3\hat{n}_\gamma - 6\hat{n}_\mu + 3\hat{n}_\nu) + x_\varepsilon^2 x_\nu x_\mu^2 (4\hat{n}_\mu - 3\hat{n}_\nu) + x_\varepsilon^2 \hat{n}_\nu \end{aligned} \tag{58}$$

Example: generating **planar rooted binary trees (PRBTs)** uniformly

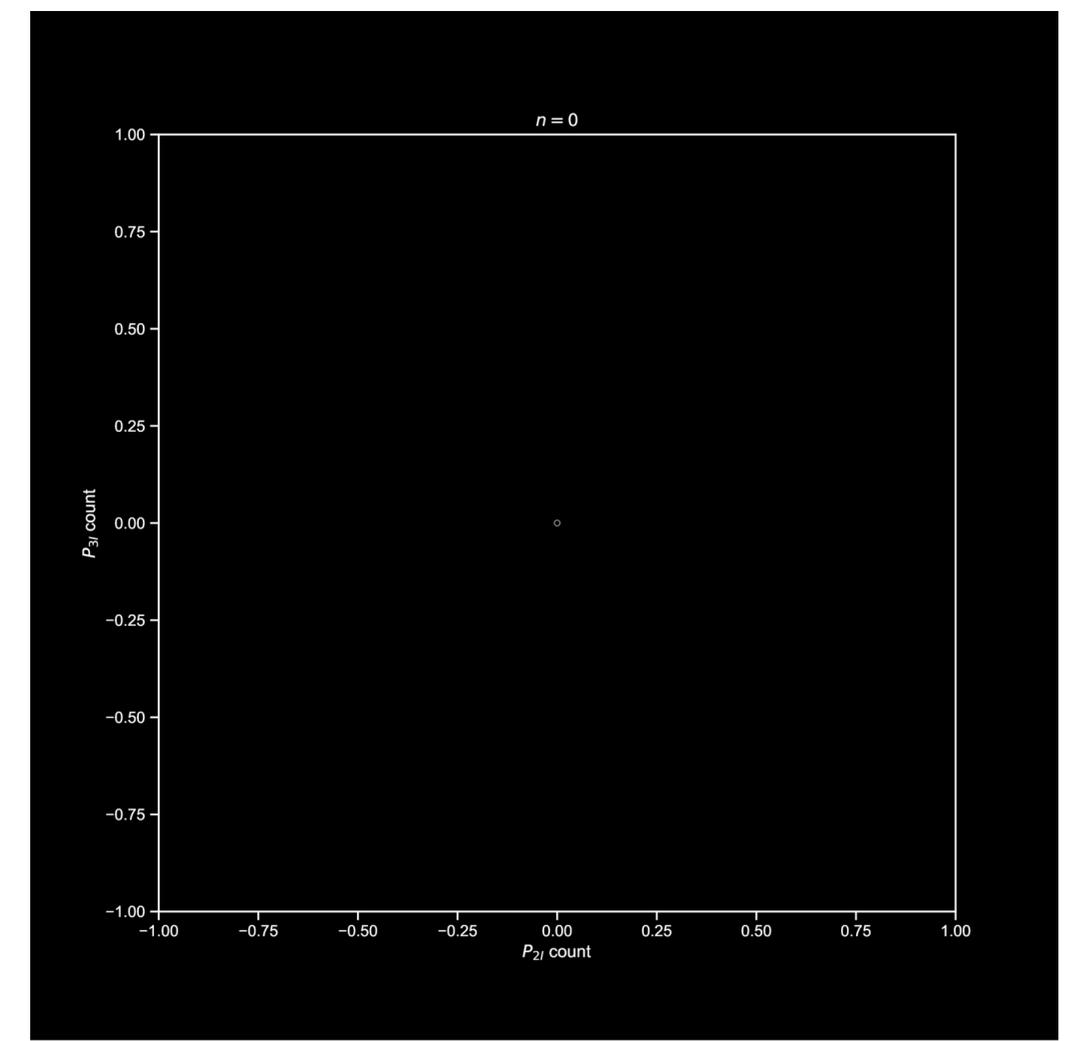
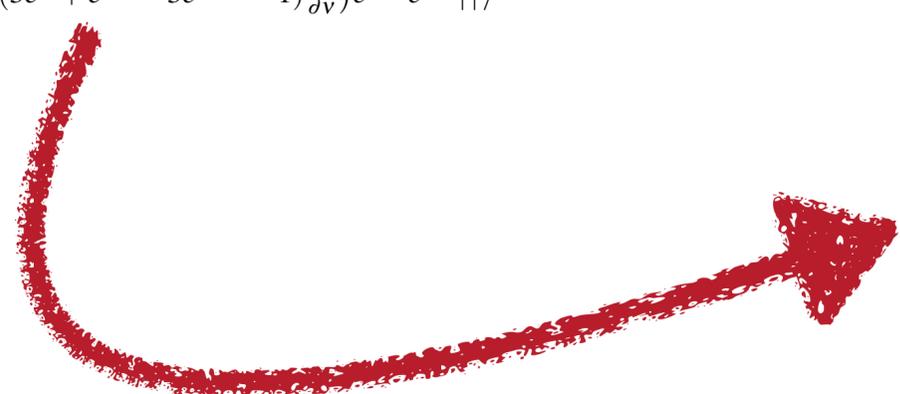
$$\hat{O}_{P_1} := \begin{array}{c} \diagup \\ | \\ * \\ \diagdown \end{array} \equiv \sum_{T \in \{I, L, R\}} \begin{array}{c} \diagup \\ | \\ T \\ \diagdown \end{array}, \quad \hat{O}_{P_2} := \begin{array}{c} \diagup \\ \diagdown \\ | \\ * \end{array} \equiv \sum_{T \in \{I, L, R\}} \begin{array}{c} \diagup \\ | \\ T \\ \diagdown \end{array}, \quad \hat{O}_{P_3} := \begin{array}{c} \diagup \\ \diagdown \\ \diagup \\ | \\ * \\ \diagdown \end{array} \equiv \sum_{T \in \{I, L, R\}} \begin{array}{c} \diagup \\ | \\ T \\ \diagdown \end{array}$$



$$\begin{aligned} \mathcal{G}(\lambda; \omega) &:= \langle | e^{\omega \cdot \hat{O}} e^{\lambda \hat{G}} | | \rangle, \quad \omega \cdot \hat{O} := \varepsilon \hat{O}_E + \gamma \hat{O}_{P_1} + \mu \hat{O}_{P_2} + \nu \hat{O}_{P_3} \\ \frac{\partial}{\partial \lambda} \mathcal{G}(\lambda; \omega) &= \langle | (e^{ad_{\omega \cdot \hat{O}}}(\hat{G})) e^{\omega \cdot \hat{O}} e^{\lambda \hat{G}} | | \rangle \stackrel{(*)}{=} \langle | (e^{ad_{\nu \hat{O}_{P_3}}} (e^{ad_{\mu \hat{O}_{P_2}}} (e^{ad_{\varepsilon \hat{O}_E + \gamma \hat{O}_{P_1}}}(\hat{G})))) e^{\omega \cdot \hat{O}} e^{\lambda \hat{G}} | | \rangle \\ &= e^{2\varepsilon + \gamma} \langle | (e^{ad_{\nu \hat{O}_{P_3}}} (e^{ad_{\mu \hat{O}_{P_2}}}(\hat{G}))) e^{\omega \cdot \hat{O}} e^{\lambda \hat{G}} | | \rangle \\ &= e^{2\varepsilon + \gamma} \langle | (e^{ad_{\nu \hat{O}_{P_3}}} (\hat{G} + (e^\mu - 1)[\hat{O}_{P_2}, \hat{G}])) e^{\omega \cdot \hat{O}} e^{\lambda \hat{G}} | | \rangle \\ &= e^{2\varepsilon + \gamma} \langle | (\hat{G} + (e^\mu - 1)[\hat{O}_{P_2}, \hat{G}] \\ &\quad + e^\nu (e^\nu - 1)[\hat{O}_{P_3}, \hat{G}] + (e^\nu - 1)(e^\mu - e^{-\nu})\hat{R}_{P_3'}) e^{\omega \cdot \hat{O}} e^{\lambda \hat{G}} | | \rangle \\ &= e^{2\varepsilon + \gamma} \langle | (2\hat{O}_E + 3(e^\mu - 1)\hat{O}_{P_1} + (4e^{\mu+\nu} - 6e^\mu + 2)\hat{O}_{P_2} \\ &\quad + (3e^\mu + e^{-\nu} - 3e^{\mu+\nu} - 1)\hat{O}_{P_3}) e^{\omega \cdot \hat{O}} e^{\lambda \hat{G}} | | \rangle \\ &= e^{2\varepsilon + \gamma} \langle | (2\frac{\partial}{\partial \varepsilon} + 3(e^\mu - 1)\frac{\partial}{\partial \gamma} + (4e^{\mu+\nu} - 6e^\mu + 2)\frac{\partial}{\partial \mu} \\ &\quad + (3e^\mu + e^{-\nu} - 3e^{\mu+\nu} - 1)\frac{\partial}{\partial \nu}) e^{\omega \cdot \hat{O}} e^{\lambda \hat{G}} | | \rangle \end{aligned}$$

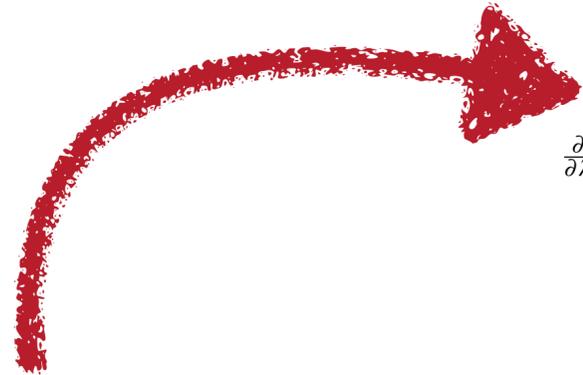


$$\begin{aligned} [\hat{O}_{P_2}, [\hat{O}_{P_2}, \hat{G}]] &= [\hat{O}_{P_2}, \hat{G}], \quad [\hat{O}_{P_2}, [\hat{O}_{P_3}, \hat{G}]] = [\hat{O}_{P_3}, \hat{G}] + \hat{R}_{P_3} \\ [\hat{O}_{P_3}, [\hat{O}_{P_3}, \hat{G}]] &= [\hat{O}_{P_3}, \hat{G}] + 2\hat{R}_{P_3'}, \quad [\hat{O}_{P_2}, \hat{R}_{P_3'}] = 0, \quad [\hat{O}_{P_3}, \hat{R}_{P_3'}] = -\hat{R}_{P_3'} \\ \langle | [\hat{O}_{P_2}, \hat{G}] | | \rangle &= \langle | (3\hat{O}_{P_1} - 2\hat{O}_{P_2}) | | \rangle, \quad \langle | [\hat{O}_{P_3}, \hat{G}] | | \rangle = \langle | (4\hat{O}_{P_2} - 3\hat{O}_{P_3}) | | \rangle, \quad \langle | \hat{R}_{P_3'} | | \rangle = \langle | \hat{O}_{P_3} | | \rangle \end{aligned}$$



Example: generating **planar rooted binary trees (PRBTs)** uniformly

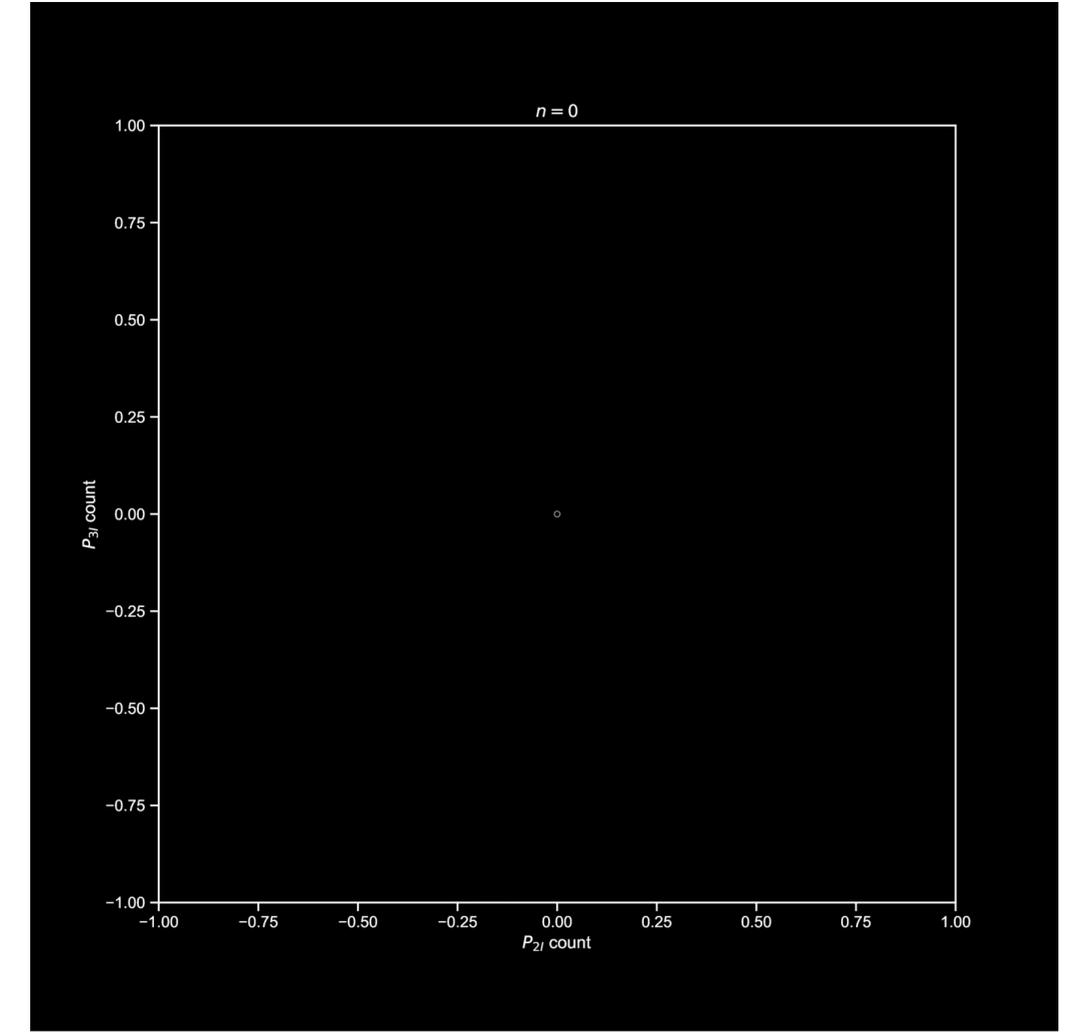
$$\hat{O}_{P_1} := \begin{array}{c} \diagup \\ | \\ * \\ \diagdown \end{array} \equiv \sum_{T \in \{I, L, R\}} \begin{array}{c} \diagup \\ | \\ T \\ \diagdown \end{array}, \quad \hat{O}_{P_2} := \begin{array}{c} \diagup \\ \diagdown \\ | \\ * \end{array} \equiv \sum_{T \in \{I, L, R\}} \begin{array}{c} \diagup \\ \diagdown \\ | \\ T \end{array}, \quad \hat{O}_{P_3} := \begin{array}{c} \diagup \\ \diagdown \\ \diagup \\ \diagdown \\ | \\ * \end{array} \equiv \sum_{T \in \{I, L, R\}} \begin{array}{c} \diagup \\ \diagdown \\ \diagup \\ \diagdown \\ | \\ T \end{array}$$



$$\begin{aligned} \mathcal{G}(\lambda; \omega) &:= \langle | e^{\omega \cdot \hat{O}} e^{\lambda \hat{G}} | | \rangle, \quad \omega \cdot \hat{O} := \varepsilon \hat{O}_E + \gamma \hat{O}_{P_1} + \mu \hat{O}_{P_2} + \nu \hat{O}_{P_3} \\ \frac{\partial}{\partial \lambda} \mathcal{G}(\lambda; \omega) &= \langle | (e^{ad_{\omega \cdot \hat{O}}}(\hat{G})) e^{\omega \cdot \hat{O}} e^{\lambda \hat{G}} | | \rangle \stackrel{(*)}{=} \langle | (e^{ad_{\nu \hat{O}_{P_3}}} (e^{ad_{\mu \hat{O}_{P_2}}} (e^{ad_{\varepsilon \hat{O}_E + \gamma \hat{O}_{P_1}}}(\hat{G})))) e^{\omega \cdot \hat{O}} e^{\lambda \hat{G}} | | \rangle \\ &= e^{2\varepsilon + \gamma} \langle | (e^{ad_{\nu \hat{O}_{P_3}}} (e^{ad_{\mu \hat{O}_{P_2}}}(\hat{G}))) e^{\omega \cdot \hat{O}} e^{\lambda \hat{G}} | | \rangle \\ &= e^{2\varepsilon + \gamma} \langle | (e^{ad_{\nu \hat{O}_{P_3}}} (\hat{G} + (e^\mu - 1)[\hat{O}_{P_2}, \hat{G}])) e^{\omega \cdot \hat{O}} e^{\lambda \hat{G}} | | \rangle \\ &= e^{2\varepsilon + \gamma} \langle | (\hat{G} + (e^\mu - 1)[\hat{O}_{P_2}, \hat{G}] \\ &\quad + e^\nu (e^\nu - 1)[\hat{O}_{P_3}, \hat{G}] + (e^\nu - 1)(e^\mu - e^{-\nu}) \hat{R}_{P_3'}) e^{\omega \cdot \hat{O}} e^{\lambda \hat{G}} | | \rangle \\ &= e^{2\varepsilon + \gamma} \langle | (2\hat{O}_E + 3(e^\mu - 1)\hat{O}_{P_1} + (4e^{\mu+\nu} - 6e^\mu + 2)\hat{O}_{P_2} \\ &\quad + (3e^\mu + e^{-\nu} - 3e^{\mu+\nu} - 1)\hat{O}_{P_3}) e^{\omega \cdot \hat{O}} e^{\lambda \hat{G}} | | \rangle \\ &= e^{2\varepsilon + \gamma} \langle | (2\frac{\partial}{\partial \varepsilon} + 3(e^\mu - 1)\frac{\partial}{\partial \gamma} + (4e^{\mu+\nu} - 6e^\mu + 2)\frac{\partial}{\partial \mu} \\ &\quad + (3e^\mu + e^{-\nu} - 3e^{\mu+\nu} - 1)\frac{\partial}{\partial \nu}) e^{\omega \cdot \hat{O}} e^{\lambda \hat{G}} | | \rangle \end{aligned}$$

$$\begin{aligned} [\hat{O}_{P_2}, \hat{G}] &= \begin{array}{c} \diagup \\ | \\ \bullet \\ \diagdown \end{array} + \begin{array}{c} \bullet \\ \diagup \\ | \\ \bullet \\ \diagdown \end{array} + \begin{array}{c} \bullet \\ \diagdown \\ | \\ \bullet \\ \diagup \end{array} - \begin{array}{c} \bullet \\ \diagup \\ \diagdown \\ | \\ \bullet \end{array} - \begin{array}{c} \bullet \\ \diagdown \\ \diagup \\ | \\ \bullet \end{array} \\ [\hat{O}_{P_3}, \hat{G}] &= \begin{array}{c} \bullet \\ \diagup \\ \diagdown \\ | \\ \bullet \end{array} + \begin{array}{c} \bullet \\ \diagdown \\ \diagup \\ | \\ \bullet \end{array} + \begin{array}{c} \bullet \\ \diagup \\ \diagdown \\ \diagup \\ | \\ \bullet \end{array} + \begin{array}{c} \bullet \\ \diagdown \\ \diagup \\ \diagdown \\ | \\ \bullet \end{array} - \begin{array}{c} \bullet \\ \diagup \\ \diagdown \\ \diagup \\ \diagdown \\ | \\ \bullet \end{array} - \begin{array}{c} \bullet \\ \diagdown \\ \diagup \\ \diagdown \\ \diagup \\ | \\ \bullet \end{array} - \hat{R}_{P_3'} \end{aligned}$$

$$\begin{aligned} [\hat{O}_{P_2}, [\hat{O}_{P_2}, \hat{G}]] &= [\hat{O}_{P_2}, \hat{G}], \quad [\hat{O}_{P_2}, [\hat{O}_{P_3}, \hat{G}]] = [\hat{O}_{P_3}, \hat{G}] + \hat{R}_{P_3} \\ [\hat{O}_{P_3}, [\hat{O}_{P_3}, \hat{G}]] &= [\hat{O}_{P_3}, \hat{G}] + 2\hat{R}_{P_3'}, \quad [\hat{O}_{P_2}, \hat{R}_{P_3'}] = 0, \quad [\hat{O}_{P_3}, \hat{R}_{P_3'}] = -\hat{R}_{P_3'} \\ \langle | [\hat{O}_{P_2}, \hat{G}] | \rangle &= \langle | (3\hat{O}_{P_1} - 2\hat{O}_{P_2}) | \rangle, \quad \langle | [\hat{O}_{P_3}, \hat{G}] | \rangle = \langle | (4\hat{O}_{P_2} - 3\hat{O}_{P_3}) | \rangle, \quad \langle | \hat{R}_{P_3'} | \rangle = \langle | \hat{O}_{P_3} | \rangle \end{aligned}$$



Combinatorial evolution equations

The **formal EMGF evolution equation** for $\mathcal{G}(\lambda; \underline{\omega})$ reads as follows:

$$\frac{\partial}{\partial \lambda} \mathcal{G}(\lambda; \underline{\omega}) = \langle | \left(e^{ad_{\underline{\omega} \cdot \hat{O}} \hat{G}} \right) e^{\underline{\omega} \cdot \hat{O}} e^{\lambda \hat{G}} | X_0 \rangle \quad (ad_A(B) := AB - BA) \quad (15)$$

Applying the version of the **jump-closure theorem** appropriate for the chosen rewriting semantics (DPO or SqPO), the above formal evolution equation may be converted into a proper **evolution equation on formal power series** if the following **polynomial jump-closure** holds:

$$(PJC') \quad \forall q \in \mathbb{Z}_{\geq 0} : \exists \underline{N}(n) \in \mathbb{Z}_{\geq 0}^m, \gamma_q(\underline{\omega}, \underline{k}) \in \mathbb{R} : \langle | ad_{\underline{\omega} \cdot \hat{O}}^{\circ q}(\hat{G}) = \sum_{\underline{k}=0}^{\underline{N}(q)} \gamma_{\underline{k}}(\underline{\omega}, \underline{k}) \langle | \hat{O}^{\underline{k}} \quad (16)$$

If a given set of observables satisfies (PJC'), the **formal evolution equation** (12) for the EMGF $\mathcal{G}(\lambda; \underline{\omega})$ may be refined into

$$\frac{\partial}{\partial \lambda} \mathcal{G}(\lambda; \underline{\omega}) = \mathbb{G}(\underline{\omega}, \underline{\partial \omega}) \mathcal{G}(\lambda; \underline{\omega}), \quad \mathbb{G}(\underline{\omega}, \underline{\partial \omega}) = \left(\langle | e^{ad_{\underline{\omega} \cdot \hat{O}}(\hat{G})} \right) \Big|_{\hat{O} \mapsto \underline{\partial \omega}}. \quad (17)$$

Combinatorial evolution equations

The **formal EMGF evolution equation** for $\mathcal{G}(\lambda; \underline{\omega})$ reads as follows:

$$\frac{\partial}{\partial \lambda} \mathcal{G}(\lambda; \underline{\omega}) = \langle | \left(e^{ad_{\underline{\omega} \cdot \hat{O}} \hat{G}} \right) e^{\underline{\omega} \cdot \hat{O}} e^{\lambda \hat{G}} | X_0 \rangle \quad (ad_A(B) := AB - BA) \quad (15)$$

Applying the version of the **jump-closure theorem** appropriate for the chosen rewriting semantics (DPO or SqPO), the above formal evolution equation may be converted into a proper **evolution equation on formal power series** if the following **polynomial jump-closure** holds:

$$(PJC') \quad \forall q \in \mathbb{Z}_{\geq 0} : \exists \underline{N}(n) \in \mathbb{Z}_{\geq 0}^m, \gamma_q(\underline{\omega}, \underline{k}) \in \mathbb{R} : \langle | ad_{\underline{\omega} \cdot \hat{O}}^{\circ q}(\hat{G}) = \sum_{k=0}^{\underline{N}(q)} \gamma_{\underline{k}}(\underline{\omega}, \underline{k}) \langle | \hat{O}^{\underline{k}} \quad (16)$$

If a given set of observables satisfies (PJC'), the **formal evolution equation** (12) for the EMGF $\mathcal{G}(\lambda; \underline{\omega})$ may be refined into

$$\frac{\partial}{\partial \lambda} \mathcal{G}(\lambda; \underline{\omega}) = \mathbb{G}(\underline{\omega}, \underline{\partial \omega}) \mathcal{G}(\lambda; \underline{\omega}), \quad \mathbb{G}(\underline{\omega}, \underline{\partial \omega}) = \left(\langle | e^{ad_{\underline{\omega} \cdot \hat{O}}(\hat{G})} \right) \Big|_{\hat{O} \mapsto \underline{\partial \omega}}. \quad (17)$$

Plan of the talk:

- I. **A Case Study in Applied Category Theory:
from Categorical Rewriting to
Rule-algebraic Combinatorics**
- II. **The `coreact.wiki` Initiative**

CoREACT

Coq-based Rewriting: towards Executable Applied Category Theory

Consortium: IRIF (UP), LIP (ENS-Lyon), LIX (École Polytechnique), Sophia-Antipolis (Inria)

Partner	Last name	First name
Université de Paris	BEHR	Nicolas
	GALLEGO	Emilio
	GHEERBRANT	Amélie
	HERBELIN	Hugo
	MELLIÈS	Paul-André
	ROGOVA	Alexandra
ENS-Lyon	HARMER	Russell
	HIRSCHOWITZ	Tom
	POUS	Damien
École Polytechnique	MIMRAM	Samuel
	WERNER	Benjamin
	ZEILBERGER	Noam
Inria Sophia-Antipolis	BERTOT	Yves
	COHEN	Cyril
	TASSI	Enrico



coreact.wiki

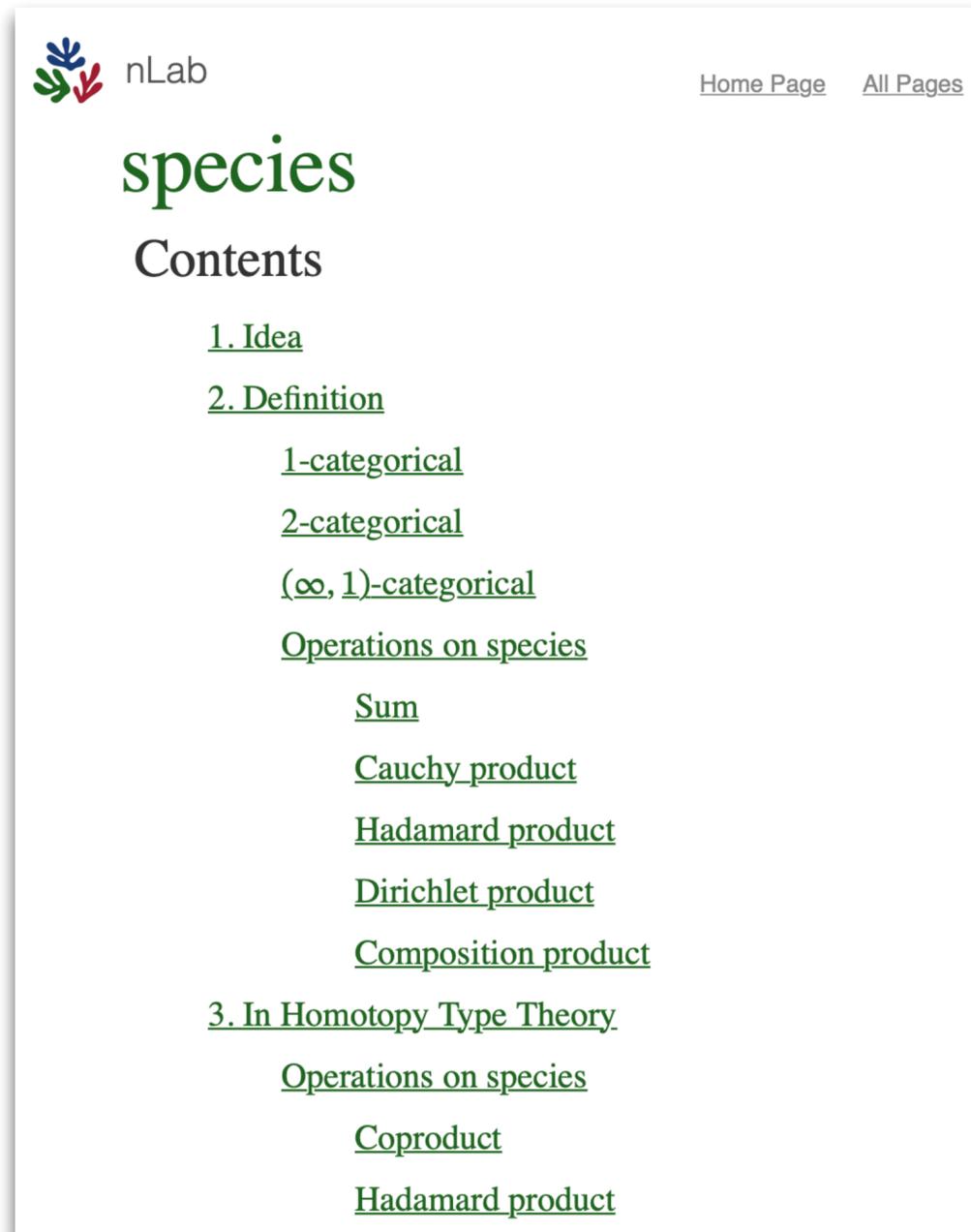
Main objectives of the CoREACT/GReTA ExACT initiative

- Development of a methodology for **diagrammatic reasoning in Coq**
- **Formalization** (in Coq) and **certification** of a representative collection of axioms and theorems for **compositional categorical rewriting theory**
- Development of a **Coq-enabled interactive database and wiki system**
- Development of a CoREACT wiki-based “**proof-by-pointing**” **engine**
- Executable **reference prototype algorithms** from categorical structures in Coq (via the use of SMT solvers/theorem provers such as Z3)

Main objectives of the CoREACT/GReTA ExACT initiative

- Development of a methodology for **diagrammatic reasoning in Coq**
- **Formalization** (in Coq) and **certification** of a representative collection of axioms and theorems for **compositional categorical rewriting theory**
- Development of a **Coq-enabled interactive database and wiki system**
- Development of a CoREACT wiki-based “**proof-by-pointing**” **engine**
- Executable **reference prototype algorithms** from categorical structures in Coq (via the use of SMT solvers/theorem provers such as Z3)

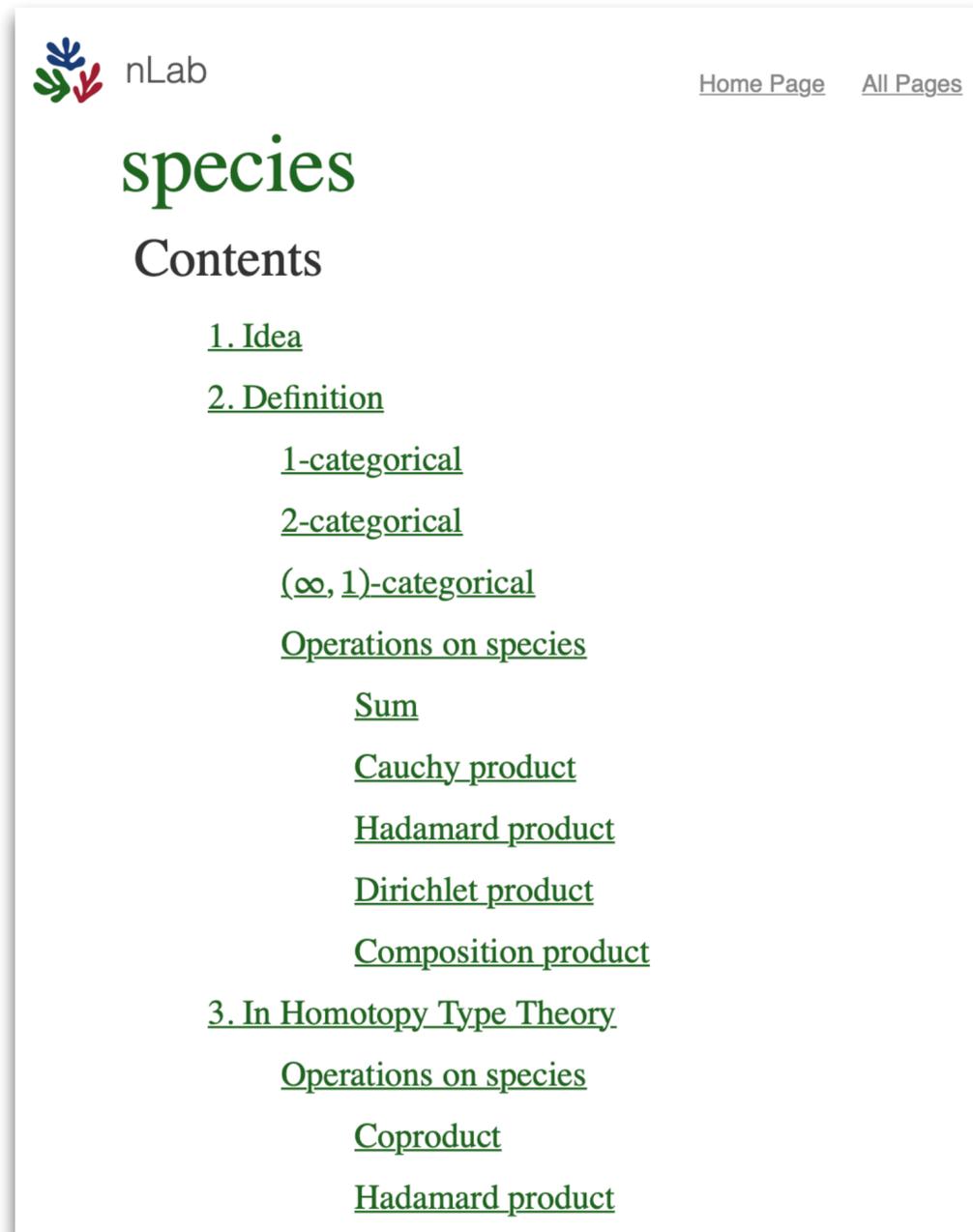
A (very non-exhaustive!) view on **wiki svstems in mathematics/ (A)CT**



The screenshot shows the nLab wiki page for 'species'. At the top left is the nLab logo (a stylized tree with blue, green, and red leaves) and the text 'nLab'. To the right are links for 'Home Page' and 'All Pages'. The main title 'species' is in a large green font. Below it is the word 'Contents' in a smaller black font. The page is organized into three numbered sections: '1. Idea', '2. Definition', and '3. In Homotopy Type Theory'. Each section contains several sub-links, all underlined in green. Under '1. Idea', there are links for '1-categorical', '2-categorical', and '(\infty, 1)-categorical'. Under '2. Definition', there is a link for 'Operations on species', which then branches into 'Sum', 'Cauchy product', 'Hadamard product', 'Dirichlet product', and 'Composition product'. Under '3. In Homotopy Type Theory', there is a link for 'Operations on species', which branches into 'Coproduct' and 'Hadamard product'.

<https://ncatlab.org>

A (very non-exhaustive!) view on **wiki systems in mathematics/ (A)CT**



nLab

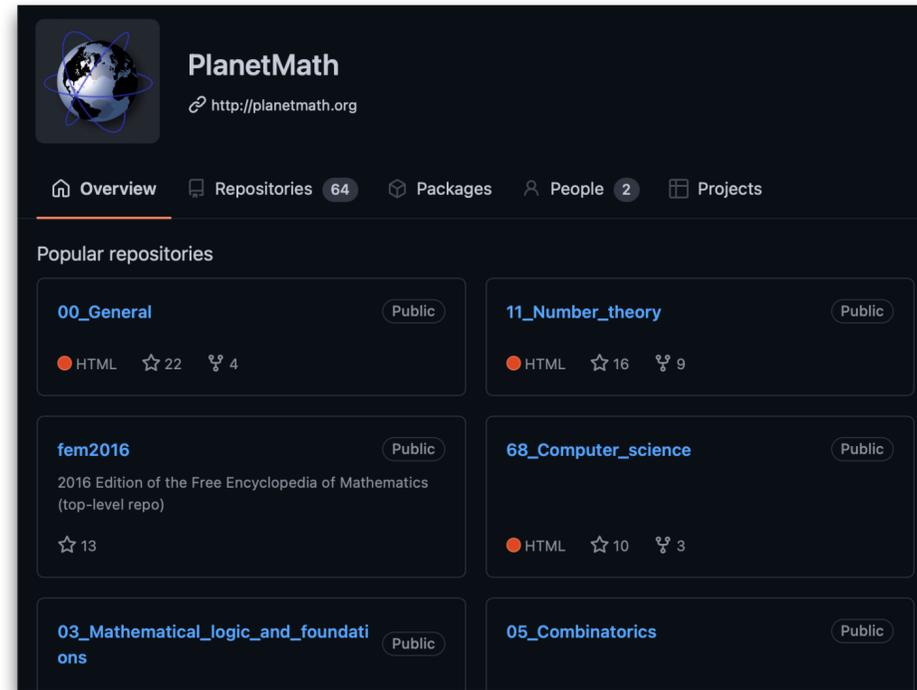
Home Page All Pages

species

Contents

- [1. Idea](#)
- [2. Definition](#)
 - [1-categorical](#)
 - [2-categorical](#)
 - [\$\(\infty, 1\)\$ -categorical](#)
 - [Operations on species](#)
 - [Sum](#)
 - [Cauchy product](#)
 - [Hadamard product](#)
 - [Dirichlet product](#)
 - [Composition product](#)
- [3. In Homotopy Type Theory](#)
 - [Operations on species](#)
 - [Coproduct](#)
 - [Hadamard product](#)

<https://ncatlab.org>



PlanetMath

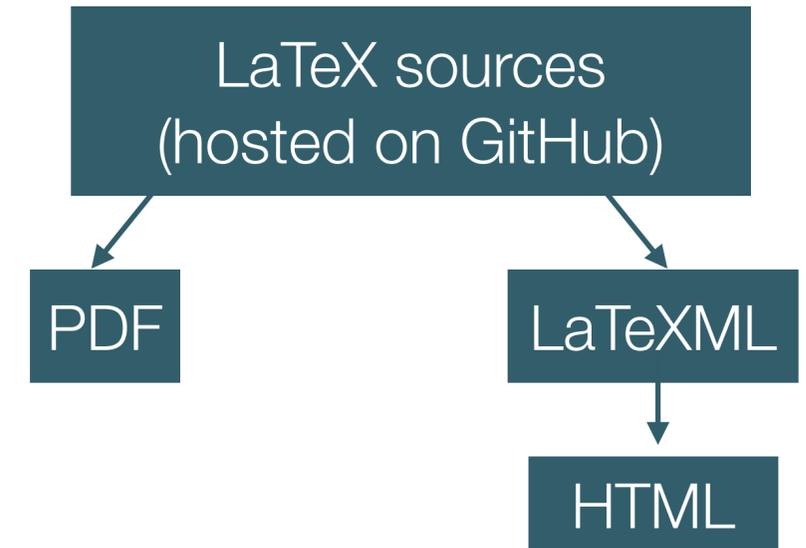
<http://planetmath.org>

Overview Repositories 64 Packages People 2 Projects

Popular repositories

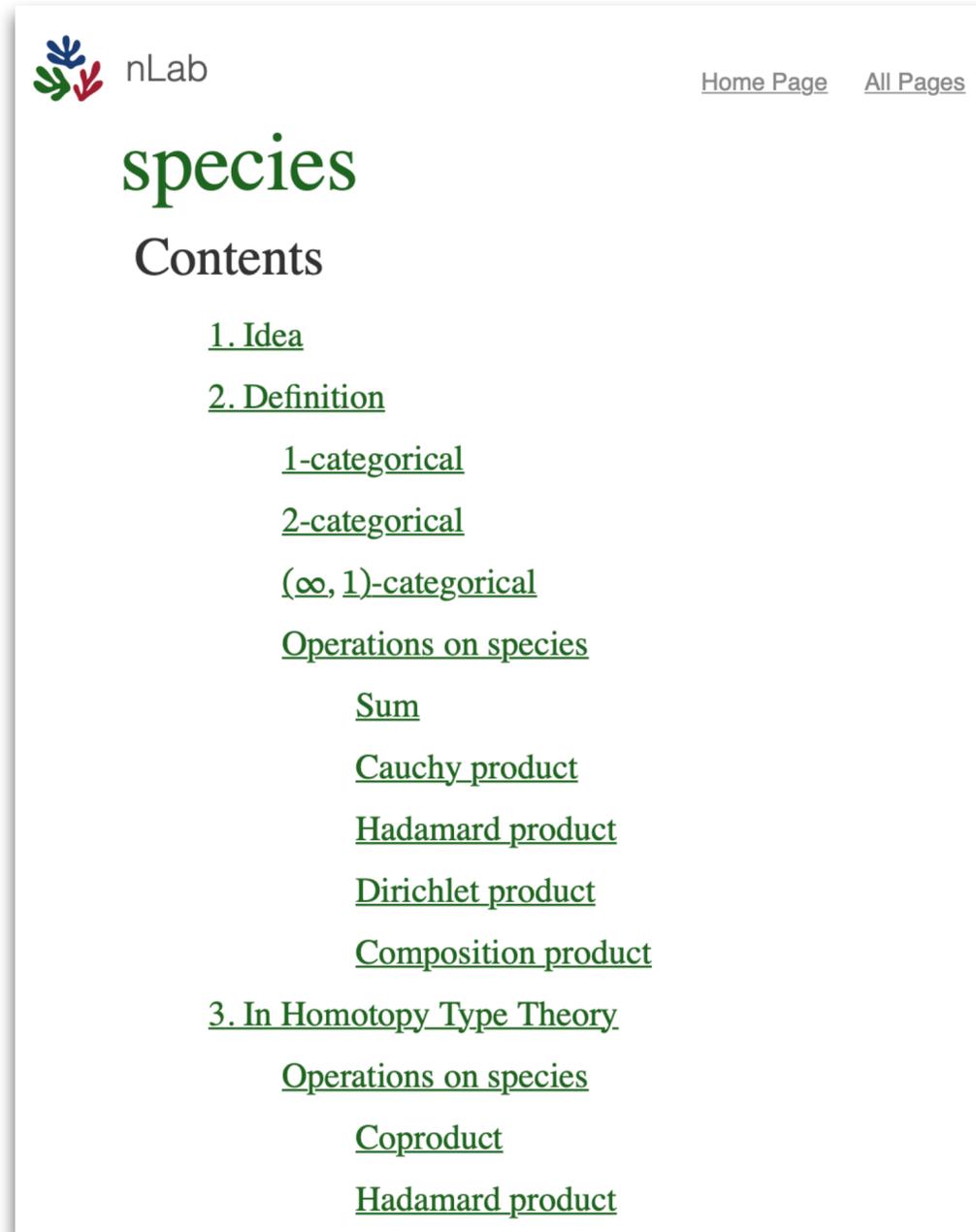
- [00_General](#) Public HTML ☆ 22 🗨 4
- [11_Number_theory](#) Public HTML ☆ 16 🗨 9
- [fem2016](#) Public 2016 Edition of the Free Encyclopedia of Mathematics (top-level repo) ☆ 13
- [68_Computer_science](#) Public HTML ☆ 10 🗨 3
- [03_Mathematical_logic_and_foundations](#) Public
- [05_Combinatorics](#) Public

<https://planetmath.org>



(semi-) automatic cross-linking
provided via NNexus system

A (very non-exhaustive!) view on **wiki systems in mathematics/ (A)CT**



nLab

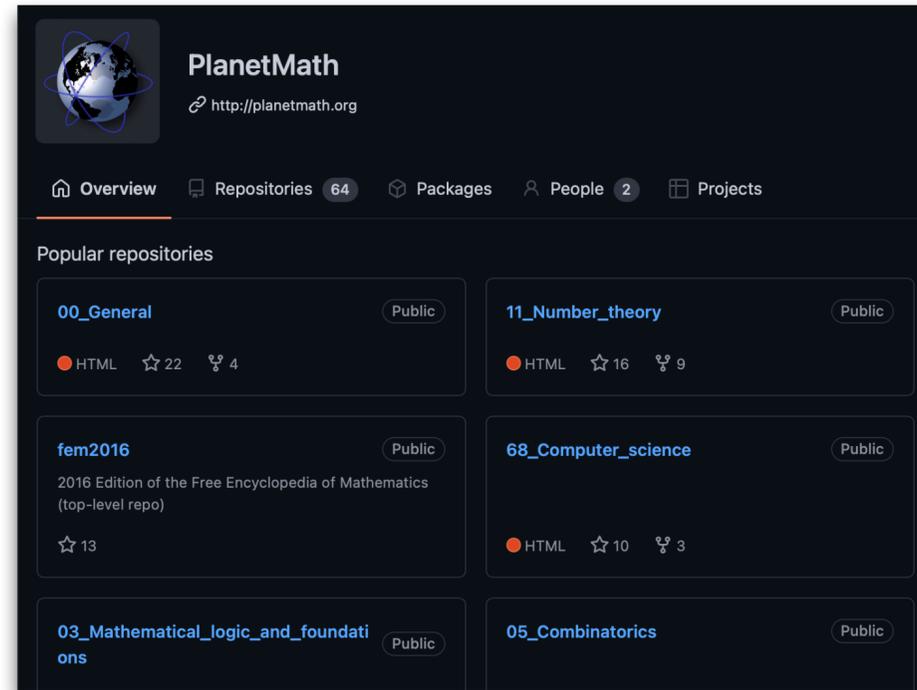
Home Page All Pages

species

Contents

- [1. Idea](#)
- [2. Definition](#)
 - [1-categorical](#)
 - [2-categorical](#)
 - [\$\(\infty, 1\)\$ -categorical](#)
 - [Operations on species](#)
 - [Sum](#)
 - [Cauchy product](#)
 - [Hadamard product](#)
 - [Dirichlet product](#)
 - [Composition product](#)
- [3. In Homotopy Type Theory](#)
 - [Operations on species](#)
 - [Coproduct](#)
 - [Hadamard product](#)

<https://ncatlab.org>



PlanetMath

<http://planetmath.org>

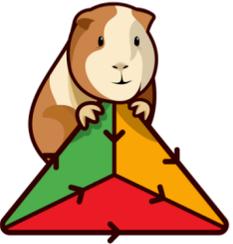
Overview Repositories 64 Packages People 2 Projects

Popular repositories

- [00_General](#) Public HTML ☆ 22 🗨 4
- [11_Number_theory](#) Public HTML ☆ 16 🗨 9
- [fem2016](#) Public 2016 Edition of the Free Encyclopedia of Mathematics (top-level repo) ☆ 13
- [68_Computer_science](#) Public HTML ☆ 10 🗨 3
- [03_Mathematical_logic_and_foundations](#) Public
- [05_Combinatorics](#) Public

<https://planetmath.org>

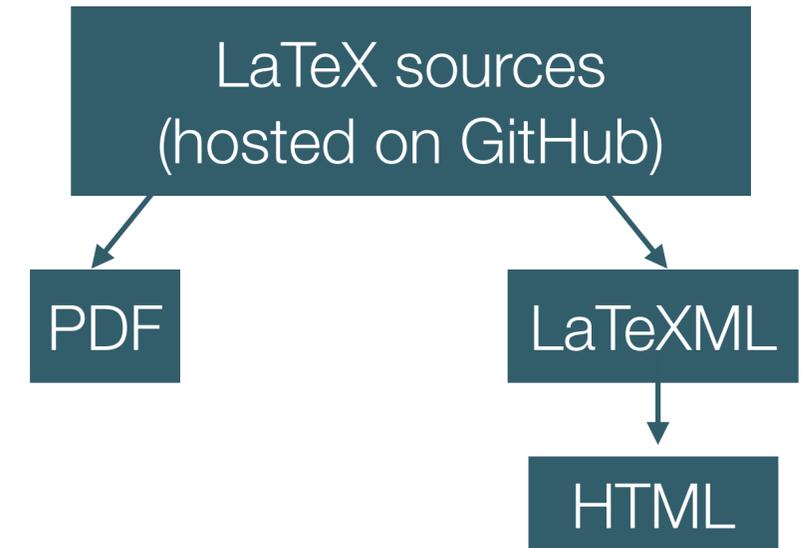




Kerodon

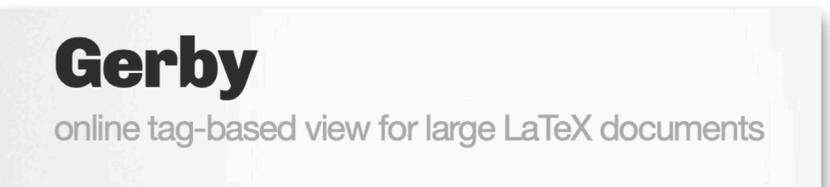
an online resource for homotopy-coherent mathematics

<https://kerodon.net>



(semi-) automatic cross-linking
provided via NNexus system

- J. Lurie's **online textbook** on categorical homotopy theory
- technology based upon **online tags view** via the Gerby system



Gerby
online tag-based view for large LaTeX documents

<https://gerby-project.github.io>

A (very non-exhaustive!) view on **proof assistants in mathematics**



<https://github.com/agda>

Hu Jason, CPP21: "Formalizing Category Theory in Agda"

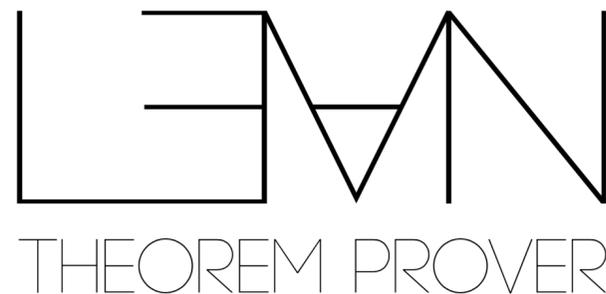
<https://youtu.be/a2txkoybw2M>



<https://coq.inria.fr>

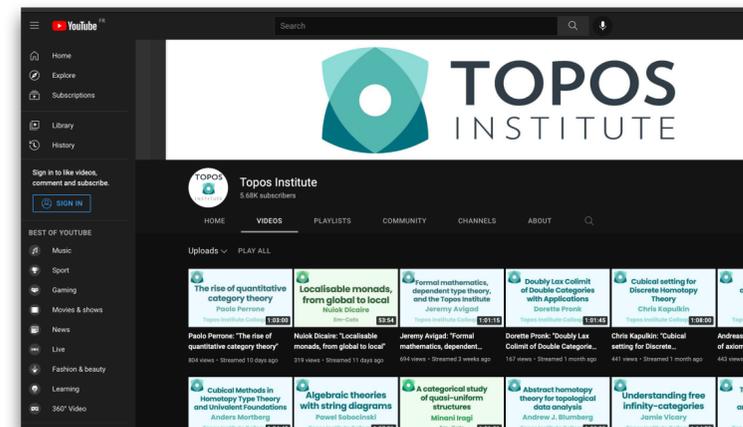
EPIT Spring School on HoTT: **Bas Spitters** Part 1 (Introduction to Coq and HoTT)

<https://youtu.be/k8T9L0qR38o>



Microsoft Research

<https://leanprover.github.io>



Jeremy Avigad: "Formal mathematics, dependent type theory, and the Topos Institute"

<https://youtu.be/Kpa8cCUZLms>

Kevin Buzzard: "What is the point of Lean's maths library?"

https://youtu.be/a1Byz_LoANE

A (very non-exhaustive!) view on **proof assistants in mathematics**



<https://github.com/agda>

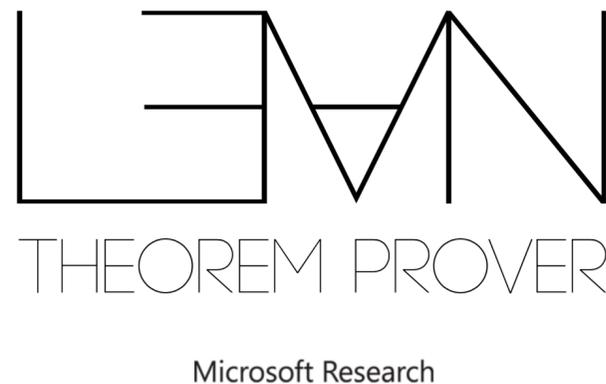
Hu Jason, CPP21: "Formalizing Category Theory in Agda"

<https://youtu.be/a2txkoybw2M>

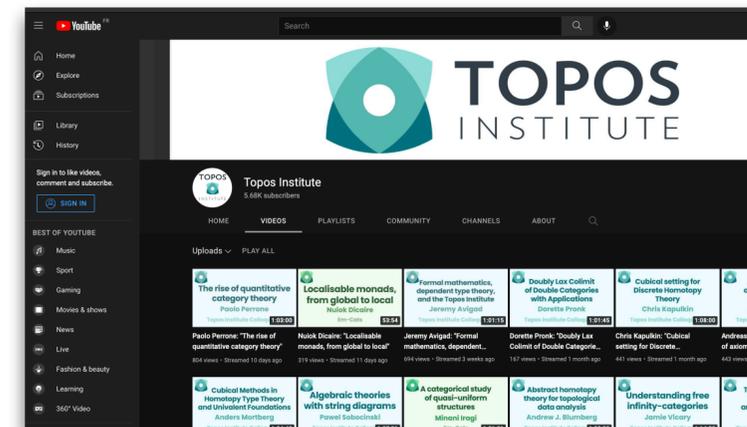


EPIT Spring School on HoTT: **Bas Spitters** Part 1 (Introduction to Coq and HoTT)

<https://youtu.be/k8T9L0qR38o>



<https://leanprover.github.io>



Jeremy Avigad: "Formal mathematics, dependent type theory, and the Topos Institute"

<https://youtu.be/Kpa8cCUZLms>

Kevin Buzzard: "What is the point of Lean's maths library?"

https://youtu.be/a1Byz_LoANE

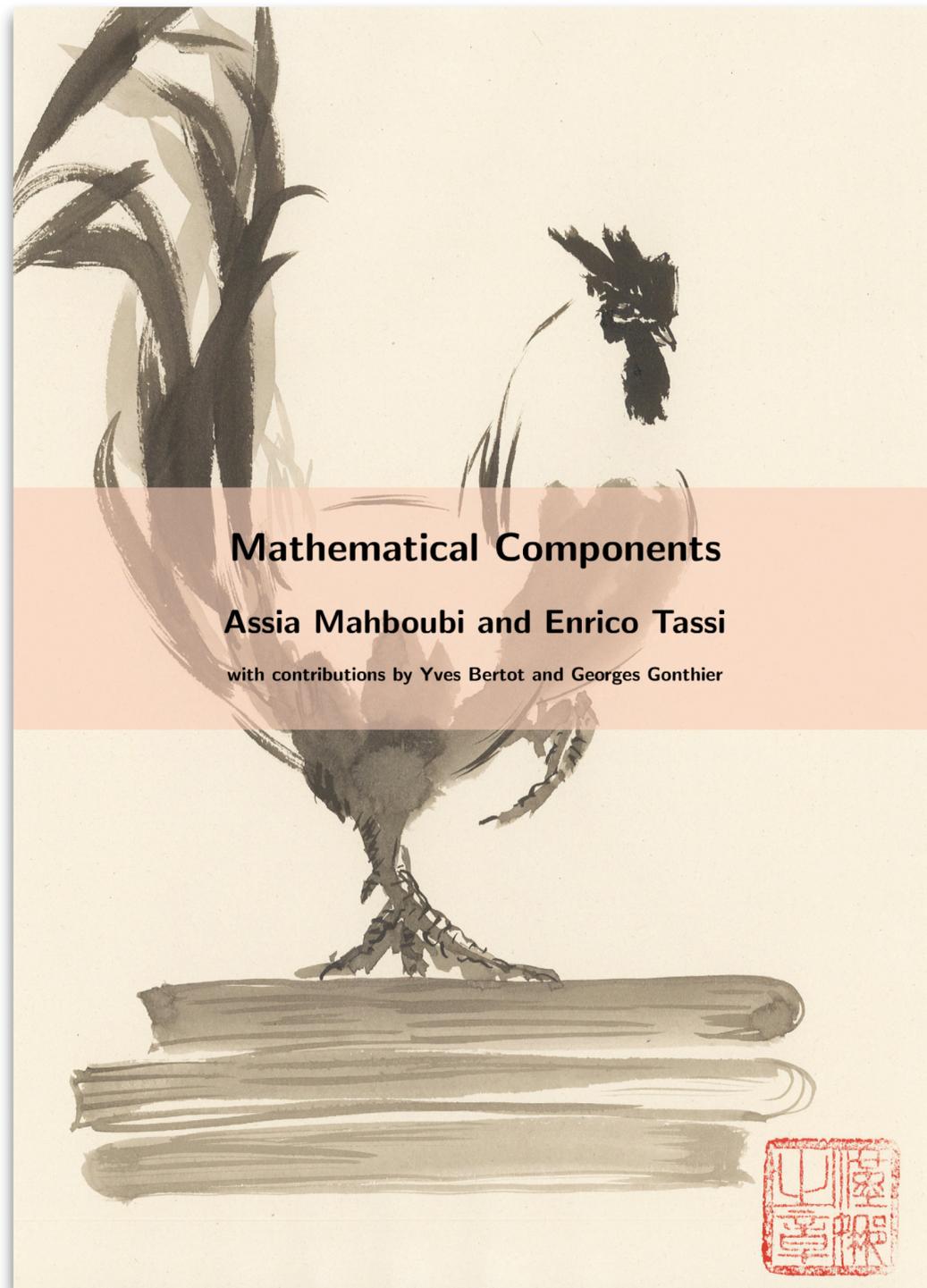
Coq — overview

The Coq Proof Assistant

<https://coq.inria.fr>

- **1984** implementation of the **Calculus of Constructions** at INRIA-Rocquencourt by **Thierry Coquand** and **G erard Huet**
- **1991** **Calculus of Inductive Constructions (CIC)** by **Christine Paulin-Mohring**
- **2002** completion of the **four color theorem proof in Coq** by **Georges Gonthier** and **Benjamin Werner** (start of the `SSReflect` development)
- (...)
- **2012** completion of the **Feit–Thompson theorem** by **Georges Gonthier et al.**
- (...)
- more than **200 people** contributed over the past >30 years (most recent stable version: **8.14**)

Coq – famous milestones



SOFTWARE FOUNDATIONS

The Software Foundations series is a broad introduction to the mathematical underpinnings of reliable software.

The principal novelty of the series is that every detail is one hundred percent formalized and machine-checked: the entire text of each volume, including the exercises, is literally a "proof script" for the Coq proof assistant.

The exposition is intended for a broad range of readers, from advanced undergraduates to PhD students and researchers. No specific background in logic or programming languages is assumed, though a degree of mathematical maturity is helpful. A one-semester course can expect to cover *Logical Foundations* plus most of *Programming Language Foundations* or *Verified Functional Algorithms*, or selections from both.

Volume 1

Logical Foundations is the entry-point to the series. It covers functional programming, basic concepts of logic, computer-assisted theorem proving, and Coq.



SOFTWARE FOUNDATIONS
Volume 1
Logical Foundations
Benjamin C. Pierce
with
Arthur Azevedo de Amorim
Chris Casinghino
Marco Gaboardi
Michael Greenberg
Catalin Hritcu
Wilhelm Sjöberg
Brent Yorgey
with
Loris D'Amorim, Andrew W. Appel, Arthur Chargueraud, Anthony Cowley, Jeffrey Foster, Dmitri Galbraith, Michael Hicks, Rangajit Jhala, Greg Morrisett, Jennifer Panik, Mikolaj Reijthart, Chang-Hee Shim, Leonid Spector, Andrew Tolmach, Stephanie Weirich, and Steve Zdancewic

Volume 2

Programming Language Foundations surveys the theory of programming languages, including operational semantics, Hoare logic, and static type systems.



SOFTWARE FOUNDATIONS
Volume 2
Programming Language Foundations
Benjamin C. Pierce
with
Arthur Azevedo de Amorim
Chris Casinghino
Marco Gaboardi
Michael Greenberg
Catalin Hritcu
Wilhelm Sjöberg
Andrew Tolmach
Brent Yorgey
with
Andrew Tolmach
Loris D'Amorim, Andrew W. Appel, Arthur Chargueraud, Anthony Cowley, Jeffrey Foster, Dmitri Galbraith, Michael Hicks, Rangajit Jhala, Greg Morrisett, Jennifer Panik, Mikolaj Reijthart, Chang-Hee Shim, Leonid Spector, Stephanie Weirich, and Steve Zdancewic

Volume 3

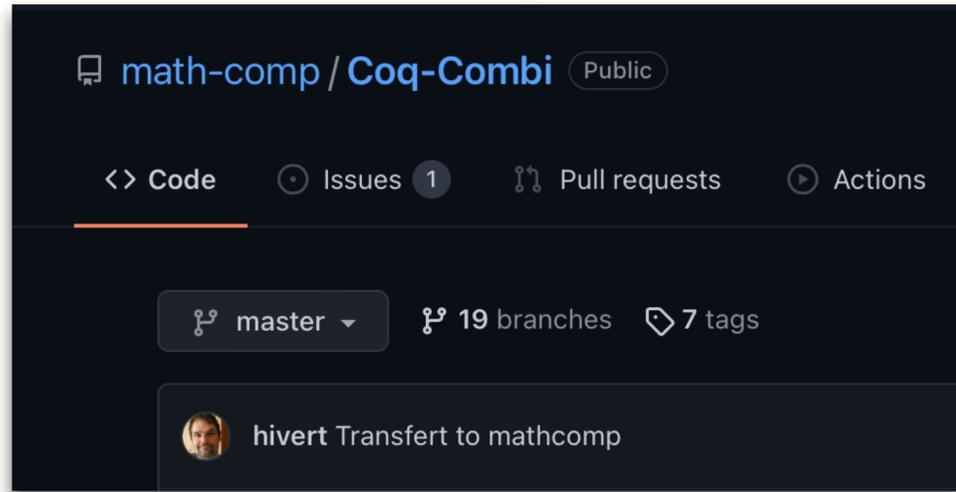
Verified Functional Algorithms shows how a

Volume 4

QuickCheck: Property-Based Testing in Coq

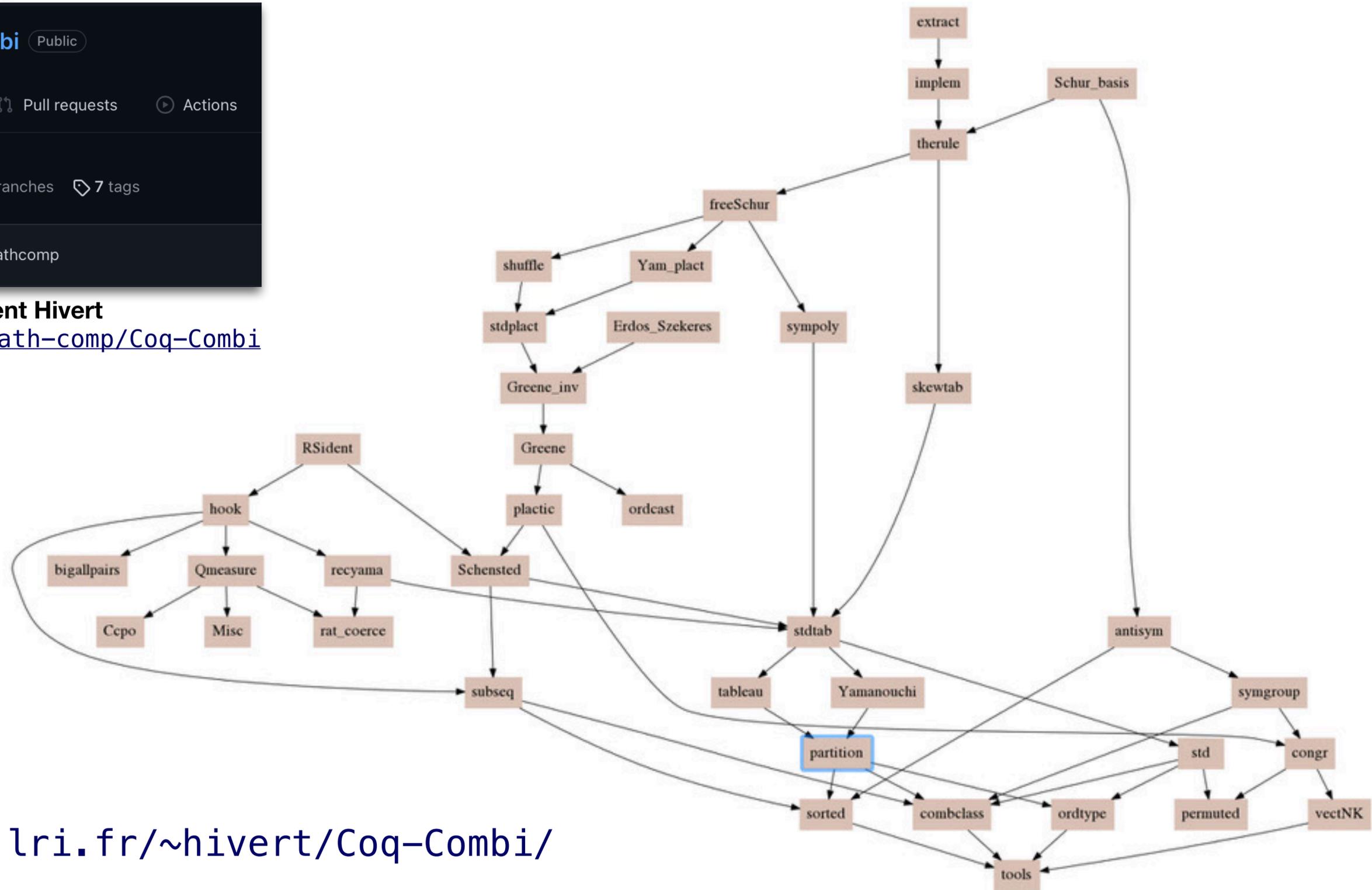
Coq-combi

Algebraic Combinatorics in Coq/SSReflect Documentation



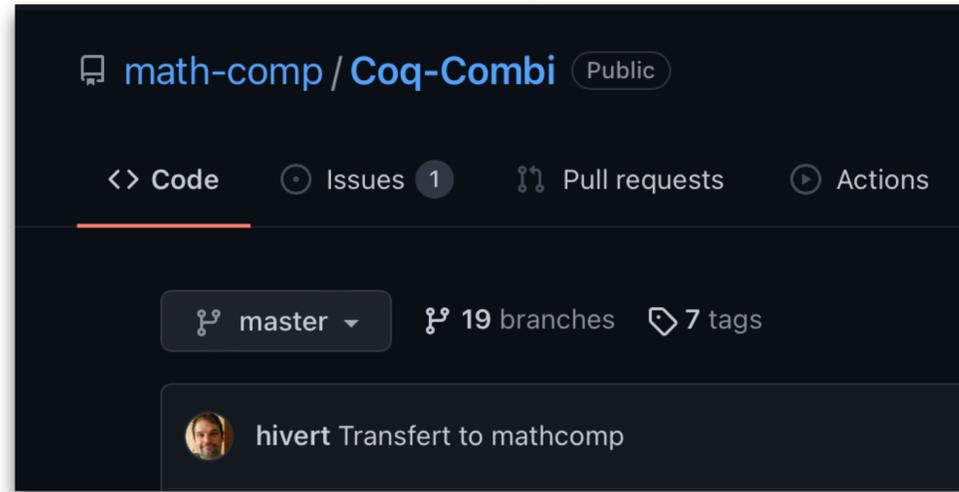
Author: **Florent Hivert**

<https://github.com/math-comp/Coq-Combi>



<https://www.lri.fr/~hivert/Coq-Combi/>

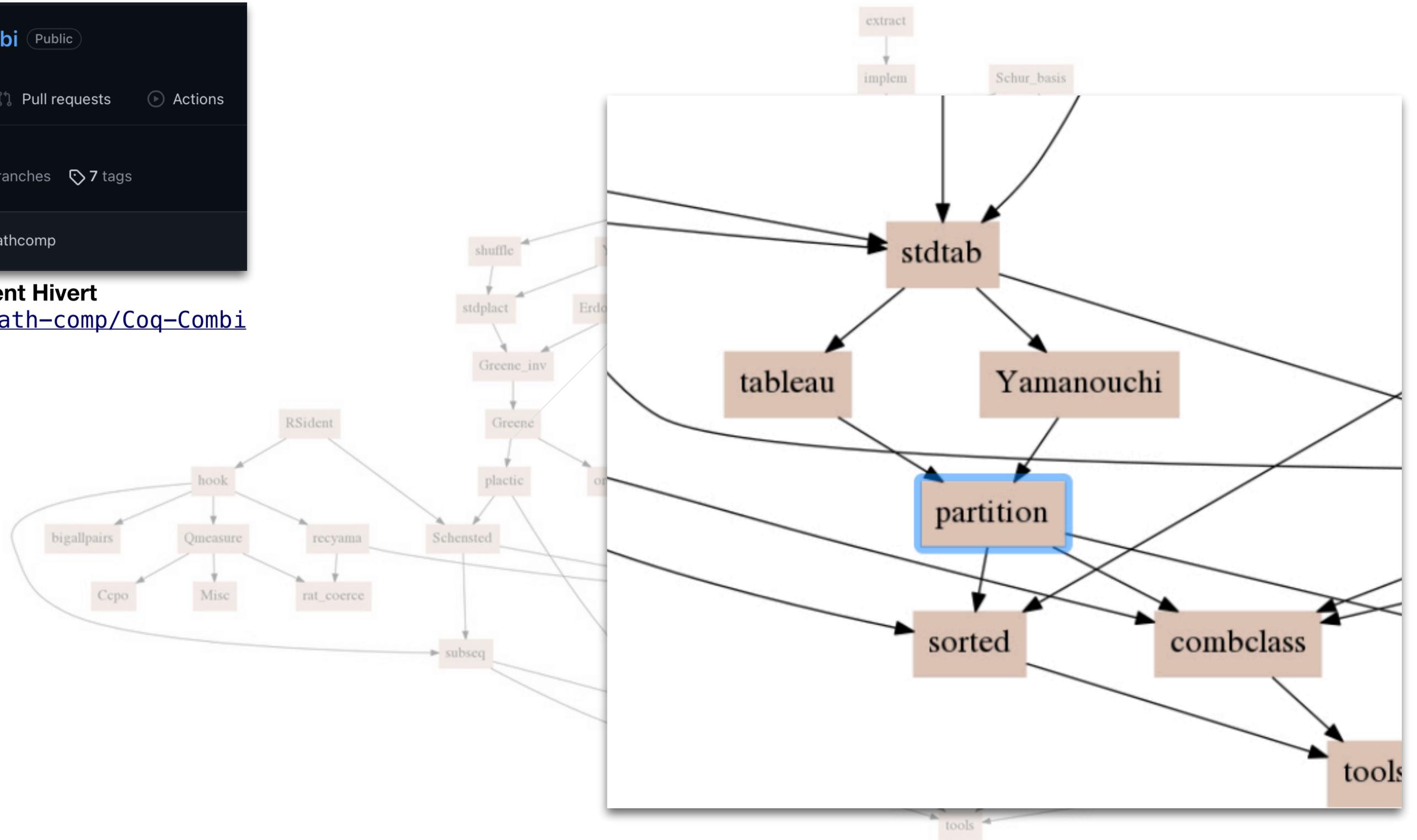
Coq-combi



Author: **Florent Hivert**

<https://github.com/math-comp/Coq-Combi>

Algebraic Combinatorics in Coq/SSReflect Documentation



Coq-combi

Combi.Combi.partition: Integer Partitions

- Shapes and Integer Partitions
- Shapes
 - A finite type `finType` for coordinate of boxes inside a shape
 - Rewriting `bigops` running along the boxes of a shape
 - Adding a box to a shape
- Integer Partitions
 - Definitions and basic properties
 - Corners, adding and removing corners
 - Conjugate of a partition
 - Partial sum of partitions
- Inclusion of Partitions and Skew Partitions
- Sigma Types for Partitions
- Counting functions
- TODO: Generalize and move in `finOrdType`

Coq-combi

Shapes and Integer Partitions

Partitions (and more generally shapes) are stored by terms of type `seq (seq nat)`. We define the following predicates and operations on `seq (seq nat)`: (r, c) is in `sh` if `r < sh[i]`

- `is_in_shape sh r c` == the box with coordinate (r, c) belongs to the shape `sh`, that is: `c < sh[r]`.
- `is_box_in_shape (r, c)` == uncurried version: same as `is_in_shape sh r c`.
- `box_in sh` == a sigma type for boxes in `sh`: `{ b | is_box_in_shape sh b }` is canonically a `subFinType`.
- `enum_box_in sh` == a full duplicate free list of the boxes in `sh`.

Integer Partitions:

- `is_part sh` == `sh` is a partition
- `rem_trail0 sh` == remove the trailing zeroes of a shape
- `is_add_corner sh i` == `i` is the row of an addable corner of `sh`
- `is_rem_corner sh i` == `i` is the row of a removable corner of `sh`
- `incr_nth sh i` == the shape obtained by adding a box at the end of the `i`-th row. This gives a partition if `i` is an addable corner of `sh` (Lemma `is_part_incr_nth`)
- `decr_nth sh i` == the shape obtained by removing a box at the end of the `i`-th row. This gives a partition if `i` is an removable corner of `sh`

Sigma Types for Partitions

Section `PartCombClass`.

```
Structure intpart : Type := IntPart {pval := seq nat; _ : is_part pval}.
Canonical intpart_subType := Eval hnf in [subType for pval].
Definition intpart_eqMixin := Eval hnf in [eqMixin of intpart by <:].
Canonical intpart_eqType := Eval hnf in EqType intpart intpart_eqMixin.
Definition intpart_choiceMixin := Eval hnf in [choiceMixin of intpart by <:].
Canonical intpart_choiceType := Eval hnf in ChoiceType intpart intpart_choiceMixin.
Definition intpart_countMixin := Eval hnf in [countMixin of intpart by <:].
Canonical intpart_countType := Eval hnf in CountType intpart intpart_countMixin.

Lemma intpartP (p : intpart) : is_part p.

Hint Resolve intpartP.

Canonical conj_intpart p := IntPart (is_part_conj (intpartP p)).

Lemma conj_intpartK : involutive conj_intpart.

Lemma intpart_sum_inj (s t : intpart) :
  (∀ k, part_sum s k = part_sum t k) → s = t.

Fixpoint enum_partnsk sm sz mx : (seq (seq nat)) :=
  if sz is sz.+1 then
    flatten [seq [seq i :: p | p <- enum_partnsk (sm - i) sz i] | i <- iota 1 (minn sm mx)]
  else if sm is sm.+1 then [::] else [:: [::]].
Definition enum_partns sm sz := enum_partnsk sm sz sm.
Definition enum_partn sm := flatten [seq enum_partns sm sz | sz <- iota 0 sm.+1 ].
```

Major usability concerns (?)

- Difficult and work-intensive to **install/compile from source** on some systems
- As both a **proof assistant** *and* a **programming language**, understanding the theory behind Coq and acquiring a working knowledge of the semantics/technical peculiarities of the Coq system is quite work-intensive
- Finding and analyzing proofs is mostly a **manual** (if assisted) **process** — **standardization** and **searchability**?
- **Curation, quality control** and medium- to long-term **maintenance** of collections of proofs is challenging
- **“Burden of interdisciplinarity”** — documenting a given piece of mathematical knowledge in a wiki system requires substantial amounts of **human-readable** and potentially highly technical text, potentially with very involved **mathematical examples** for illustration, while designing corresponding proofs in Coq requires a form of programming which has to be aided by some form of **Coq API** documentation and ideally a library of **code examples**
- ...

Major usability concerns (?)

- Difficult and work-intensive to **install/compile from source** on some systems
- As both a **proof assistant** *and* a **programming language**, understanding the theory behind Coq and acquiring a working knowledge of the semantics/technical peculiarities of the Coq system is quite work-intensive
- Finding and analyzing proofs is mostly a **manual** (if assisted) **process** — **standardization** and **searchability**?
- **Curation, quality control** and medium- to long-term **maintenance** of collections of proofs is challenging
- **“Burden of interdisciplinarity”** — documenting a given piece of mathematical knowledge in a wiki system requires substantial amounts of **human-readable** and potentially highly technical text, potentially with very involved **mathematical examples** for illustration, while designing corresponding proofs in Coq requires a form of programming which has to be aided by some form of **Coq API** documentation and ideally a library of **code examples**
- ...



<https://youtu.be/4084o1hk1Qs>

Welcome to the jsCoq Interactive Online System!

Welcome to the jsCoq technology demo! jsCoq is an interactive, web-based environment for the Coq Theorem prover, and is a collaborative development effort. See the [list of contributors](#) below.

jsCoq is open source. If you find any problem or want to make any contribution, you are extremely welcome! We await your feedback at [GitHub](#) and [Zulip](#).

Instructions:

The following document contains embedded Coq code. All the code is editable and can be run directly on the page. Once jsCoq finishes loading, you are free to experiment by stepping through the proof and viewing intermediate proof states on the right panel.

Actions:

Button	Key binding	Action
	Alt + ↑ / ↓ or Alt + N / P	Move through the proof.
	Alt + Enter or Alt + -	Run (or go back) to the current point.
	F8	Toggles the goal panel.

Creating your own proof scripts:

The *scratchpad* offers simple, local storage functionality. It also allows you to share your development with other users in a manner that is similar to Pastebin.

A First Example: The Infinitude of Primes

If you are new to Coq, check out this [introductory tutorial](#) by Mike Nahas. As a more advanced showcase, we display a proof of the infinitude of primes in Coq. The proof relies on the [Mathematical Components](#) library from the MSR/Inria team led by Georges Gonthier, so our first step will be to load it:

```
1 From Coq Require Import ssreflect ssrfun ssrbool.
2 From mathcomp Require Import eqtype ssrnat div prime.
```

Ready to do Proofs!

Once the basic environment has been set up, we can proceed to the proof:

```
3 (* A nice proof of the infinitude of primes, by Georges Gonthier *)
4 Lemma prime_above m : {p | m < p & prime p}.
5 Proof.
```

Goals

```
jsCoq (0.13.3), Coq 8.13.2/81300 (September 2021),
compiled on Sep 21 2021 15:32:50
OCaml 4.12.0, Js_of_ocaml 3.9.0
```

Coq worker is ready.
====> Loaded packages [init]

Messages

```
Coq.Init.Datatypes loaded.
Coq.Init.Logic_Type loaded.
Coq.Init.Specif loaded.
Coq.Init.Decimal loaded.
Coq.Init.Hexadecimal loaded.
Coq.Init.Number loaded.
Coq.Init.Nat loaded.
Coq.Init.Byte loaded.
Coq.Init.Numeral loaded.
Coq.Init.Peano loaded.
```

<https://github.com/jscoq>

```
9 Lemma gf_ensembles (n : ℕ) : gf ensembles n = gcard (BAut (Fin n)).
10 Proof.
11   unfold gf. path_via (gcard (BAut (Fin n)) * 1).
12   - f_ap. unfold hfiber. simpl. path_via (gcard Unit).
13     + apply gcard_equiv'. apply equiv_contr_unit.
14     + apply gcard_unit.
15   - apply mult_1_r.
16 Defined.
17
18 Lemma contr_stuff_spec (P : FinSet → Type) (HP : ∀ A, Contr (P A))
19   : spec_from_stuff P = ensembles.
20 Proof.
21   path_via (spec_from_stuff (λ _ => Unit)).
22   - apply path_stuff_spec. intro A.
23     apply equiv_contr_unit.
24   - apply path_sigma_uncurried. refine (λ _ => _).
25     + apply path_universe_uncurried.
26       refine (equiv_adjointify _ _ _).
27       * apply pr1.
28       * apply (λ A => (A; tt)).
29       * intro A. reflexivity.
30     * intro A. apply path_sigma_hprop. reflexivity.
31   + simpl. apply path_arrow. intro A.
32     refine ((transport_arrow _ _ _) @ _).
33     refine ((transport_const _ _ _) @ _).
34     path_via (transport idmap
35       (path_universe_uncurried
36         (equiv_inverse
37           (equiv_adjointify pr1 (λ A0 : FinSet => (A0; tt))
38             (λ A0 : FinSet => 1)
39             (λ A0 : { _ : FinSet & Unit} =>
40               path_sigma_hprop (let (proj1_sig, _) := A0 in proj1_sig; tt) A0
41               1))) A).1.
42     f_ap. f_ap. simpl. symmetry. apply path_universe_V_uncurried. simpl.
43     path_via ((equiv_inverse
44       (equiv_adjointify pr1 (λ A0 : FinSet => (A0; tt))
45         (λ A0 : FinSet => 1)
46         (λ A0 : { _ : FinSet & Unit} =>
47           path_sigma_hprop (let (proj1_sig, _) := A0 in proj1_sig; tt)
48           A0 1))) A).1.
49     f_ap. apply transport_path_universe_uncurried.
50 Defined.
51
52 Lemma gf_contr_stuff_spec (P : FinSet → Type) (HP : ∀ A, Contr (P A)) (n : ℕ)
53   : gf (spec_from_stuff P) n = gcard (BAut (Fin n)).
54 Proof.
55   refine (λ _ @ (gf_ensembles _)). f_ap.
56   apply contr_stuff_spec. apply HP.
57 Defined.
58
59 (** ** Decidable (-1)-stuff ** *)
60
```

Coq worker is ready.
====> Loaded packages [init]

Messages

```
Coq.Init.Datatypes loaded.
Coq.Init.Logic_Type loaded.
Coq.Init.Specif loaded.
Coq.Init.Decimal loaded.
Coq.Init.Hexadecimal loaded.
Coq.Init.Number loaded.
Coq.Init.Nat loaded.
Coq.Init.Byte loaded.
Coq.Init.Numeral loaded.
Coq.Init.Peano loaded.
Coq.Init.Wf loaded.
Coq.Init.Tactics loaded.
Coq.Init.Tauto loaded.
/lib/Coq/syntax/number_string_notation_plugin.cma loaded.
/lib/Coq/ltac/tauto_plugin.cma loaded.
/lib/Coq/cc/cc_plugin.cma loaded.
/lib/Coq/finorder/ground_plugin.cma loaded.
```

Core developer team

- Emilio Jesús Gallego Arias , Inria, Université de Paris, IRIF
- Shachar Itzhaky , Technion

Past Contributors

- Benoît Pin, CRI, MINES ParisTech

Welcome to the jsCoq Interactive

Welcome to the jsCoq technology demo! jsCoq is an interactive Coq Theorem prover, and is a collaborative development environment. jsCoq is open source. If you find any problem or want to contribute, we await your feedback at [GitHub](#) and [Zulip](#).

Instructions:

The following document contains embedded Coq code. You can click on the page. Once jsCoq finishes loading, you are free to interact with the Coq code, viewing intermediate proof states on the right panel.

Actions:

Button	Key binding	Action
↕	Alt+↑ or Alt+N/P	Move through the proof
↕	Alt+Enter or Alt+↵	Run (or go back) to the goal
⊞	F8	Toggles the goal panel

Creating your own proof scripts:

The scratchpad offers simple, local storage functionality. You can share your proof scripts with other users in a manner that is similar to Pastebin.

A First Example: The Infinitude of Primes

If you are new to Coq, check out this [introductory tutorial](#). In this demo, we display a proof of the infinitude of primes in Coq. The proof is taken from the library from the MSR/Inria team led by Georges Gonthier.

```
1 From Coq Require Import ssreflect
2 From mathcomp Require Import eqtype
```

Ready to do Proofs!

Once the basic environment has been set up, we can start proving.

```
3 (* A nice proof of the infinitude of primes *)
4 Lemma prime_above m : {p | m < p & prime p} = ∅.
5 Proof.
```

Core developers

- Emilio Jesuino
- Shachar

Past contributors

- Benoît P.

```
9 Lemma gf_ensembles (n : ℕ) : gf ensembles n = gcard (BAut (Fin n)).
10 Proof.
11   unfold gf. path_via (gcard (BAut (Fin n)) * 1).
12   - f_ap. unfold hfiber. simpl. path_via (gcard Unit).
13     + apply gcard_equiv'. apply equiv_contr_unit.
14     + apply gcard_unit.
15   - apply mult_1_r.
16 Defined.
17
18 Lemma contr_stuff_spec (P : FinSet → Type) (HP : ∀ A, Contr (P A))
19   : spec_from_stuff P = ensembles.
20 Proof.
21   path_via (spec_from_stuff (λ _ ⇒ Unit)).
22   - apply path_stuff_spec. intro A.
23     apply equiv_contr_unit.
24   - apply path_sigma_uncurried. refine (_; _).
25     + apply path_universe_uncurried.
26       refine (equiv_adjointify _ _ _).
27       * apply pr1.
28       * apply (λ A ⇒ (A; tt)).
29       * intro A. reflexivity.
30       * intro A. apply path_sigma_hprop. reflexivity.
31     + simpl. apply path_arrow. intro A.
32       refine ((transport_arrow _ _ _) @ _).
33       refine ((transport_const _ _) @ _).
34     path_via (transport idmap
35       (path_universe_uncurried
36         (equiv_inverse
37           (equiv_adjointify pr1 (λ A₀ : FinSet ⇒ (A₀; tt))
38             (λ A₀ : FinSet ⇒ 1)
39             (λ A₀ : { _ : FinSet & Unit } ⇒
40               path_sigma_hprop (let (proj1_sig, _) := A₀ in proj1_sig; tt) A₀
41                 1))) A).1.
42     f_ap. f_ap. simpl. symmetry. apply path_universe_V_uncurried. simpl.
43     path_via ((equiv_inverse
44       (equiv_adjointify pr1 (λ A₀ : FinSet ⇒ (A₀; tt))
45         (λ A₀ : FinSet ⇒ 1)
46         (λ A₀ : { _ : FinSet & Unit } ⇒
47           path_sigma_hprop (let (proj1_sig, _) := A₀ in proj1_sig; tt)
48             A₀ 1))) A).1.
49     f_ap. apply transport_path_universe_uncurried.
50 Defined.
51
52 Lemma gf_contr_stuff_spec (P : FinSet → Type) (HP : ∀ A, Contr (P A)) (n : ℕ)
53   : gf (spec_from_stuff P) n = gcard (BAut (Fin n)).
54 Proof.
55   refine (_ @ (gf_ensembles _)). f_ap.
56   apply contr_stuff_spec. apply HP.
57 Defined.
58
59 (** ** Decidable (-1)-stuff ** *)
60
```



Coq worker is ready.

==> Loaded packages [init]

Messages

```
Coq.Init.Datatypes loaded.
Coq.Init.Logic_Type loaded.
Coq.Init.Specif loaded.
Coq.Init.Decimal loaded.
Coq.Init.Hexadecimal loaded.
Coq.Init.Number loaded.
Coq.Init.Nat loaded.
Coq.Init.Byte loaded.
Coq.Init.Numeral loaded.
Coq.Init.Peano loaded.
Coq.Init.Wf loaded.
Coq.Init.Tactics loaded.
Coq.Init.Tauto loaded.
/lib/Coq/syntax/number_string_notation_plugin.cma loaded.
/lib/Coq/ltac/tauto_plugin.cma loaded.
/lib/Coq/cc/cc_plugin.cma loaded.
/lib/Coq/firstorder/ground_plugin.cma loaded.
```

Towards **automated** theorem-proving and **tactics-learning**

CoqHammer
Proof Automation for Dependent Type Theory

View on GitHub

CoqHammer is an automated reasoning tool for Coq. It helps in your search for Coq proofs.

Since version 1.3, the CoqHammer system consists of two major separate components.

1. The `sauto` general proof search tactic for the Calculus of Inductive Construction.

A "super" version of `sauto`. The underlying proof search procedure is based on a fairly general inhabitation procedure for the Calculus of Inductive Constructions. While it is in theory complete only for a "first-order" fragment of CIC, in practice it can handle a much larger part of Coq's logic. In contrast

<https://coqhammer.github.io>

The Tactician

A Seamless, Interactive Tactic Learner and Prover for Coq

Online Demo Install Now

<https://coq-tactician.github.io>

SMTCoq

Communication between Coq and SAT/SMT solvers

SMTCoq Presentation

SMTCoq is a `Coq` plugin that checks proof witnesses coming from external SAT and SMT solvers. It provides:

- a certified checker for proof witnesses coming from the SAT solver `ZChaff` and the SMT solvers `veriT` and `CVC4`. This checker increases the confidence in these tools by checking their answers a posteriori and allows to import new theorems proved by these solvers in Coq;
- decision procedures through new tactics that discharge some Coq goals to `ZChaff`, `veriT`, `CVC4`, and their combination.

<https://smtcoq.github.io>

math-comp / hierarchy-builder Public

Notifications Star 55 Fork 8

<> Code Issues 24 Pull requests 13 Actions Projects Wiki Security

master 34 branches 10 tags Go to file Code

CohenCyril Merge pull request #2... 2fa17b7 21 days ago 915 commits

- .github/workflows complete CI mathcomp on HB 21 days ago
- .nix complete CI mathcomp on HB 21 days ago
- .vscode Introducing FactoryName.sort 7 months ago
- HB #[hnf] 3 months ago
- build-support/coq Duplication from nixpkgs 8 months ago
- examples optim successfull 4 months ago

About

High level commands to declare a hierarchy based on packed classes

coq mathcomp elpi

Readme MIT License

Releases 6

Hierarchy Builder 1.2.0 Latest on 24 Sep

<https://github.com/math-comp/hierarchy-builder>

Coq-community

The screenshot shows the GitHub profile for 'coq-community'. At the top, there's a navigation bar with links for 'Why GitHub?', 'Team', 'Enterprise', 'Explore', 'Marketplace', and 'Pricing'. A search bar and 'Sign in'/'Sign up' buttons are also present. The profile header includes the organization's name 'coq-community', a description: 'A project for a collaborative, community-driven effort for the long-term maintenance and advertisement of Coq packages.', and the website 'https://coq-community.org'. Below this, there are tabs for 'Overview', 'Repositories (58)', 'Packages', 'People (14)', and 'Projects'. The 'Pinned' section displays six repositories: 'manifesto', 'hydra-battles', 'awesome-coq', 'vscoq', 'docker-coq', and 'templates'. Each repository card shows its name, public status, a brief description, and statistics like stars and forks. To the right, there's a 'People' section with avatars of maintainers, a 'Top languages' section listing Coq, Shell, OCaml, Dockerfile, and JavaScript, and a 'Most used topics' section with tags like 'coq', 'docker-coq-action', 'mathcomp', 'nix-action', and 'coq-library'.

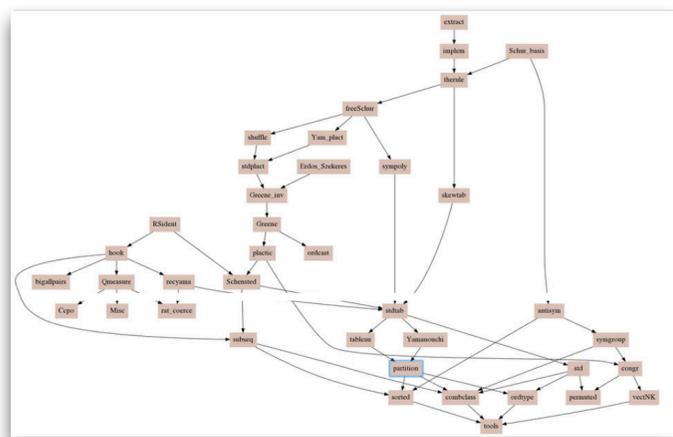
<https://github.com/coq-community>

“A project for a **collaborative, community-driven** effort for the **long-term maintenance** and **advertisement** of Coq packages.”

- **58 repositories**

The screenshot shows the GitHub repository page for 'Mathematical Components'. The navigation bar includes 'Overview', 'Repositories', 'Packages', 'People', and 'Projects'. A search bar and filters for 'Type', 'Language', and 'Sort' are at the top. The repository list shows several entries: 'Abel' (A proof of Abel-Ruffini theorem), 'algebra-tactics' (Ring and field tactics for Mathematical Components), 'analysis' (Mathematical Components compliant Analysis Library), 'apery' (A formal proof of the irrationality of zeta(3), the Apéry constant), 'bigenough' (Asymptotic reasoning with bigenough), 'cad' (Formalizing Cylindrical Algebraic Decomposition related theories in mathcomp), and 'Coq-Combi' (Algebraic Combinatorics in Coq). Each entry displays its name, public status, a brief description, topic tags, and statistics like stars, forks, and issues.

The **coreact.wiki** proposal



knowledge graphs



(graph-) **database**

wiki entry (e.g., a Lemma)

#hash (auto-generated)

tags

list of cross-references

bibliographic references

code origin references

Human-readable text

LaTeX-based, with annotations permitting generation of cross-references via NNexus

Machine-readable Coq-formalization

Including compatible Coq version and possibly different variants for (1) different Coq versions and/or (2) different implementation strategies/frameworks/theories.

Examples (both maths & Coq)

Curated in j sCoq, directly executable from within the wiki entry in the form of a literate web document and/or as a bundle of a Coq file with instructions for a particular Docker image for Coq.

Proof tactics and performance data

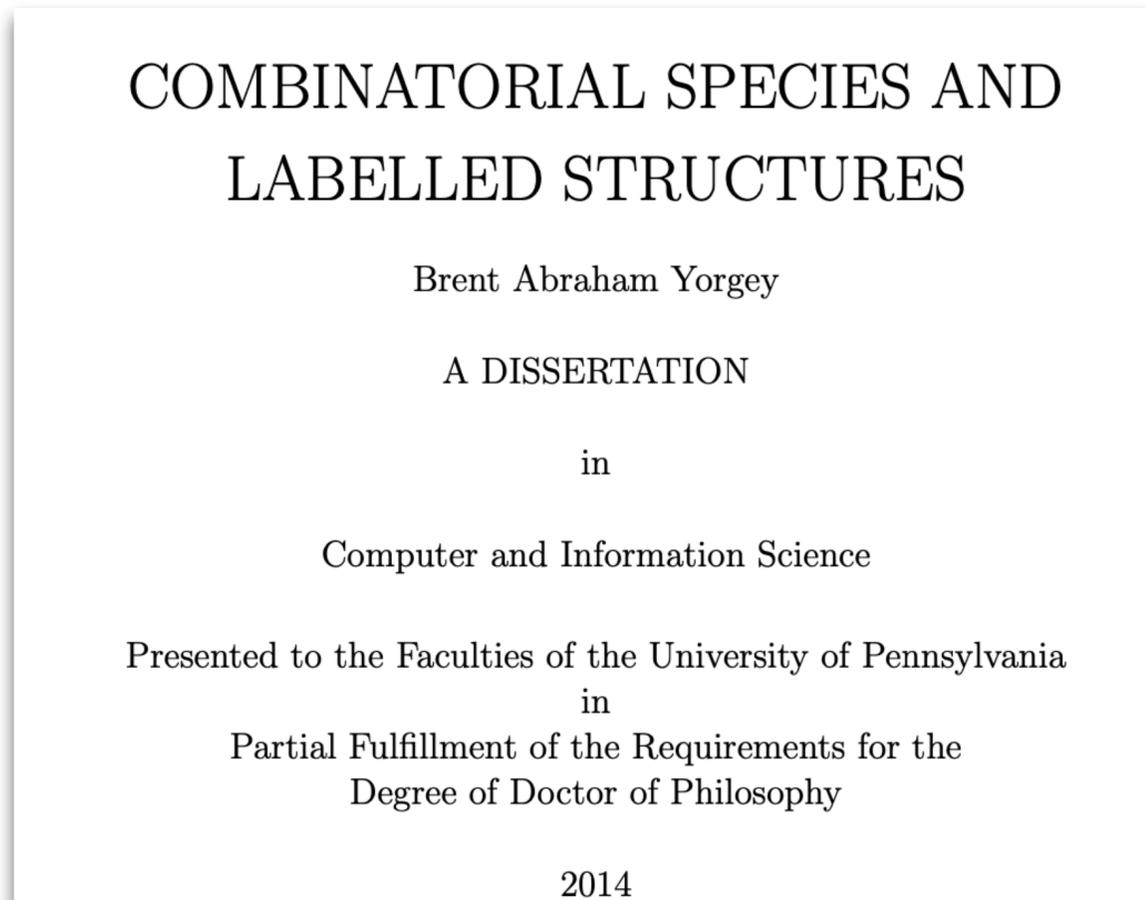
Machine-learned tactics data, cross-evaluation of performance of different variants of implementations, user annotations on different Coq versions/libraries used

Maintenance: collaborate with/
adopt standards of **Coq-community**



PDF & HTML **textbook**

A proposal for a **wiki-topic case study: HoTT Species**



<https://repository.upenn.edu/edissertations/1512/>

The screenshot shows the GitHub interface for the repository 'jldoughertyii/hott-species'. The repository is public and has 11 stars and 0 forks. The main content area shows a commit history table with columns for file name, commit message, and time ago. The files listed are 'coq', '.gitignore', 'LICENSE', 'README.md', 'references.bib', 'species.pdf', and 'species.tex'. The 'README.md' file is selected, showing its content which includes the title 'hott-species' and the subtitle 'Combinatorial species in HoTT'. An abstract is also visible, discussing combinatorial species and their connection to homotopy type theory (HoTT).

File	Commit Message	Time Ago
coq	correcting the coproduct	7 years ago
.gitignore	adding notes	7 years ago
LICENSE	Initial commit	7 years ago
README.md	Initial commit	7 years ago
references.bib	initial commit	7 years ago
species.pdf	mixed up ogfs and egfs wrt labeling	7 years ago
species.tex	mixed up ogfs and egfs wrt labeling	7 years ago

Species in HoTT

John Dougherty
May 23, 2015

Abstract

Combinatorial species were developed by Joyal (1981) as an abstract treatment of enumerative combinatorics, especially problems of counting the number of ways of putting some structure on a finite set. Many of the results of species theory are special cases of more general properties of homotopy types, making homotopy type theory (HoTT) a useful tool for dealing with species. These tools become even more apposite when one generalizes species to higher groupoids, as Baez and Dolan (2001) do. What follows are notes I wrote while learning about species. They're mainly summary of the notes Derek Wise took during John Baez's "Quantization and Categorification" seminar in AY2004 (Baez and Wise, 2003, 2004b,a), with some reference to Bergeron et al. (2013), Baez and Dolan (2001), and Aguiar and Mahajan (2010).

<https://github.com/jldoughertyii/hott-species>

A proposal for a **wiki-topic case study: HoTT Species**

Defining species	1
Computing cardinalities	2
▼ Speciation	5
Coproduct	5
Hadamard product	5
Cauchy product	6
Composition	7
Differentiation	8
Pointing	9
Inhabiting	9
▼ Examples	10
(-2)-stuff	10
(-1)-stuff	10
0-stuff	11
Fock space	13
Cayley's Formula	13
References	13

If F is a...	then $ F (z) = \sum \frac{a_n}{n!} z^n$ where:	since these numbers are cardinalities of:
stuff type	$a_n \in \mathbb{R}^+ = [0, \infty)$	(tame) groupoids = 1-groupoids
structure type	$a_n \in \mathbb{N}$	(finite) sets = 0-groupoids
property type	$a_n \in \{0, 1\} \cong \{F, T\}$	truth values = -1-groupoids
vacuous property type	$a_n \in \{1\} \cong \{T\}$	true = <u>the only</u> -2-groupoid

John Baez, Quantum Gravity Seminar — Spring 2004: Quantization and Categorification, *Week 3 - Evaluating and composing stuff types* (notes by *D. Wise*)

<https://math.ucr.edu/home/baez/qg-spring2004/s04week03.pdf>

A proposal for a **wiki-topic case study: HoTT Species**

Defining species	1
Computing cardinalities	2
▼ Speciation	5
Coproduct	5
Hadamard product	5
Cauchy product	6
Composition	7
Differentiation	8
Pointing	9
Inhabiting	9
▼ Examples	10
(-2)-stuff	10
(-1)-stuff	10
0-stuff	11
Fock space	13
Cayley's Formula	13
References	13

If F is a...	then $ F (z) = \sum \frac{a_n}{n!} z^n$ where:	since these numbers are cardinalities of:
stuff type	$a_n \in \mathbb{R}^+ = [0, \infty)$	(tame) groupoids = 1-groupoids
structure type	$a_n \in \mathbb{N}$	(finite) sets = 0-groupoids
property type	$a_n \in \{0, 1\} \cong \{F, T\}$	truth values = -1-groupoids
vacuous property type	$a_n \in \{1\} \cong \{T\}$	true = <u>the only</u> -2-groupoid

John Baez, Quantum Gravity Seminar — Spring 2004: Quantization and Categorification, *Week 3 - Evaluating and composing stuff types* (notes by *D. Wise*)

<https://math.ucr.edu/home/baez/qg-spring2004/s04week03.pdf>

A proposal for a **wiki-topic case study: HoTT Species**

Defining species	1
Computing cardinalities	2
▼ Speciation	5
Coproduct	5
Hadamard product	5
Cauchy product	6
Composition	7
Differentiation	8
Pointing	9
Inhabiting	9
▼ Examples	10
(-2)-stuff	10
(-1)-stuff	10
0-stuff	11
Fock space	13
Cayley's Formula	13
References	13

If F is a...	then $ F (z) = \sum \frac{a_n}{n!} z^n$ where:	since these numbers are cardinalities of:
stuff type	$a_n \in \mathbb{R}^+ = [0, \infty)$	(tame) groupoids = 1-groupoids
structure type	$a_n \in \mathbb{N}$	(finite) sets = 0-groupoids
property type	$a_n \in \{0, 1\} \cong \{F, T\}$	truth values = -1-groupoids
vacuous property type	$a_n \in \{1\} \cong \{T\}$	true = <u>the only</u> -2-groupoid

John Baez, Quantum Gravity Seminar — Spring 2004: Quantization and Categorification, *Week 3 - Evaluating and composing stuff types* (notes by *D. Wise*)

<https://math.ucr.edu/home/baez/qg-spring2004/s04week03.pdf>

GRETA - Graph Transformation Theory and Applications

Université de Paris
INSTITUT DE RECHERCHE EN INFORMATIQUE FONDAMENTALE

GRETA *ExACT*

International Online Workgroup on
Executable Applied Category Theory
for Rewriting Systems

hosted at

The central aims of this workgroup consist in providing an interdisciplinary forum for exploring the diverse aspects of applied category theory relevant in graph transformation systems and their generalizations, in developing a methodology for formalizing diagrammatic proofs as relevant in rewriting theories via proof assistants such as Coq, and in establishing a community-driven wiki system and repository for mathematical knowledge in our research field (akin to a domain-specific Coq-enabled variant of the nLab). A further research question will explore the possibility of deriving reference prototype implementations of concrete rewriting systems (e.g., over multi- or simple directed graphs) directly from the category-theoretical semantics, in the spirit of the translation-based approaches (utilizing theorem provers such as Microsoft Z3).

- To receive regular updates on the GRETA ExACT workgroup sessions, please consider subscribing to our [mailing list](#).
- To suggest speakers and topics for upcoming sessions, and for any other form of feedback and discussions, please consider joining the GRETA ExACT [Mattermost channel](#).

YouTube FR

Search

SIGN IN

Home

Explore

Subscriptions

Library

History

Friday November 12, 2021, 10:00 CET

SMTCoq: the power of SMT solving in Coq

Chantal Keller

PLAY ALL

GRETA-ExACT (Executable Applied Category Theory) online workgroup

4 videos • No views • Last updated on 8 Nov 2021

GRETA Seminar

SUBSCRIBE

1 GRETA-ExACT session #1: "SMTCoq: the power of SMT solving in Coq" GRETA Seminar 55:36

2 GRETA-ExACT session #2: "Hierarchy Builder" GRETA Seminar 1:05:41

3 GRETA-ExACT session #3: "Formalizing Category Theory using Type Theory: A Discussion" GRETA Seminar • Scheduled for 10/12/2021, 15:00 LIVE

4 GRETA seminar #20: "GRETA-ExACT: towards Executable Applied Category Theory" GRETA Seminar 1:46:40

coreact.wiki



```
83
84
85 (** * Species Sum/Coproduct *)
86
87 Definition spec_sum (X Y : Species) : Species
88   := ((X.1 + Y.1)%type; sum_rect _ X.2 Y.2).
89
90 Lemma sigma_functor_sum (X : Type) (P Q : X -> Type) :
91   {x : X & P x} + {x : X & Q x} <-> {x : X & (P x + Q x)%type}.
92 Proof.
93   refine (equiv_adjointify _ _ _).
94   - intros [x w] | [x w]]; exists x; [left | right]; apply w.
95   - intros [x [w | w]]; [left | right]; apply (x; w).
96   - intros [x [w | w]]; reflexivity.
97   - intros [[x w] | [x w]]; reflexivity.
98 Defined.
99
100 Definition stuff_spec_sum (P Q : FinSet -> Type) := fun A => (P A + Q A)%type.
101
102 Lemma stuff_spec_sum_correct (P Q : FinSet -> Type) :
103   spec_from_stuff (stuff_spec_sum P Q)
104   =
105   spec_sum (spec_from_stuff P) (spec_from_stuff Q).
106 Proof.
107   apply path_sigma_uncurried. refine (.; _).
108   - unfold stuff_spec_sum. simpl. symmetry.
109   apply path_universe_uncurried.
110   apply sigma_functor_sum.
111   - simpl. apply path_arrow. intros x.
112   refine ((transport_arrow _ _ _) @ _).
113   refine ((transport_const _ _) @ _).
114   path_via (transport idmap
115             (path_universe_uncurried (sigma_functor_sum FinSet P Q)
116             x).1.
117             f_ap. f_ap. apply inv_V.
118             nath via ((sigma functor sum FinSet P Q) x).1.
```

Mercci beaucoup !