# A PBPO$^{+}$ Graph Rewriting Tutorial

**Roy Overbeek** & Jörg Endrullis

1 August 2022 @ TERMGRAPH

Vrije Universiteit Amsterdam, The Netherlands

# Introduction

Last year, we proposed the algebraic graph rewriting formalism PBPO$^+$:

**Overbeek, R., Endrullis, J., and Rosset, A. (2021).** Graph rewriting and relabeling with PBPO$^+$.
In *Proc. Conf. on Graph Transformation (ICGT21)*, LNCS

which is a modification of PBPO:

**Corradini, A., Duval, D., Echahed, R., Prost, F., and Ribeiro, L. (2017).** The pullback-pushout approach to algebraic graph transformation.
In *Proc. Conf. on Graph Transformation (ICGT17)*, LNCS

Multiple tutorials exist for DPO and SPO, but none for PBPO$^+$ or related algebraic formalisms (PBPO, AGREE).

## Didactic Approach

We will introduce two toy formalisms:

- ToyPushout (ToyPO)
- ToyPullback (ToyPB)

And we will see how they combine into PBPO$^+$.

# Didactic Approach

We will introduce two toy formalisms:

- ToyPushout (ToyPO)
- ToyPullback (ToyPB)

And we will see how they combine into PBPO$^+$.

**Definition (Graph)**

A **graph** $G = (V, E, s, t)$ consists of a set of **vertices** $V$, a set **edges** $E$, a **source function** $s : E \to V$ and a **target function** $t : E \to V$.

A **graph homomorphism** $G \to G'$ consists of functions

- $\phi_V : V_G \to V_{G'}$
- $\phi_E : E_G \to E_{G'}$

such that

- $s_{G'} \circ \phi_E \; = \; \phi_V \circ s_G$
- $t_{G'} \circ \phi_E \; = \; \phi_V \circ t_G$

# ToyPO Rule and Match

We can interpret a graph homomorphism ρ as a graph rewrite rule:

$L =$


$= R$

$\approx$

“**identify** nodes *a* and *b*, and **add** a node *c*”

# ToyPO Rule and Match

We can interpret a graph homomorphism ρ as a graph rewrite rule:


≈

"**identify** nodes *a* and *b*, and **add** a node *c*"

**Definition (ToyPO Rule)**

A **ToyPO rule** is a morphism $\rho : L \to R$. $L$ and $R$ are called **patterns**.

# ToyPO Rule and Match

We can interpret a graph homomorphism $\rho$ as a graph rewrite rule:


$$\approx$$

“**identify** nodes $a$ and $b$, and **add** a node $c$”

**Definition (ToyPO Rule)**

A **ToyPO rule** is a morphism $\rho : L \to R$. $L$ and $R$ are called **patterns**.

Injective homomorphisms $m : L \rightarrowtail G$ model finding occurrences of $L$ in $G$:

# ToyPO Rule and Match

We can interpret a graph homomorphism ρ as a graph rewrite rule:

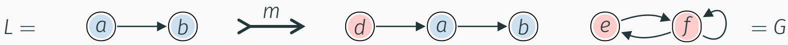

≈

"**identify** nodes *a* and *b*, and **add** a node *c*"

## Definition (ToyPO Rule)

A **ToyPO rule** is a morphism $\rho : L \to R$. $L$ and $R$ are called **patterns**.

Injective homomorphisms $m : L \rightarrowtail G$ model finding occurrences of $L$ in $G$:

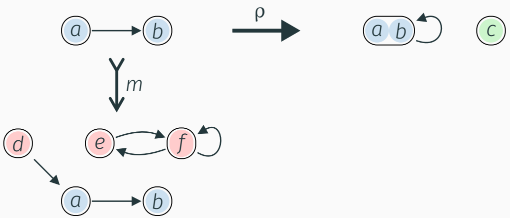



## Definition (ToyPO Match)

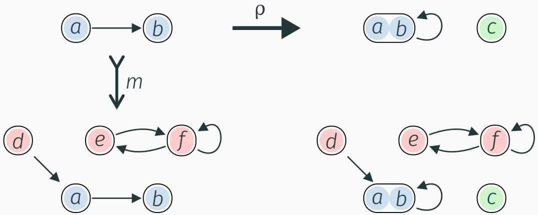
A **ToyPO match** for a rule $\rho : L \to R$ in $G$ is an injective morphism $m : L \rightarrowtail G$. Image $m(L)$ is an **occurrence** of $L$ in $G$.

# ToyPO Rewrite Step

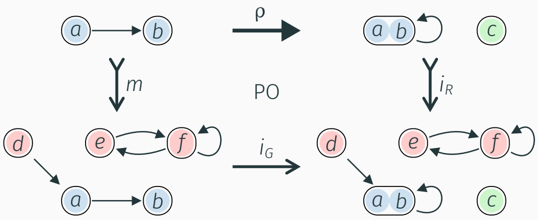
# ToyPO Rewrite Step



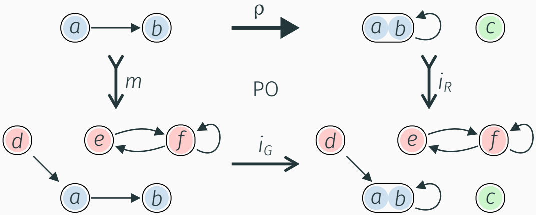
# ToyPO Rewrite Step



# ToyPO Rewrite Step



# ToyPO Rewrite Step

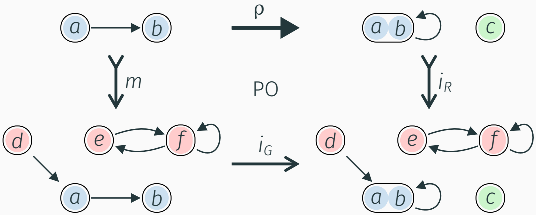



## Definition (Pushout)

The **pushout** of a **span** $G \xleftarrow{m} L \xrightarrow{\rho} R$

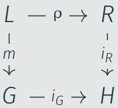
$$\begin{array}{ccc} L & \xrightarrow{\rho} & R \\ \downarrow{\scriptstyle m} & & \\ G & & \end{array}$$

## ToyPO Rewrite Step

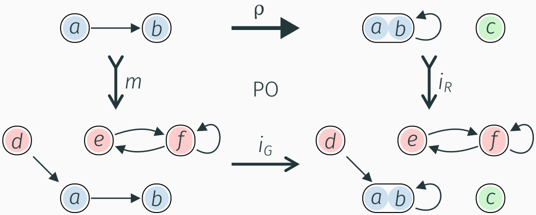



### Definition (Pushout)

The **pushout** of a **span** $G \xleftarrow{m} L \xrightarrow{\rho} R$
is a **cospan** $\sigma = G \xrightarrow{i_G} H \xleftarrow{i_R} R$



## ToyPO Rewrite Step



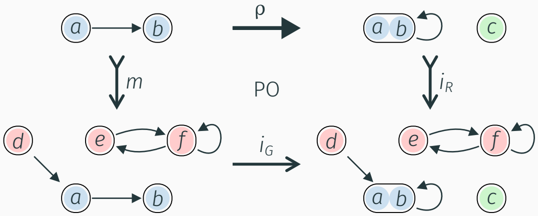

### Definition (Pushout)

The **pushout** of a **span** $G \overset{m}{\leftarrow} L \overset{\rho}{\rightarrow} R$
is a **cospan** $\sigma = G \overset{i_G}{\rightarrow} H \overset{i_R}{\leftarrow} R$ such that

1. **σ is a candidate solution**: $i_G \circ m = i_R \circ \rho$;

$$\begin{array}{ccc} L & \xrightarrow{\rho} & R \\ {\scriptstyle m}\downarrow & = & \downarrow{\scriptstyle i_R} \\ G & \xrightarrow{i_G} & H \end{array}$$

# ToyPO Rewrite Step

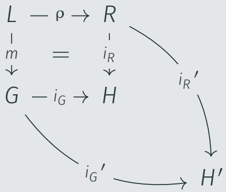



## Definition (Pushout)

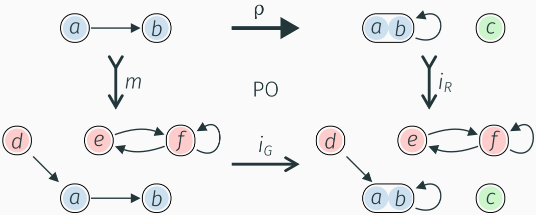
The **pushout** of a **span** $G \overset{m}{\leftarrow} L \overset{\rho}{\rightarrow} R$

is a **cospan** $\sigma = G \overset{i_G}{\rightarrow} H \overset{i_R}{\leftarrow} R$ such that

1. **$\sigma$ is a candidate solution**: $i_G \circ m = i_R \circ \rho$;
2. **$\sigma$ is the minimal solution**: for any cospan $G \overset{i_G'}{\rightarrow} H' \overset{i_R'}{\leftarrow} R$ that satisfies ${i_G}' \circ m = {i_R}' \circ \rho$,



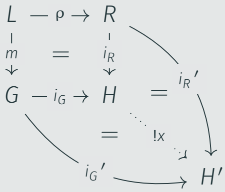
# ToyPO Rewrite Step



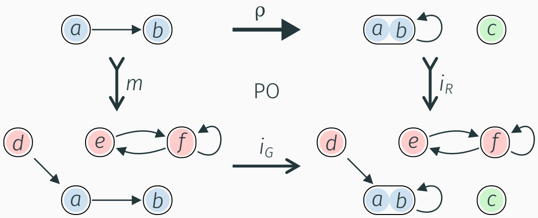

## Definition (Pushout)

The **pushout** of a **span** $G \xleftarrow{m} L \xrightarrow{\rho} R$
is a **cospan** $\sigma \ = \ G \xrightarrow{i_G} H \xleftarrow{i_R} R$ such that

1. **σ is a candidate solution**: $i_G \circ m = i_R \circ \rho$;
2. **σ is the minimal solution**: for any cospan $G \xrightarrow{i_G'} H' \xleftarrow{i_R'} R$ that satisfies $i_G{}' \circ m = i_R{}' \circ \rho$, there exists a **unique** $x : H \to H'$ with $i_G{}' = x \circ i_G$ and $i_R{}' = x \circ i_R$.



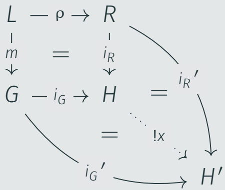
# ToyPO Rewrite Step




## Definition (Pushout)

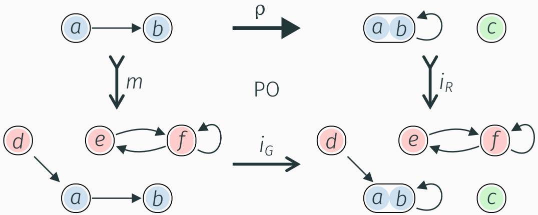
The **pushout** of a **span** $G \overset{m}{\leftarrow} L \overset{\rho}{\rightarrow} R$
is a **cospan** $\sigma \ = \ G \overset{i_G}{\rightarrow} H \overset{i_R}{\leftarrow} R$ such that

1. **$\sigma$ is a candidate solution**: $i_G \circ m = i_R \circ \rho$;
2. **$\sigma$ is the minimal solution**: for any cospan $G \overset{i_G'}{\rightarrow} H' \overset{i_R'}{\leftarrow} R$ that satisfies $i_G{}' \circ m = i_R{}' \circ \rho$, there exists a **unique** $x : H \rightarrow H'$ with $i_G{}' = x \circ i_G$ and $i_R{}' = x \circ i_R$.




Think of a pushout as a **gluing construction** or a **fibered union**.

# ToyPO Rewrite Step




## Definition (ToyPO Rewrite Step)

A rule $\rho : L \to R$ and match $m : L \rightarrowtail G$ induce a **ToyPO rewrite step** $G \Rightarrow^{\rho,m}_{\mathrm{ToyPO}} H$ if there exists a pushout of the form:

$$\begin{array}{ccc} L & \xrightarrow{\rho} & R \\ {\scriptstyle m}\big\downarrow & \mathrm{PO} & \big\downarrow{\scriptstyle i_R} \\ G & \xrightarrow{i_G} & H \end{array}$$

# Deleting and Duplicating

The pushout allows us to **identify** and **add** elements.

# Deleting and Duplicating

The pushout allows us to **identify** and **add** elements.

But we would also like to **delete** and **duplicate** elements.

# Deleting and Duplicating

The pushout allows us to **identify** and **add** elements.

But we would also like to **delete** and **duplicate** elements.

**First idea:** read a morphism from right to left:


$\approx$

"**duplicate** node *ab* (orienting the loop from *a* to *b*), and **delete** node *c*"

# Deleting and Duplicating

The pushout allows us to **identify** and **add** elements.

But we would also like to **delete** and **duplicate** elements.

**First idea:** read a morphism from right to left:


$\approx$

"**duplicate** node *ab* (orienting the loop from *a* to *b*), and **delete** node *c*"

## Definition (Pushout Complement)

A **pushout complement** for $G \overset{m}{\leftarrow} R \overset{\rho}{\leftarrow} L$

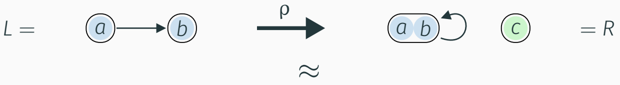
$$\begin{array}{ccc} R & \xleftarrow{\rho} & L \\ \downarrow{\scriptstyle m} & & \\ G & & \end{array}$$

## Deleting and Duplicating

The pushout allows us to **identify** and **add** elements.

But we would also like to **delete** and **duplicate** elements.

**First idea:** read a morphism from right to left:




"**duplicate** node *ab* (orienting the loop from *a* to *b*), and **delete** node *c*"

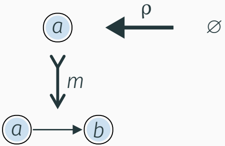
**Definition (Pushout Complement)**

A **pushout complement** for $G \xleftarrow{m} R \xleftarrow{\rho} L$
is a pair of morphisms $G \xleftarrow{l_2} H \xleftarrow{l_1} L$ such that we have:

$$\begin{array}{ccc} R & \xleftarrow{\ \rho\ } & L \\ {\scriptstyle m}\downarrow & \text{PO} & \downarrow{\scriptstyle l_1} \\ G & \xleftarrow{\ l_2\ } & H \end{array}$$

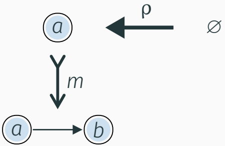
## Two Caveats

1. Pushout complements might not exist (*for this example: why not?*):



## Two Caveats

1. Pushout complements might not exist (*for this example: why not?*):

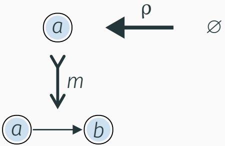



$\Longrightarrow$ Not **necessarily** a problem:

- For graphs, it blocks application when edges would be left dangling.

# Two Caveats

1. Pushout complements might not exist (*for this example: why not?*):

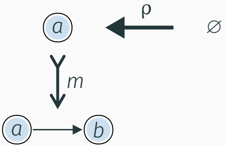



$\Longrightarrow$ Not **necessarily** a problem:

- For graphs, it blocks application when edges would be left dangling.
- **But**: we might prefer some other policy (highly domain-dependent).

# Two Caveats

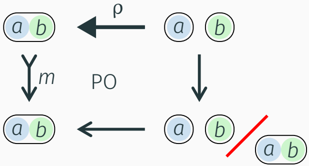
1. Pushout complements might not exist (*for this example: why not?*):




$\Longrightarrow$ Not **necessarily** a problem:

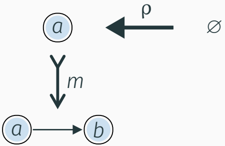
- For graphs, it blocks application when edges would be left dangling.
- **But**: we might prefer some other policy (highly domain-dependent).

2. Pushout complements are not always unique:



# Two Caveats

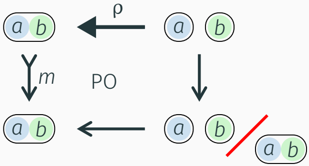
1. Pushout complements might not exist (*for this example: why not?*):




$\Longrightarrow$ Not **necessarily** a problem:

- For graphs, it blocks application when edges would be left dangling.
- **But**: we might prefer some other policy (highly domain-dependent).

2. Pushout complements are not always unique:

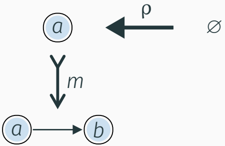



$\Longrightarrow$ **usually** a problem:

- nondeterminism & changes rule semantics

## Two Caveats

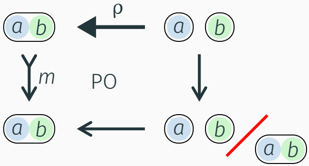
1. Pushout complements might not exist (*for this example: why not?*):




$\implies$ Not **necessarily** a problem:

- For graphs, it blocks application when edges would be left dangling.
- **But**: we might prefer some other policy (highly domain-dependent).

2. Pushout complements are not always unique:




$\implies$ **usually** a problem:

- nondeterminism & changes rule semantics
- difficult question: under what conditions are pushout complements unique?

# Frameworks in the Literature

## Definition (Double Pushout Rewriting [Ehrig et al., 1973])

A **DPO rewrite rule** ρ is a span $L \overset{l}{\leftarrowtail} K \overset{r}{\rightarrow} R$.

# Frameworks in the Literature

**Definition (Double Pushout Rewriting [Ehrig et al., 1973])**

A **DPO rewrite rule** $\rho$ is a span $L \overset{l}{\leftarrowtail} K \overset{r}{\to} R$. A diagram

$$\begin{array}{ccccc} L & \overset{l}{\leftarrowtail} & K & \overset{r}{\longrightarrow} & R \\ {\scriptstyle m}\downarrow & \text{PO} & \downarrow & \text{PO} & \downarrow \\ G_L & \longleftarrow & G_K & \longrightarrow & G_R \end{array}$$

defines a **DPO rewrite step** $G_L \Rightarrow^{\rho,m}_{\mathrm{DPO}} G_R$.

# Frameworks in the Literature

**Definition (Double Pushout Rewriting [Ehrig et al., 1973])**

A **DPO rewrite rule** $\rho$ is a span $L \overset{l}{\leftarrowtail} K \overset{r}{\rightarrow} R$. A diagram

$$\begin{array}{ccccc} L & \overset{l}{\leftarrowtail} & K & \overset{r}{\longrightarrow} & R \\ {\scriptstyle m}\downarrow & \text{PO} & \downarrow & \text{PO} & \downarrow \\ G_L & \longleftarrow & G_K & \longrightarrow & G_R \end{array}$$

defines a **DPO rewrite step** $G_L \Rightarrow^{\rho,m}_{\mathrm{DPO}} G_R$.

Injective $l$ ensures uniqueness of pushout complements in **Graph**, but:

- not in all categories; and
- we lose the ability to duplicate.

# Frameworks in the Literature

**Definition (Double Pushout Rewriting [Ehrig et al., 1973])**

A **DPO rewrite rule** $\rho$ is a span $L \overset{l}{\leftarrowtail} K \overset{r}{\rightarrow} R$. A diagram

$$\begin{array}{ccccc} L & \overset{l}{\leftarrowtail} & K & \overset{r}{\longrightarrow} & R \\ {\scriptstyle m}\downarrow & \text{PO} & \downarrow & \text{PO} & \downarrow \\ G_L & \longleftarrow & G_K & \longrightarrow & G_R \end{array}$$

defines a **DPO rewrite step** $G_L \Rightarrow^{\rho,m}_{\mathrm{DPO}} G_R$.

Injective $l$ ensures uniqueness of pushout complements in **Graph**, but:

- not in all categories; and
- we lose the ability to duplicate.

Alternative approaches:

- Single Pushout (SPO): partial morphisms, deletes dangling edges
- Sesqui Pushout (SqPO): final pullback complements, allows duplication
- AGREE: uses partial map classifiers, allows more control over duplication
- ...

# A Different Strategy: Dualizing ToyPO

$\implies$ Instead of a match $m : L \to G$, we will look for an $\alpha : G \to L'$.

## A Different Strategy: Dualizing ToyPO

$\Longrightarrow$ Instead of a match $m : L \to G$, we will look for an $\alpha : G \to L'$.

*Questions:*

1. *If*

$$L' = a \rightleftarrows b$$

*how can we describe those $G$ for which there exists an $\alpha : G \to L'$?*

# A Different Strategy: Dualizing ToyPO

$\Longrightarrow$ Instead of a match $m : L \to G$, we will look for an $\alpha : G \to L'$.

*Questions:*

1. *If*

$$L' = a \rightleftarrows b$$

*how can we describe those G for which there exists an $\alpha : G \to L'$?*
*Bipartite or 2-colorable, where $\alpha$ is a proof (assigns node colors).*

# A Different Strategy: Dualizing ToyPO

$\Longrightarrow$ Instead of a match $m : L \to G$, we will look for an $\alpha : G \to L'$.

*Questions:*

1. *If*

$$L' = a \rightleftarrows b$$

*how can we describe those $G$ for which there exists an $\alpha : G \to L'$?*

*Bipartite or 2-colorable, where $\alpha$ is a proof (assigns node colors).*

2. *If*

$$L' = \circlearrowleft x \circlearrowright \qquad ?$$

# A Different Strategy: Dualizing ToyPO

$\implies$ Instead of a match $m : L \to G$, we will look for an $\alpha : G \to L'$.

*Questions:*

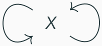
1. *If*

   $L' =$ 
   

   *how can we describe those G for which there exists an $\alpha : G \to L'$?*

   *Bipartite or 2-colorable, where $\alpha$ is a proof (assigns node colors).*

2. *If*

   $L' =$ 
   
   ?

   *Any graph, where an $\alpha$ assigns one of 2 edge "colors" to each edge.*

# A Different Strategy: Dualizing ToyPO

$\Longrightarrow$ Instead of a match $m : L \to G$, we will look for an $\alpha : G \to L'$.

*Questions:*

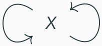
1. *If*

   $$L' = a \rightleftarrows b$$

   *how can we describe those G for which there exists an $\alpha : G \to L'$?*

   *Bipartite or 2-colorable, where $\alpha$ is a proof (assigns node colors).*

2. *If*

   $L' =$ 

?

   *Any graph, where an $\alpha$ assigns one of 2 edge "colors" to each edge.*

So we can now think of $L'$ as a type graph, and $\alpha$ a typing.
We will call $\alpha$ an **adherence**.

# A Different Strategy: Dualizing ToyPO

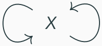
$\Longrightarrow$ Instead of a match $m : L \to G$, we will look for an $\alpha : G \to L'$.

*Questions:*

1. *If*

   $L' =$ 
   

   *how can we describe those $G$ for which there exists an $\alpha : G \to L'$?*
   *Bipartite or 2-colorable, where $\alpha$ is a proof (assigns node colors).*
2. *If*

   $L' =$ 
   
   ?

   *Any graph, where an $\alpha$ assigns one of 2 edge "colors" to each edge.*

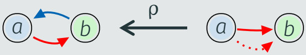
So we can now think of $L'$ as a type graph, and $\alpha$ a typing.
We will call $\alpha$ an **adherence**.

**Definition (ToyPB Rule)**

A **ToyPB rule** is a morphism $\rho : L' \leftarrow R'$. $L'$ and $R'$ are called **type graphs**.

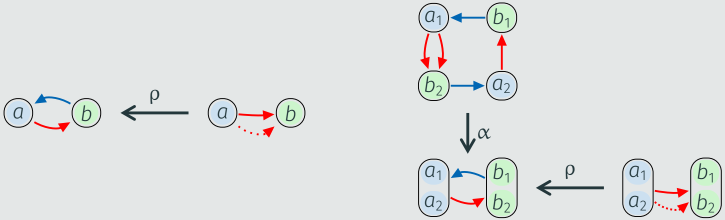
# Examples of Expected Behavior

## Example



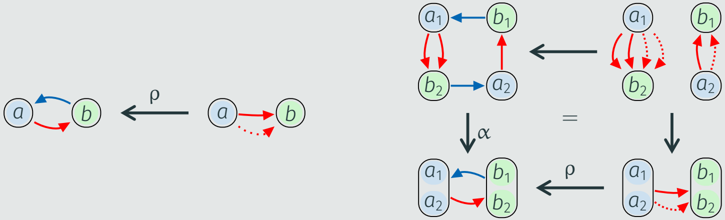
# Examples of Expected Behavior

## Example



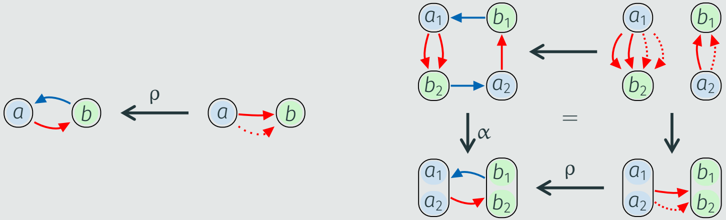
# Examples of Expected Behavior

## Example

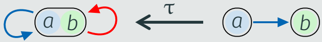


# Examples of Expected Behavior

## Example

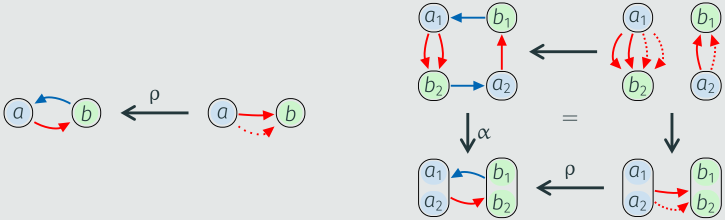



## Example

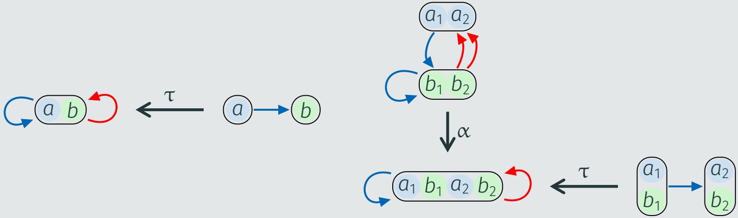


# Examples of Expected Behavior

## Example

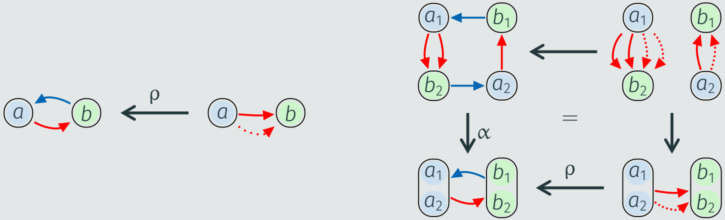



## Example

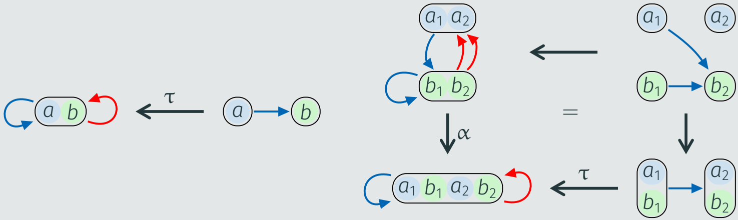


# Examples of Expected Behavior

## Example




## Example



# Pullbacks

The **dual** of a pushout is a pullback.

Pullbacks capture the expected behavior.

## Definition (Pullback)

The **pullback** of a cospan $G \xrightarrow{\alpha} L' \xleftarrow{\rho} R'$

$$\begin{array}{ccc} G & & \\ \downarrow{\alpha} & & \\ L' & \xleftarrow{\rho} & R' \end{array}$$

# Pullbacks

The **dual** of a pushout is a pullback.

Pullbacks capture the expected behavior.

## Definition (Pullback)

The **pullback** of a cospan $G \xrightarrow{\alpha} L' \xleftarrow{\rho} R'$ is a span $\sigma = G \xleftarrow{i_G} H \xrightarrow{i_R} R$

$$\begin{array}{ccc} G & \xleftarrow{i_G} & H \\ \downarrow \alpha & & \downarrow i_R \\ L' & \xleftarrow{\rho} & R' \end{array}$$

# Pullbacks

The **dual** of a pushout is a pullback.
Pullbacks capture the expected behavior.

## Definition (Pullback)

The **pullback** of a cospan $G \xrightarrow{\alpha} L' \xleftarrow{\rho} R'$ is a span $\sigma = G \xleftarrow{i_G} H \xrightarrow{i_R} R$ such that

1. $\sigma$ **is a candidate solution**: $\alpha \circ i_G = \rho \circ i_R$;

$$\begin{array}{ccc} G & \xleftarrow{i_G} & H \\ \downarrow \alpha & = & \downarrow i_R \\ L' & \xleftarrow{\rho} & R' \end{array}$$

# Pullbacks

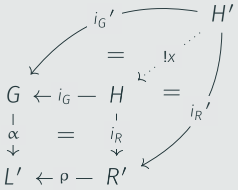
The **dual** of a pushout is a pullback.

Pullbacks capture the expected behavior.

## Definition (Pullback)

The **pullback** of a cospan $G \xrightarrow{\alpha} L' \xleftarrow{\rho} R'$ is a span $\sigma = G \xleftarrow{i_G} H \xrightarrow{i_R} R$ such that

1. **$\sigma$ is a candidate solution**: $\alpha \circ i_G = \rho \circ i_R$;
2. **$\sigma$ is the minimal solution**: for any span $G \xleftarrow{i_G{}'} H' \xrightarrow{i_R{}'} R'$ that satisfies $\alpha \circ i_G{}' = \rho \circ i_R{}'$, there exists a **unique** morphism $x : H' \to H$ such that $i_G{}' = i_G \circ x$ and $i_R{}' = i_R \circ x$.



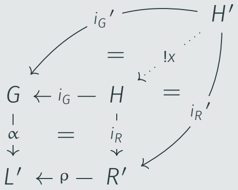
# Pullbacks

The **dual** of a pushout is a pullback.
Pullbacks capture the expected behavior.

**Definition (Pullback)**

The **pullback** of a cospan $G \xrightarrow{\alpha} L' \xleftarrow{\rho} R'$ is a span $\sigma = G \xleftarrow{i_G} H \xrightarrow{i_R} R$ such that

1. **$\sigma$ is a candidate solution**: $\alpha \circ i_G = \rho \circ i_R$;
2. **$\sigma$ is the minimal solution**: for any span $G \xleftarrow{i_G'} H' \xrightarrow{i_R'} R'$ that satisfies $\alpha \circ i_G' = \rho \circ i_R'$, there exists a **unique** morphism $x : H' \to H$ such that $i_G' = i_G \circ x$ and $i_R' = i_R \circ x$.




Think of a pullback as a **fibered product**:

$$H = \{(x, y) \in G \times R' \mid \alpha(x) = \rho(y)\}$$

# ToyPB

**Definition (ToyPB Rule)**

A **ToyPB rule** is a morphism $\rho : L' \leftarrow R'$. $L'$ and $R'$ are called **type graphs**.

**Definition (Adherence Morphism)**

An **adherence** for a ToyPB rule $\rho : L' \leftarrow R'$ is a morphism $\alpha : G \to L'$.

**Definition (ToyPB Rewrite Step)**

A ToyPB rule $\rho : L' \leftarrow R'$ and adherence morphism $\alpha : G \to L'$ induce a **ToyPB rewrite step** $G \Rightarrow^{\rho,\alpha}_{\mathrm{ToyPB}} H$ if there exists a pullback of the form

$$\begin{array}{ccc} G & \xleftarrow{i_G} & H \\ {\scriptstyle \alpha}\downarrow & \mathrm{PB} & \downarrow{\scriptstyle i_R} \\ L' & \xleftarrow{\rho} & R' \end{array}$$

# Combining ToyPB and ToyPO

Inverted ToyPO followed by ToyPO is easy to combine (giving DPO):

$$
\begin{array}{ccc}
L & \xleftarrow{\;l\;} & K \\
{\scriptstyle m}\downarrow & \text{PO} & \downarrow{\scriptstyle m'} \\
G & \longleftarrow & X
\end{array}
\qquad\qquad
\begin{array}{ccc}
K & \xrightarrow{\;r\;} & R \\
{\scriptstyle m'}\downarrow & \text{PO} & \downarrow \\
X & \longrightarrow & H
\end{array}
$$

# Combining ToyPB and ToyPO

Inverted ToyPO followed by ToyPO is easy to combine (giving DPO):

$$\begin{array}{ccccc} L & \xleftarrow{l} & K & \xrightarrow{r} & R \\ {\scriptstyle m}\downarrow & \text{PO} & \downarrow{\scriptstyle m'} & \text{PO} & \downarrow \\ G & \longleftarrow & X & \longrightarrow & H \end{array}$$

# Combining ToyPB and ToyPO

Inverted ToyPO followed by ToyPO is easy to combine (giving DPO):

$$
\begin{array}{ccccc}
L & \xleftarrow{\;l\;} & K & \xrightarrow{\;r\;} & R \\
\big\downarrow m & \text{PO} & \big\downarrow m' & \text{PO} & \big\downarrow \\
G & \longleftarrow & X & \longrightarrow & H
\end{array}
$$

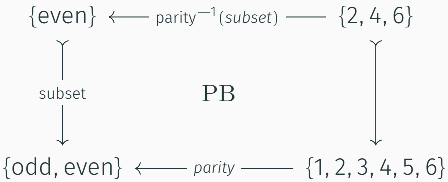
Combining ToyPB with ToyPO is less immediate because they work on different layers.

We need to:

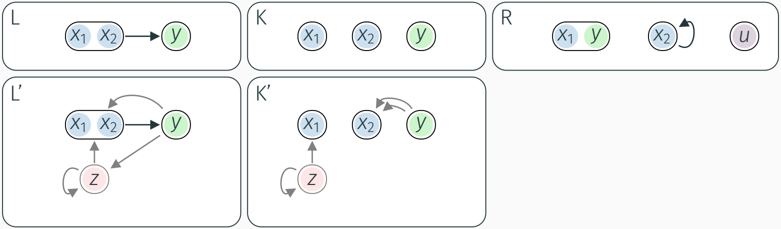
1. make matches and adherences play nice; and
2. find the right way to link a ToyPO step to a ToyPB step.

# Computing Preimages with Pullbacks

If one leg of a pullback is injective, pullbacks compute **preimages**:



# PBPO$^+$ Rewrite Rule



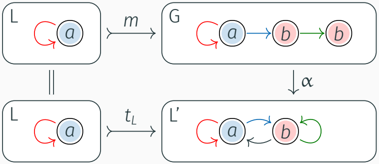

**Definition (PBPO$^+$ Rule [Corradini et al., 2019, Overbeek et al., 2021])**

A PBPO$^+$ **rewrite rule** $\rho$ is a diagram

$$\rho = \begin{array}{ccc} L & \xleftarrow{l} & K & \xrightarrow{r} & R \\ \downarrow t_L & \text{PB} & \downarrow t_K & & \\ L' & \xleftarrow{l'} & K' & & \end{array}$$

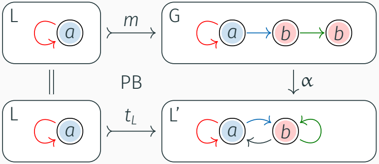
$L$ is the **lhs pattern** of the rule, $L'$ its **type graph**, and $t_L$ the **embedding** of $L$ into $L'$. $K$ is the **interface**. $R$ is the **rhs pattern** or **replacement for** $L$.

# Strong Match




For the step, we will find a match $m : L \rightarrowtail G$ and adherence $\alpha : G \to L'$. We want $\alpha$ to map **only** the occurrence $m(L)$ into the type graph embedding $t_L(L)$.

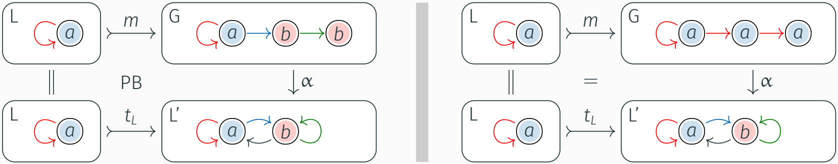
# Strong Match




For the step, we will find a match $m : L \rightarrowtail G$ and adherence $\alpha : G \to L'$. We want $\alpha$ to map **only** the occurrence $m(L)$ into the type graph embedding $t_L(L)$.

In other words, the preimage $\alpha^{-1}(t_L)$ must be $L$. We call this a **strong match**.

# Strong Match

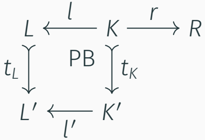


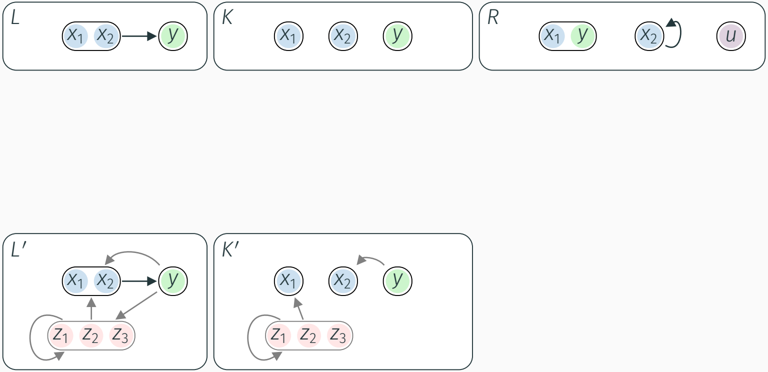

For the step, we will find a match $m : L \rightarrowtail G$ and adherence $\alpha : G \to L'$. We want $\alpha$ to map **only** the occurrence $m(L)$ into the type graph embedding $t_L(L)$.

In other words, the preimage $\alpha^{-1}(t_L)$ must be $L$. We call this a **strong match**.

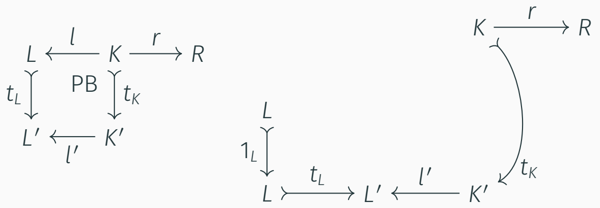
The right is a commuting square, but not a pullback.

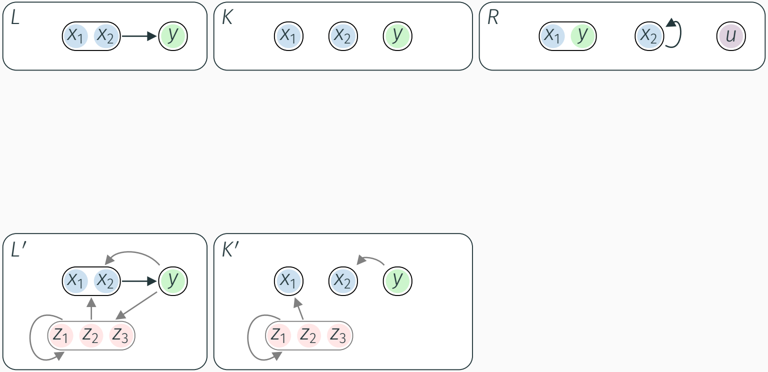
# Definition: PBPO+ Rewrite Step



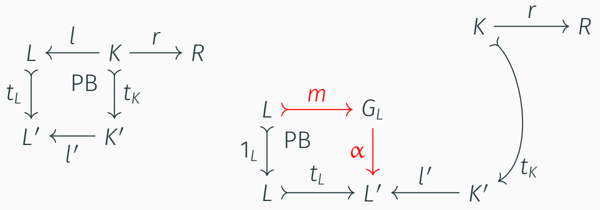



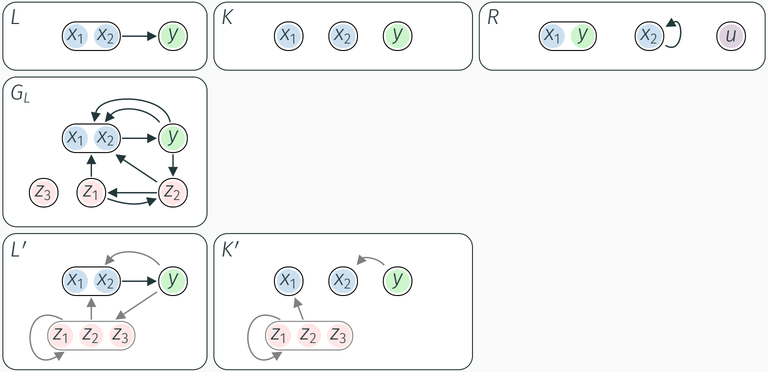
# Definition: PBPO$^+$ Rewrite Step



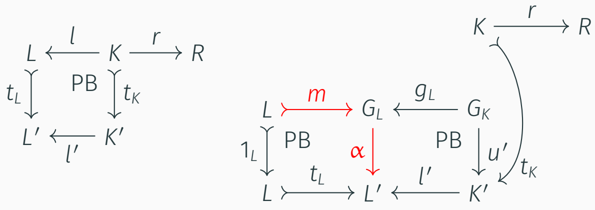



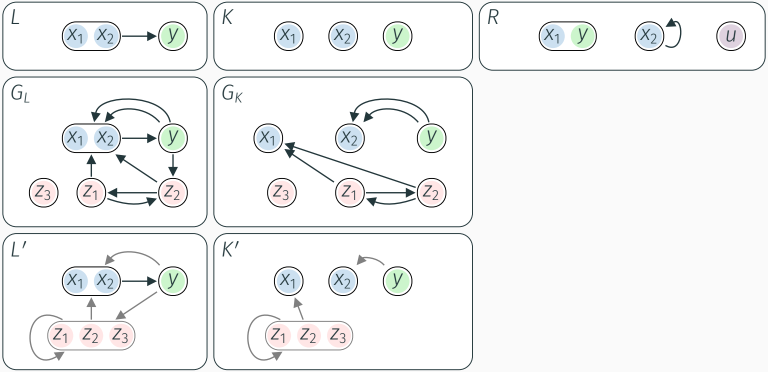
# Definition: $PBPO^+$ Rewrite Step



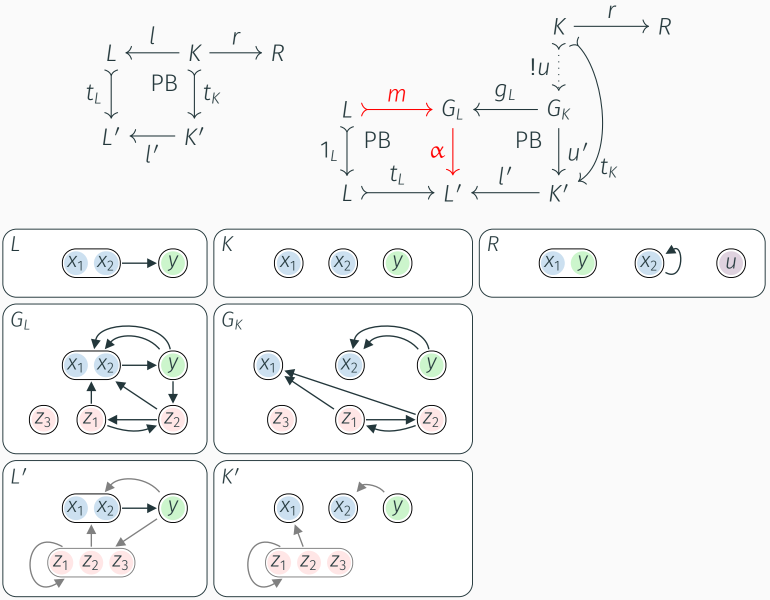



# Definition: PBPO+ Rewrite Step

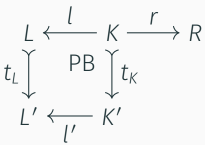


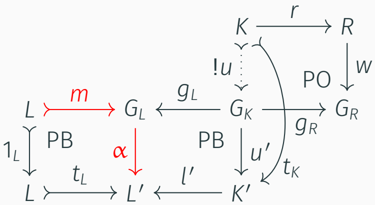



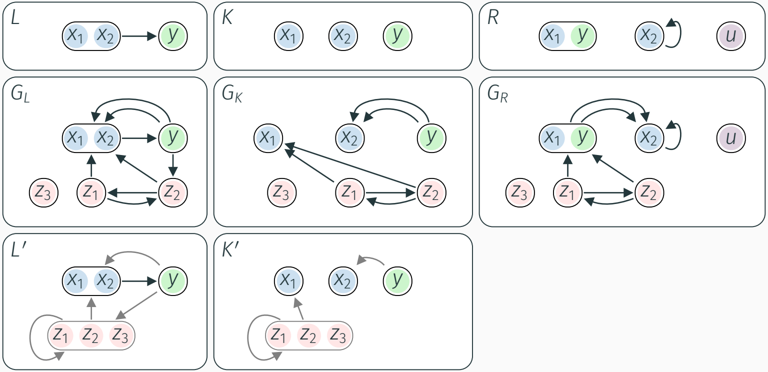
# Definition: PBPO+ Rewrite Step



# Definition: PBPO$^+$ Rewrite Step



# Closing Remarks

We intend to develop a tool for teaching.

Thank you!