

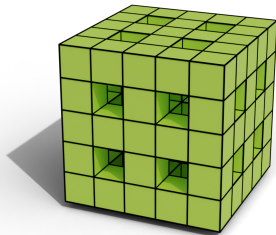
Graph transformations

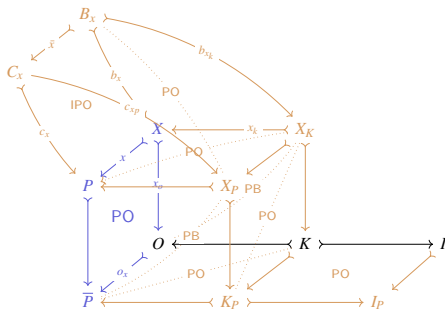
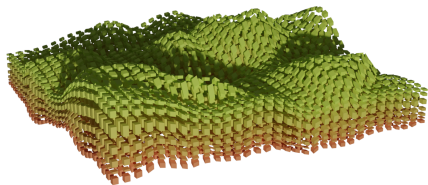
Application to geometric modeling

Romain Pascual

November 10, 2023

Centre Lagrange





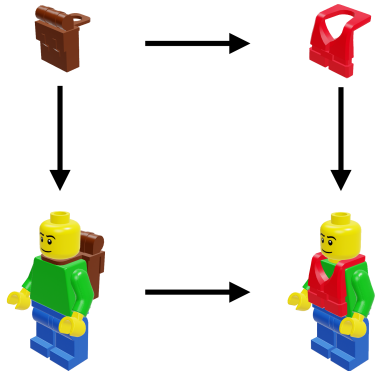
Disclaimer 1: I use category theory as a tool. If some results can be obtained more directly from topos theory, feel free to point it out (I will add your tool to my toolkit!)

Disclaimer 1: I use category theory as a tool. If some results can be obtained more directly from topos theory, feel free to point it out (I will add your tool to my toolkit!)

Disclaimer 2: One can study the 'usual' notions of rewriting in this framework. I will mostly present the formalisation of rewriting itself.

Graph transformations

► From strings to non-linear structures



String rewriting

Σ alphabet

$u \rightarrow v$ rewriting rules (u and v are words on Σ^*)

String rewriting

Σ alphabet

$u \rightarrow v$ rewriting rules (u and v are words on Σ^*)

$aba \rightarrow aaa$

String rewriting

Σ alphabet

$u \rightarrow v$ rewriting rules (u and v are words on Σ^*)

$aba \rightarrow aaa$

baa**ab**aab

String rewriting

Σ alphabet

$u \rightarrow v$ rewriting rules (u and v are words on Σ^*)

$$aba \rightarrow aaa$$

$$baaabaab \Rightarrow_{aba \rightarrow aaa} baaaaaab$$

String rewriting

Σ alphabet

$u \rightarrow v$ rewriting rules (u and v are words on Σ^*)

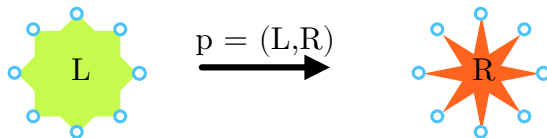
$$aba \rightarrow aaa$$

$$baaabaab \Rightarrow_{aba \rightarrow aaa} baaaab$$

- Chomsky hierarchy
- Generalized to abstract rewriting system $(\Sigma, \rightarrow)$

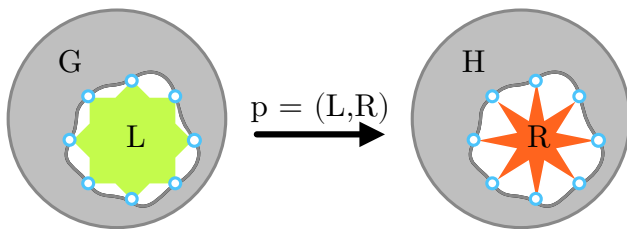
Graph transformations

Goal : Generalize from strings to graphs



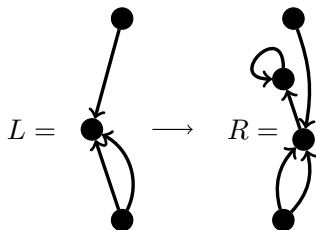
Graph transformations

Goal : Generalize from strings to graphs



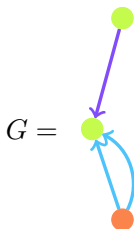
- Find in G a graph similar to L
- Remove it from G
- Reconnect R within the more global context

Rewriting rule on graphs?



How is this rule even defined? How can we apply it?

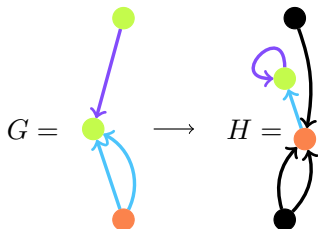
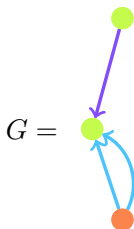
The category **Graph**



A graph $G = (V, E, s, t)$:

- a set of **nodes** V ,
- a set of **arcs** E ,
- a **source function** $s : E \rightarrow V$,
- a **target arrow** $t : E \rightarrow V$,

The category **Graph**



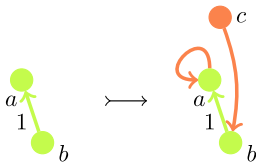
A graph $G = (V, E, s, t)$:

- a set of **nodes** V ,
- a set of **arcs** E ,
- a **source function** $s : E \rightarrow V$,
- a **target arrow** $t : E \rightarrow V$,

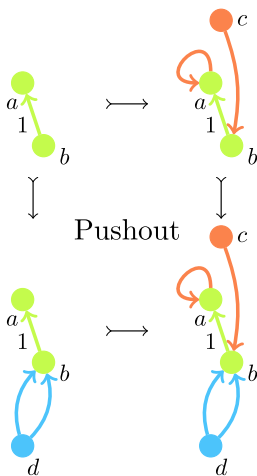
A morphism $G \rightarrow H$:

- a **node function** $V_G \rightarrow V_H$,
 - an **arc function** $E_G \rightarrow E_H$,
- preserving structure.

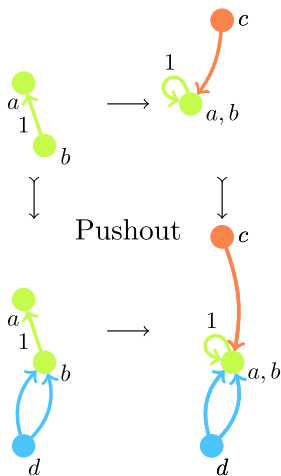
A graph rewriting?



A graph rewriting?



A graph rewriting?



Question:

Why is it insufficient?

How to compute pushouts?



A little digression ...

Pushout



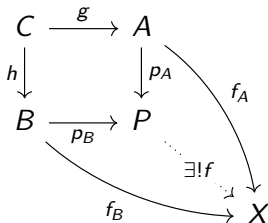
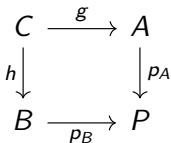
Colimit of the functor $(\bullet \leftarrow \bullet \rightarrow \bullet) \rightarrow \mathcal{C}$



Colimit of the functor $(\bullet \leftarrow \bullet \rightarrow \bullet) \rightarrow \mathcal{C}$

Pushout of $A \xleftarrow{g} C \xrightarrow{h} B$ is $A \xrightarrow{p_A} P \xleftarrow{p_B} B$ such that

- the left diagram commutes
- the construction is universal: for any other commutative square, there exists a unique morphism such that the right diagram commutes.



Pushout in **Set**



Pushout of $A \xleftarrow{g} C \xrightarrow{h} B$ in **Set** given by

$$P = (A \sqcup B)_{/\sim}$$

- $A \sqcup B$ disjoint union of A and B
- $g(c) \sim h(c)$ for all $c \in C$
- $p_X: X \rightarrow P$ maps $x \in X$ to its equivalence class ($X = A$ or B)

Pushout in **Set**



Pushout of $A \xleftarrow{g} C \xrightarrow{h} B$ in **Set** given by

$$P = (A \sqcup B)_{/\sim}$$

- $A \sqcup B$ disjoint union of A and B
- $g(c) \sim h(c)$ for all $c \in C$
- $p_X: X \rightarrow P$ maps $x \in X$ to its equivalence class ($X = A$ or B)

We can implement pushouts in **Set**!

Pushout in **Set**



Pushout of $A \xleftarrow{g} C \xrightarrow{h} B$ in **Set** given by

$$P = (A \sqcup B)_{/\sim}$$

- $A \sqcup B$ disjoint union of A and B
- $g(c) \sim h(c)$ for all $c \in C$
- $p_X: X \rightarrow P$ maps $x \in X$ to its equivalence class ($X = A$ or B)

We can implement pushouts in **Set**!

Extends to presheaves via pointwise computation (see AlgebraicJulia¹)
Certified for graphs in Isabelle/HOL²

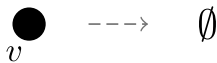
¹Brown et al. 2022.

²Söldner et al. 2022.

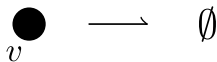
Question:

Why is it insufficient?

Deleting structure



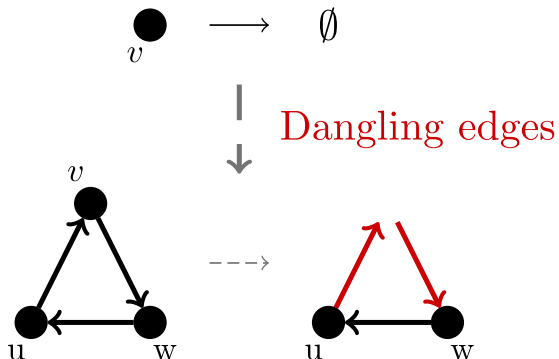
Deleting structure



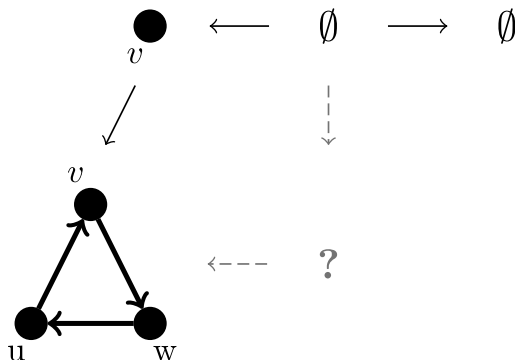
Deleting structure



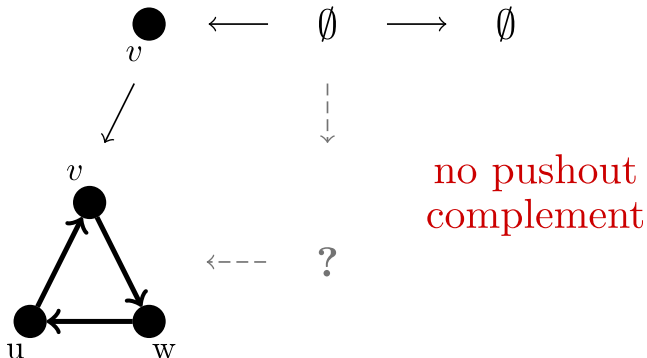
Deleting structure



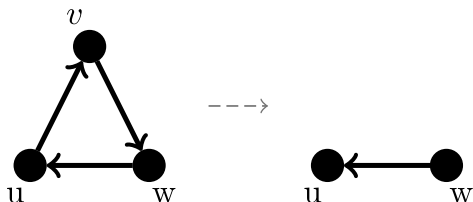
Deleting structure



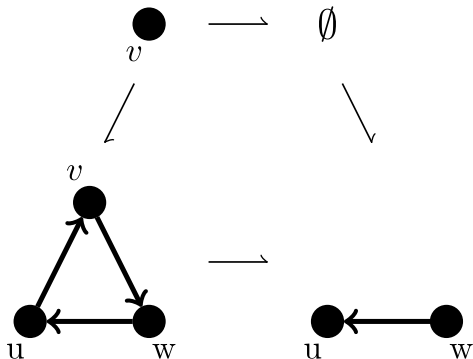
Deleting structure



Deleting structure

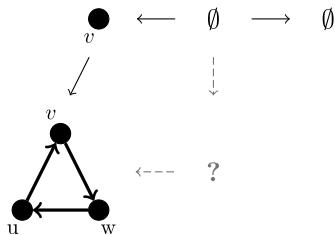


Deleting structure

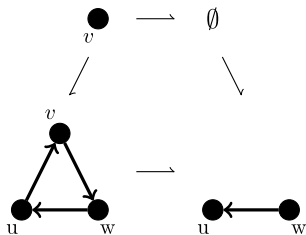


Multiple approaches¹

Double-Pushout



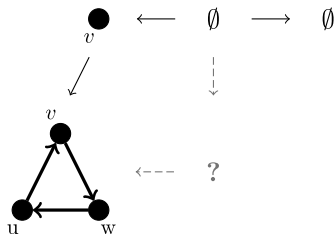
Single-Pushout



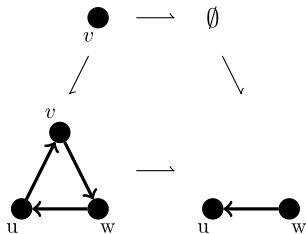
¹Rozenberg 1997.

Multiple approaches¹

Double-Pushout



Single-Pushout



Sesqui-Pushout, Double-Pullback, Pullback-Pushout, Double-Pushout on cospans ...

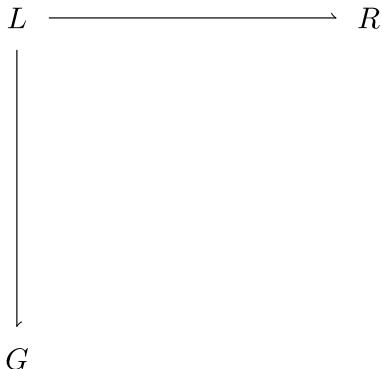
¹Rozenberg 1997.

A diagram chasing experience



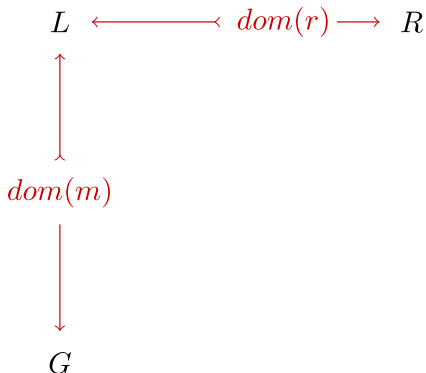
A little digression ...

Single Pushout rewriting¹ in **pGraph**



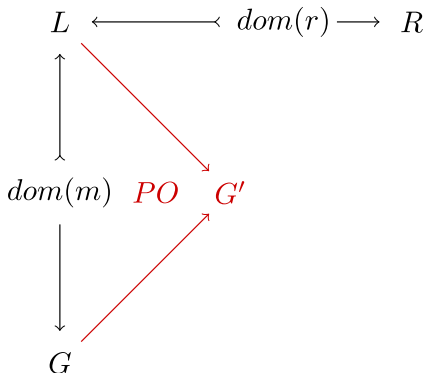
¹H. Ehrig et al. 1997.

Single Pushout rewriting¹ in **pGraph**



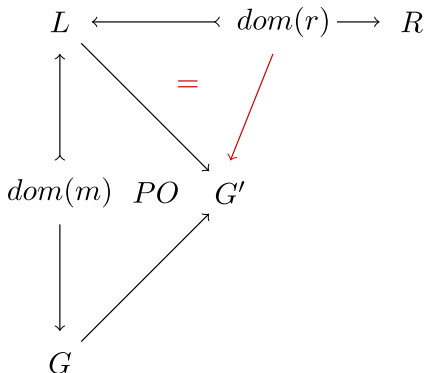
¹H. Ehrig et al. 1997.

Single Pushout rewriting¹ in **pGraph**



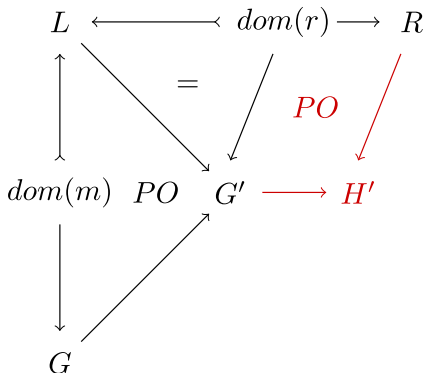
¹H. Ehrig et al. 1997.

Single Pushout rewriting¹ in **pGraph**



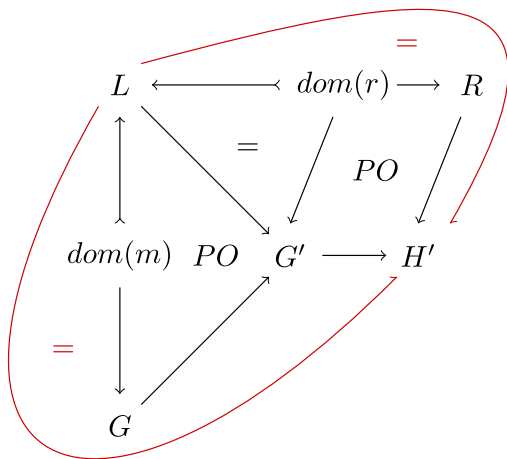
¹H. Ehrig et al. 1997.

Single Pushout rewriting¹ in **pGraph**

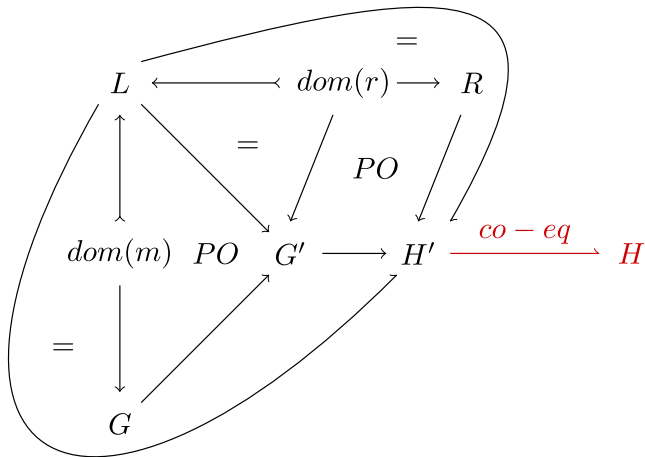


¹H. Ehrig et al. 1997.

Single Pushout rewriting¹ in **pGraph**

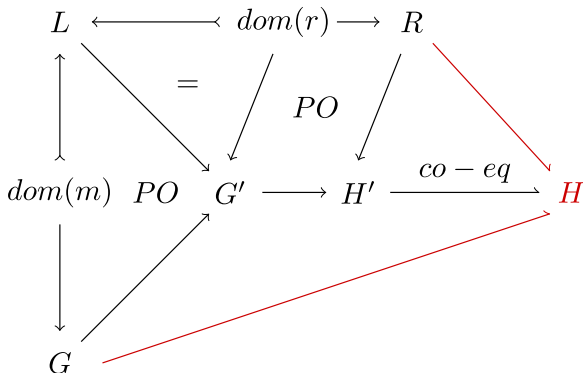


¹H. Ehrig et al. 1997.

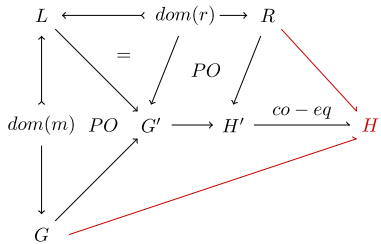


2. Graph transformations

Single Pushout rewriting¹ in **pGraph**

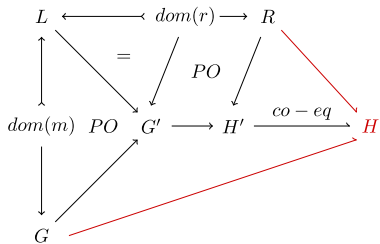


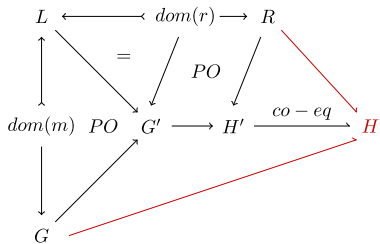
¹H. Ehrig et al. 1997.





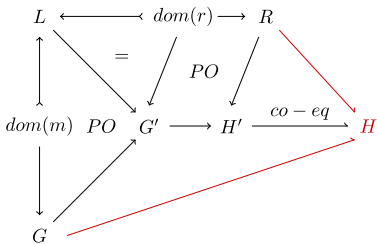
A diagram is a 'constructive' proof





A diagram is a 'constructive' proof

- a proof
- an algorithm



A diagram is a 'constructive' proof

- a proof
- an algorithm

Towards a formal proof?

See the CoREACT project

Rule and matching

A **rule** is a span $p = L \leftarrow I \rightarrow R$

- **L** Left-hand side
- **I** Interface
- **R** Right-hand side

Rule and matching

A **rule** is a span $p = L \leftarrow I \rightarrow R$

- Left-hand side
- Interface
- Right-hand side

A **match** for p is a morphism $m: L \rightarrow G$

Double-pushout rewriting

$p = L \leftarrow I \rightarrow R$ rewrites G via $m: L \rightarrow G$ if there exists a diagram

$$\begin{array}{ccccc} L & \longleftarrow & I & \longrightarrow & R \\ m \downarrow & & \downarrow & & \downarrow m^* \\ G & \longleftarrow & D & \longrightarrow & H \end{array}$$

where both squares are pushouts

$G \Rightarrow^{p,m} H$ is the **direct derivation** of G by p via m

Question:

Where is the issue?

Can we use any morphisms?



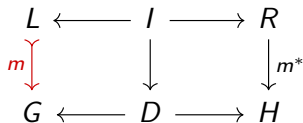
A little digression ...

Arbitrary vs monos



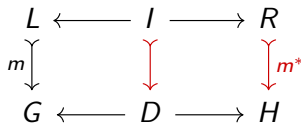
$$\begin{array}{ccccc} L & \longleftarrow & I & \longrightarrow & R \\ m \downarrow & & \downarrow & & \downarrow m^* \\ G & \longleftarrow & D & \longrightarrow & H \end{array}$$

¹Habel et al. 2001.



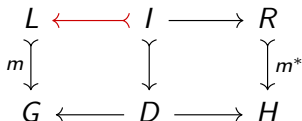
- 1 Monic matches do not lessen the expressiveness¹ and allow for explicit counting when deleting nodes or edges

¹Habel et al. 2001.



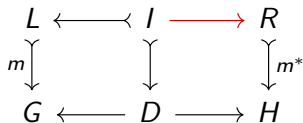
- 1 Monic matches do not lessen the expressiveness¹ and allow for explicit counting when deleting nodes or edges

¹Habel et al. 2001.



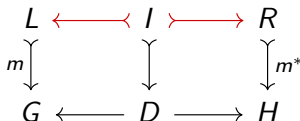
- 1 Monic matches do not lessen the expressiveness¹ and allow for explicit counting when deleting nodes or edges
- 2 $I \rightarrow L$ non-monic allows cloning nodes but result in non-deterministic transformations

¹Habel et al. 2001.



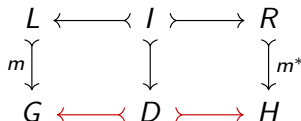
- 1 Monic matches do not lessen the expressiveness¹ and allow for explicit counting when deleting nodes or edges
- 2 $I \rightarrow L$ non-monic allows cloning nodes but result in non-deterministic transformations
- 3 $I \rightarrow R$ non-monic allows for merging

¹Habel et al. 2001.



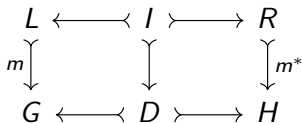
- 1 Monic matches do not lessen the expressiveness¹ and allow for explicit counting when deleting nodes or edges
- 2 $I \rightarrow L$ non-monic allows cloning nodes but result in non-deterministic transformations
- 3 $I \rightarrow R$ non-monic allows for merging
- 4 $I \rightharpoonup L$ and $I \rightharpoonup R$ monic allows for reversible transformations

¹Habel et al. 2001.



- 1 Monic matches do not lessen the expressiveness¹ and allow for explicit counting when deleting nodes or edges
- 2 $I \rightarrow L$ non-monic allows cloning nodes but result in non-deterministic transformations
- 3 $I \rightarrow R$ non-monic allows for merging
- 4 $I \rightharpoonup L$ and $I \rightharpoonup R$ monic allows for reversible transformations

¹Habel et al. 2001.



- 1 Monic matches do not lessen the expressiveness¹ and allow for explicit counting when deleting nodes or edges
- 2 $I \rightarrow L$ non-monic allows cloning nodes but result in non-deterministic transformations
- 3 $I \rightarrow R$ non-monic allows for merging
- 4 $I \rightharpoonup L$ and $I \rightharpoonup R$ monic allows for reversible transformations

See the summary 'From Conservative to Radical' in (Heckel et al. 2020)

¹Habel et al. 2001.

Question:

Where was the issue?

$$\begin{array}{ccccc}
 L & \longleftarrow & I & \longrightarrow & R \\
 m \downarrow & PO & \downarrow & PO & \downarrow m^* \\
 G & \longleftarrow & D & \longrightarrow & H
 \end{array}$$

- Does D exist?
- Is it even unique?

$$\begin{array}{ccccc}
 L & \longleftarrow & I & \longrightarrow & R \\
 m \downarrow & POC & \downarrow & PO & \downarrow m^* \\
 G & \longleftarrow & D & \longrightarrow & H
 \end{array}$$

- Does D exist? **Not necessarily!**
- Is it even unique? **Not necessarily!**

$$\begin{array}{ccccc}
 L & \longleftarrow & I & \longrightarrow & R \\
 m \downarrow & POC & \downarrow & PO & \downarrow m^* \\
 G & \longleftarrow & D & \longrightarrow & H
 \end{array}$$

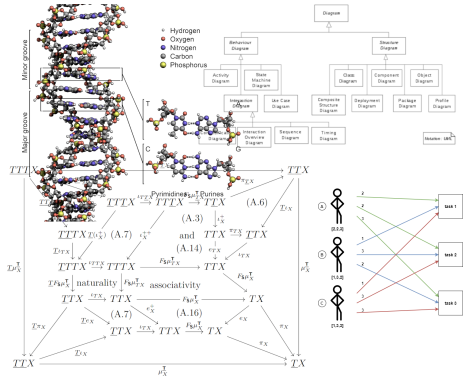
- Does D exist? **Not necessarily!**
- Is it even unique? **Not necessarily!**

A match $m: L \rightarrow G$ satisfies the **gluing condition** with respect to the rule $L \leftarrow I \rightarrow R$ if $K \rightarrow L \rightarrow R$ admits a pushout complement.

Set-based characterisation in **Graph**

A multiplicity of graphs

► How to add additional information to the graphs?



A graph is a presheaf on

$$E \rightrightarrows V$$

Add

- information (also called decorations¹)?
- constraints?

Note: I write presheaves using the $\mathcal{C}$ -Set convention², i.e., as copresheaves

¹Heckel et al. 2020.

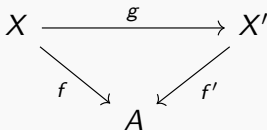
²Reyes et al. 2004.

Slice category

Let A be an object of a category $\mathcal{A}$.

The **slice category** of $\mathcal{A}$ over A , written $\mathcal{A}/A$ is the category with

- morphisms $X \rightarrow A$ in $\mathcal{A}$ as objects
- triangles



as morphisms

Comma categories

Let $\mathcal{A} \xrightarrow{P} \mathcal{C} \xleftarrow{Q} \mathcal{B}$ be categories and functors.

The **comma category** $(P \downarrow Q)$ is the category with

- triples (A, h, B) for objects with $A \in \mathcal{A}$, $B \in \mathcal{B}$, and $h: P(A) \rightarrow Q(B)$ in $\mathcal{C}$

Comma categories

Let $\mathcal{A} \xrightarrow{P} \mathcal{C} \xleftarrow{Q} \mathcal{B}$ be categories and functors.

The **comma category** $(P \downarrow Q)$ is the category with

- triples (A, h, B) for objects with $A \in \mathcal{A}$, $B \in \mathcal{B}$, and $h: P(A) \rightarrow Q(B)$ in $\mathcal{C}$
- pairs of morphisms $(f: A \rightarrow A', g: B \rightarrow B')$ for the morphism $(A, h, B) \rightarrow (A', h', B')$ such that

$$\begin{array}{ccc} P(A) & \xrightarrow{P(f)} & P(A') \\ h \downarrow & & \downarrow h' \\ Q(B) & \xrightarrow{Q(g)} & Q(B') \end{array}$$

commutes

Comma categories generalize slice categories

Proof: $\mathcal{A}/A$ is $(1_{\mathcal{A}} \downarrow A)$, where

- $1_{\mathcal{A}}: \mathcal{A} \rightarrow \mathcal{A}$ is the identity functor on the category $\mathcal{A}$
- the object A is identified with the unique functor $A: * \rightarrow \mathcal{A}$ that maps the unique object of $*$ to A

Undirected graphs



A little digression ...

Undirected graphs via involution



Undirected graphs: arcs are non-oriented

Undirected graphs via involution



Undirected graphs: arcs are non-oriented

In (Serre 1980): an undirected graph is a graph $G = (E, V, s, t)$ equipped with a fixed-point-free involution $i: E \rightarrow E$ such that $s \circ i = t$ and $t \circ i = s$.

Undirected graphs via involution



Undirected graphs: arcs are non-oriented

In (Serre 1980): an undirected graph is a graph $G = (E, V, s, t)$ equipped with a fixed-point-free involution $i: E \rightarrow E$ such that $s \circ i = t$ and $t \circ i = s$.

There is a categorical construction relying on schema theory to allow equational constraints on presheaves.

Undirected graphs as a comma category



In (Behr et al. 2020): **uGraph** is the comma category $(1_{\mathbf{Set}} \downarrow P_{(1,2)})$ where

- $1_{\mathbf{Set}}: \mathbf{Set} \rightarrow \mathbf{Set}$ is the identity functor on **Set**
- $P_{(1,2)}: \mathbf{Set} \rightarrow \mathbf{Set}$ is the $(1, 2)$ -restricted covariant power set functor

Spelled out, **uGraph** has objects $G = (V_G, E_G, i_G)$ with

- V_G set of nodes
- E_G set of arcs
- $i_G: E_G \rightarrow P_{(1,2)}(V_G)$ incidence function

and morphisms $m: G \rightarrow H = (m_V: V_G \rightarrow V_H, m_E: E_G \rightarrow E_H)$ such that

$$\begin{array}{ccc} E_G & \xrightarrow{m_E} & E_H \\ i_G \downarrow & & \downarrow i_H \\ P_{(1,2)}(V_G) & \xrightarrow{P_{(1,2)}(m_V)} & P_{(1,2)}(V_H) \end{array}$$

commutes

Labeled graphs

$\Sigma = (\Sigma_V, \Sigma_E)$ a pair of alphabets

- Σ_V vertex labels
- Σ_E edge labels

Labeled graphs

$\Sigma = (\Sigma_V, \Sigma_E)$ a pair of alphabets

- Σ_V vertex labels
- Σ_E edge labels

A **labeled** (or colored) graph is a presheaf on

$$\Sigma_E \xleftarrow{l_E} E \begin{matrix} \xrightarrow{s} \\ \xleftarrow{t} \end{matrix} V \xrightarrow{l_V} \Sigma_V$$

We obtain the category Σ -**Graph**

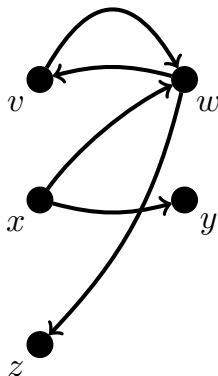
Typed graph

No relations between the labels in Σ -**Graph**!

Typed graph

No relations between the labels in Σ -**Graph**!

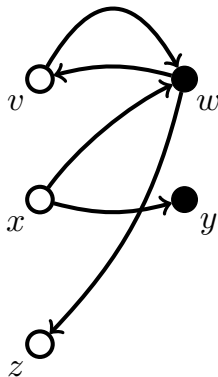
Bipartite graphs



Typed graph

No relations between the labels in Σ -**Graph**!

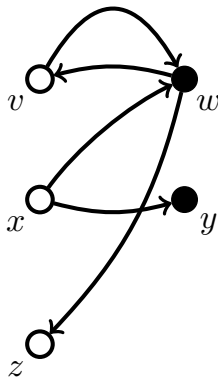
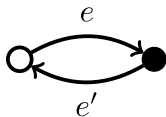
Bipartite graphs:black or white vertices



Typed graph

No relations between the labels in Σ -**Graph**!

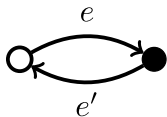
Bipartite graphs: black or white vertices,
bi-color arcs



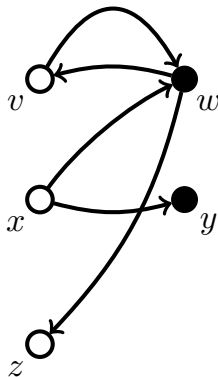
Typed graph

No relations between the labels in Σ -**Graph**!

Bipartite graphs: black or white vertices,
bi-color arcs



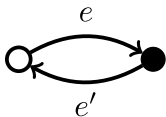
The **type graph** TG describe the relations
between the decorations



Typed graph

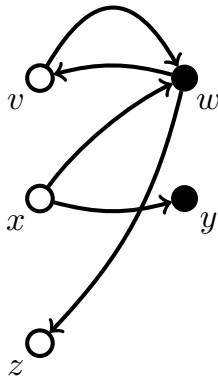
No relations between the labels in Σ -**Graph**!

Bipartite graphs: black or white vertices,
bi-color arcs



The **type graph** TG describe the relations
between the decorations

The category of **graphs typed over TG** is the
slice category **Graph**/TG



Typed graphs generalize labeled graphs



Proof: Given $\Sigma = (\Sigma_V, \Sigma_E)$, Σ -**Graph** is **Graph**/ $\text{TG}(\Sigma)$ with $\text{TG}(\Sigma) = (V_\Sigma, E_\Sigma, s_\Sigma, t_\Sigma)$ defined as

- $V_\Sigma = \Sigma_V$
- $E_\Sigma = \Sigma_E \times \Sigma_V \times \Sigma_V$
- $s_\Sigma: E_\Sigma \rightarrow V_\Sigma; (e, a, b) \mapsto a$
- $t_\Sigma: E_\Sigma \rightarrow V_\Sigma; (e, a, b) \mapsto b$

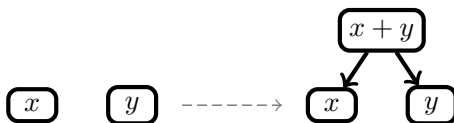
Computing with the information

Labeled or typed graph: **immutable** information

Computing with the information

Labeled or typed graph: **immutable** information

We might want to decorate the rules with expressions that compute values



Computing with the information

Labeled or typed graph: **immutable** information

We might want to decorate the rules with expressions that compute values



Write rules decorated with terms and use the match to evaluate them

Data types (Syntax)

Signature: $\Omega = (S, F)$

- S sorts
- F function names with a profile mapping $\rho: F \rightarrow S^* \times S$

Form the category **Sig**(Ω)

$T_\Omega(X) = \bigsqcup_{s \in S} T_\Omega(X)_s$ are the **Ω -terms** with variables in $(X_s)_{s \in S}$ (defined inductively from function names)

Data types (Semantics)

Ω -algebra: $\mathcal{A}$ given by

- non-empty carrier sets $(\mathcal{A}_s)_{s \in S}$
- a function $f^{\mathcal{A}}: \mathcal{A}_{s_1} \times \dots \times \mathcal{A}_{s_m} \rightarrow \mathcal{A}_s$ for each function name f with profile $\rho(f) = (s_1, \dots, s_m, s)$

Form the category **Alg**(Ω)

Term algebra

The Ω -term algebra $\mathcal{T}_\Omega(X)$ over the set of variables X maps each expression to its string representation

- $T_\Omega(X)_s$ as carrier sets
- $f^{\mathcal{T}_\Omega(X)}(t_1, \dots, t_n)$ is the string $f(t_1, \dots, t_n)$

Term algebra

The Ω -term algebra $\mathcal{T}_\Omega(X)$ over the set of variables X maps each expression to its string representation

- $\mathcal{T}_\Omega(X)_s$ as carrier sets
- $f^{\mathcal{T}_\Omega(X)}(t_1, \dots, t_n)$ is the string $f(t_1, \dots, t_n)$

Extends any function $f: X \rightarrow \mathcal{A}$ where $\mathcal{A}$ is an Ω -algebra to a unique algebra morphism $f^*: \mathcal{T}_\Omega(X) \rightarrow \mathcal{A}$ (the evaluation of terms)

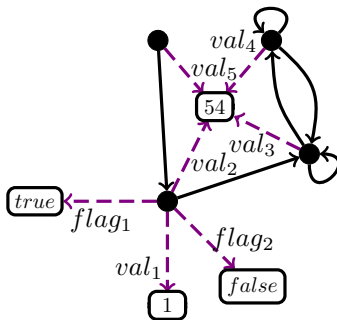
Graphs with data

E-Graph: presheaf on

$$E \begin{array}{c} \xrightarrow{s} \\ \xleftarrow{t} \end{array} V \xleftarrow{sa_V} A_V \xrightarrow{ta_V} D_V$$

Here: no data on the arcs for simplicity

An E-graph



Attributed graphs

Ω -attributed graph $AG = (G, \mathcal{A})$

- G is an E-graph
- $\mathcal{A}$ is an Ω -algebra

such that

$$D_G = \bigsqcup_{s \in S} \mathcal{A}_s$$

Morphism of attributed graphs: morphism of Ω -algebra and morphism of E-graph such that the obvious diagrams commute

Form the category $\Omega\text{-}\mathbf{AGraph}$

Attributed graphs

Ω -attributed graph $AG = (G, \mathcal{A})$

- G is an E-graph
- $\mathcal{A}$ is an Ω -algebra

such that

$$D_G = \bigsqcup_{s \in S} \mathcal{A}_s$$

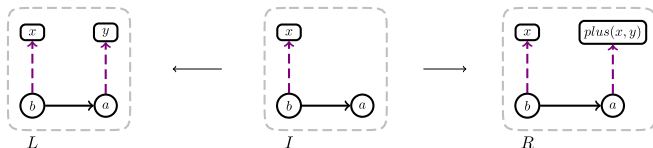
Morphism of attributed graphs: morphism of Ω -algebra and morphism of E-graph such that the obvious diagrams commute

Form the category $\Omega\text{-}\mathbf{AGraph}$

One can type attributed graphs by slicing (using the terminal algebra)

Attributed graph rewriting

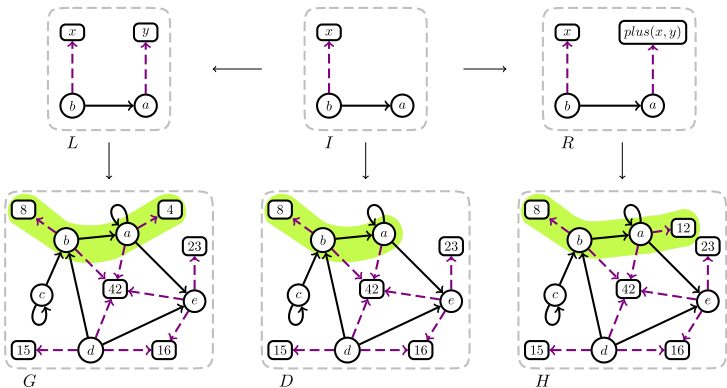
Ω -attributed rule: span where the graphs share the algebra $\mathcal{T}_\Omega(X)$



Attributed graph rewriting

Ω -attributed rule: span where the graphs share the algebra $\mathcal{T}_\Omega(X)$

In $G \Rightarrow^{p,m} H$, the match $m: L \rightarrow G$ provides the evaluation of the terms appearing in L by mapping the variables to values in G .



Attributed graphs prevent **structure-depending computations**

‘Average the values of the neighbor of a node’

¹Ghamarian et al. 2012.

Attributed graphs prevent **structure-depending computations**

‘Average the values of the neighbor of a node’

Solved using **multi-rules** in Groove¹

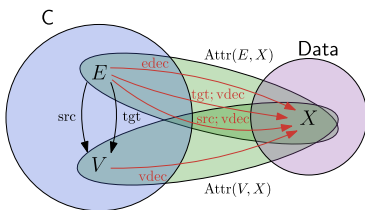
¹Ghamarian et al. 2012.

Attributed graphs prevent **structure-depending computations**

‘Average the values of the neighbor of a node’

Solved using **multi-rules** in Groove¹

AlgebraicJulia models attributed graphs via the double category of profunctors



¹Ghamarian et al. 2012.

Various categories of graphs

- **Graph**
- **uGraph**
- **Σ -Graph**
- **Ω -AGraph**
- **sGraph** (simple graphs, i.e., without parallel arcs)
- hypergraphs
- ...

Various categories of graphs

- **Graph**
- **uGraph**
- **Σ -Graph**
- **Ω -AGraph**
- **sGraph** (simple graphs, i.e., without parallel arcs)
- hypergraphs
- ...

All admit DPO as a semantics for rewriting!

Various categories of graphs

- **Graph**
- **uGraph**
- **Σ -Graph**
- **Ω -AGraph**
- **sGraph** (simple graphs, i.e., without parallel arcs)
- hypergraphs
- ...

All admit DPO as a semantics for rewriting!

$$\begin{array}{ccccc} L & \longleftarrow & I & \longrightarrow & R \\ m \downarrow & POC & \downarrow & PO & \downarrow m^* \\ G & \longleftarrow & D & \longrightarrow & H \end{array}$$

- Does D exist? **Not necessarily!**
- Is it even unique? **Not necessarily!**

Various categories of graphs

- **Graph**
- **uGraph**
- **Σ -Graph**
- **Ω -AGraph**
- **sGraph** (simple graphs, i.e., without parallel arcs)
- hypergraphs
- ...

All admit DPO as a semantics for rewriting!

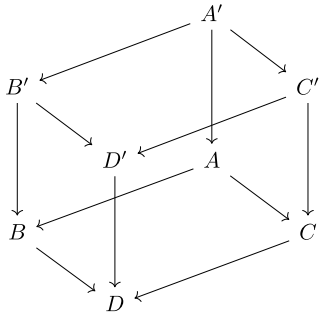
$$\begin{array}{ccccc} L & \xleftarrow{\quad} & I & \xrightarrow{\quad} & R \\ m \downarrow & POC & \downarrow & PO & \downarrow m^* \\ G & \xleftarrow{\quad} & D & \xrightarrow{\quad} & H \end{array}$$

- Does D exist? **Not necessarily!**
- Is it even unique? **Not necessarily!**

Can we axiomatize DPO rewriting?

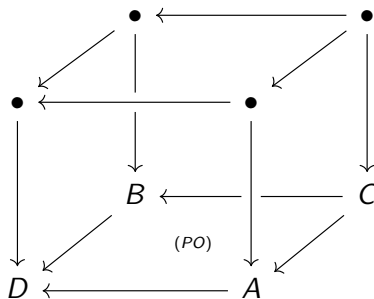
Adhesive categories

- Axiomatizing DPO rewriting



van Kampen square¹

A pushout is a **van Kampen square** if for all commutative cubes



such that the right and back faces are pullbacks, then the top face is a pushout if and only if the left and front faces are pullbacks

¹Lack et al. 2004.

Adhesive category

A category is **adhesive** if

- it has pullbacks

¹Lack et al. 2005.

²Johnstone et al. 2007.

Adhesive category

A category is **adhesive** if

- it has pullbacks
- it has pushouts along monomorphisms, i.e., pushouts where at least one of the span morphisms is a monomorphism

¹Lack et al. 2005.

²Johnstone et al. 2007.

Adhesive category

A category is **adhesive** if

- it has pullbacks
- it has pushouts along monomorphisms, i.e., pushouts where at least one of the span morphisms is a monomorphism
- pushouts along monomorphism are van Kampen squares

¹Lack et al. 2005.

²Johnstone et al. 2007.

Adhesive category

A category is **adhesive** if

- it has pullbacks
- it has pushouts along monomorphisms, i.e., pushouts where at least one of the span morphisms is a monomorphism
- pushouts along monomorphism are van Kampen squares

Example of adhesive categories: **Set**, any presheaf topos¹, any elementary topos²

¹Lack et al. 2005.

²Johnstone et al. 2007.

DPO rewriting in adhesive categories

In an adhesive category, the pushout complement of $A \rightrightarrows B \rightarrow C$, called pushout complement of monos, is unique up to isomorphism, provided it exists.

Therefore one usually considers linear rules (i.e., spans of monos) in an adhesive category.

DPO rewriting in adhesive categories

In an adhesive category, the pushout complement of $A \rightrightarrows B \rightarrow C$, called pushout complement of monos, is unique up to isomorphism, provided it exists.

Therefore one usually considers linear rules (i.e., spans of monos) in an adhesive category.

Adhesivity implies all the ‘usual’ rewriting results on the DPO semantics

$\mathcal{M}$ -adhesivity¹

Ω -**AGraph** is not adhesive

¹Hartmut Ehrig et al. 2006.

$\mathcal{M}$ -adhesivity¹

Ω -**AGraph** is not adhesive

A category is $\mathcal{M}$ -adhesive if:

- it has pullbacks along $\mathcal{M}$ -morphisms
- it has pushouts along $\mathcal{M}$ -morphisms
- pushouts along $\mathcal{M}$ -morphisms are van Kampen squares

where $\mathcal{M}$ is a class of monomorphisms containing all isomorphisms, stable under composition, pullback, and pushout.

¹Hartmut Ehrig et al. 2006.

$\mathcal{M}$ -adhesivity¹

Ω -**AGraph** is not adhesive

A category is $\mathcal{M}$ -adhesive if:

- it has pullbacks along $\mathcal{M}$ -morphisms
- it has pushouts along $\mathcal{M}$ -morphisms
- pushouts along $\mathcal{M}$ -morphisms are van Kampen squares

where $\mathcal{M}$ is a class of monomorphisms containing all isomorphisms, stable under composition, pullback, and pushout.

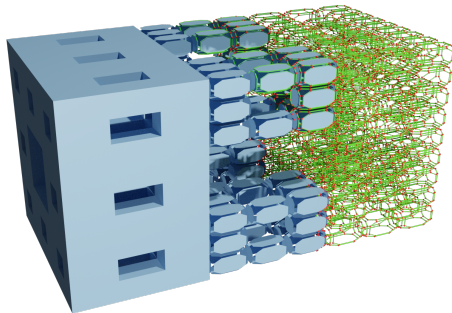
Ω -**AGraph** is $\mathcal{M}$ -adhesive where $\mathcal{M}$ -morphisms consist of monos of E-graphs and isomorphisms of algebra

¹Hartmut Ehrig et al. 2006.

Some categories in which we (want to) do DPO rewriting are not $\mathcal{M}$ -adhesive, e.g., rotation systems

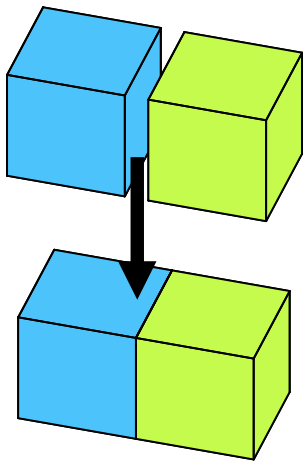
Geometric modeling

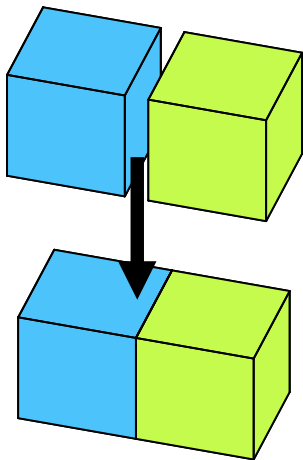
- Formalizing modeling operations as graph transformations







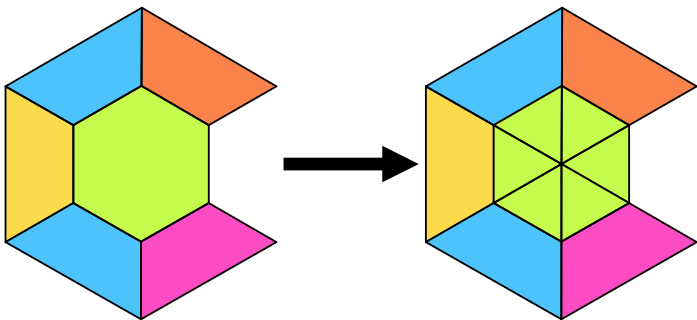




CGAL's sew operation

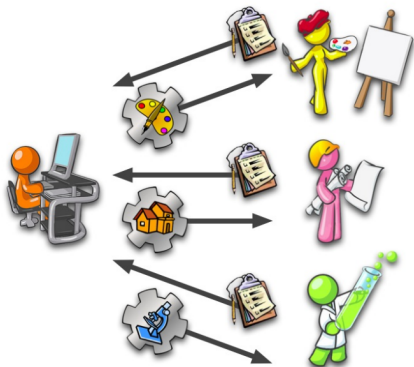
```
template<unsigned int i>
void sew(Dart_descriptor adart1, Dart_descriptor adart2)

{
    CGAL_assertion( i<=dimension );
    CGAL_assertion( (is_sewable<i>(adart1,adart2)) );
    size_type amark=get_new_mark();
    CGAL::GMap_dart_iterator_basic_of_involution<Self, i>
        I1(*this, adart1, amark);
    CGAL::GMap_dart_iterator_basic_of_involution<Self, i>
        I2(*this, adart2, amark);
    for ( ; I1.cont(); ++I1, ++I2 )
    {
        Helper::template Foreach_enabled_attributes_except
            <CGAL::internal::GMap_group_attribute_functor<Self, i>, i>::
                run(*this, I1, I2);
    }
    negate_mark( amark );
    for ( I1.rewind(), I2.rewind(); I1.cont(); ++I1, ++I2 )
    {
        basic_link_alpha<i>(I1, I2);
    }
    negate_mark( amark );
    CGAL_assertion( is_whole_map_unmarked(amarck) );
    free_mark(amarck);
}
```

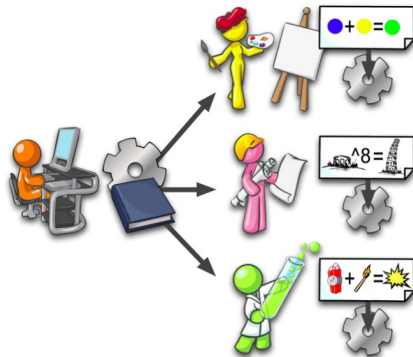


Ambition : define a *domain-specific language* (DSL) for geometric modeling

Motivations : abstraction, performance, conciseness, correctness, simplicity

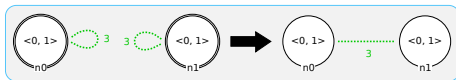


Classical approach

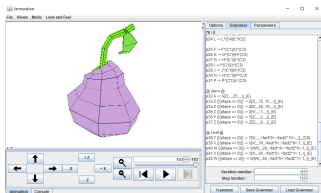


Jerboa¹

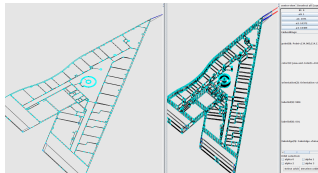
¹Belhaouari et al. 2014



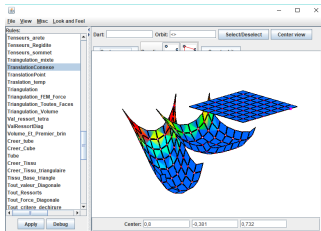
Plant growth



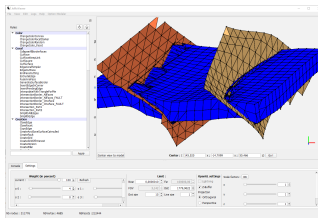
Architecture



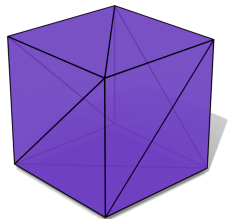
Spring-mass simulations



Geology

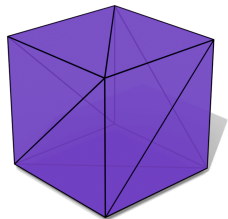


Topology vs geometry

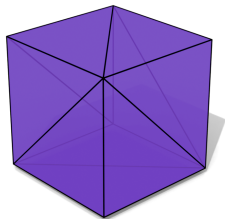


Object

Topology vs geometry

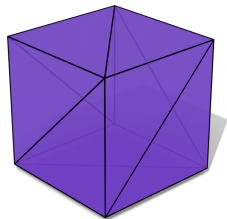


Object

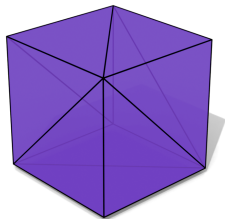


Different topologie

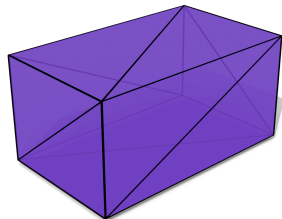
Topology vs geometry



Object

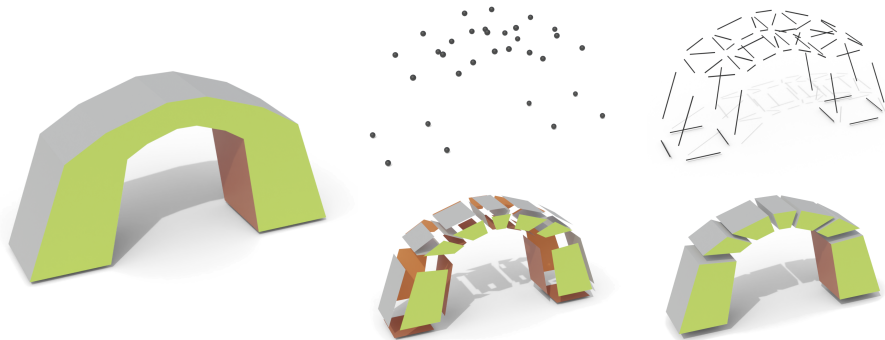


Different topologie

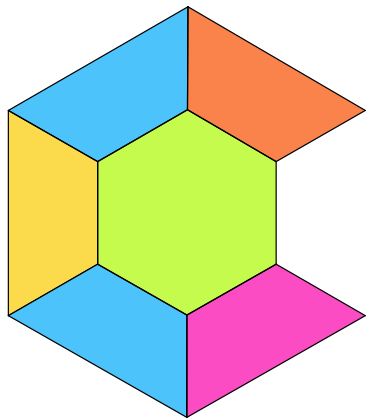


Different geometry

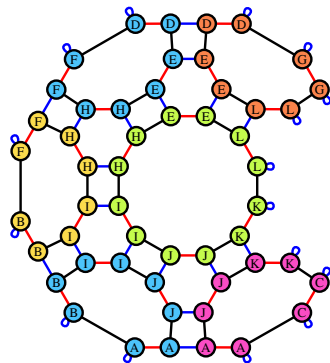
Topological cells



A combinatorial description of topology



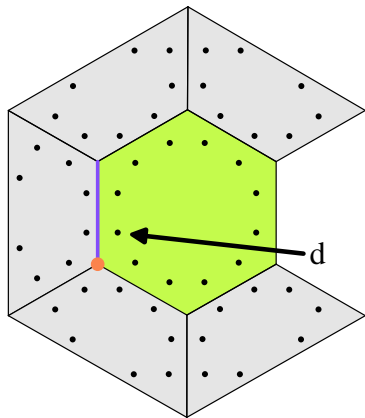
A combinatorial description of topology



Gmaps built as graphs

Color legend: 0, 1, 2

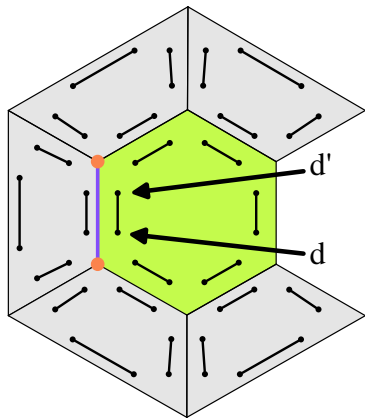
A combinatorial description of topology



Gmaps built as graphs

d : orange vertex, purple edge, the green face

A combinatorial description of topology



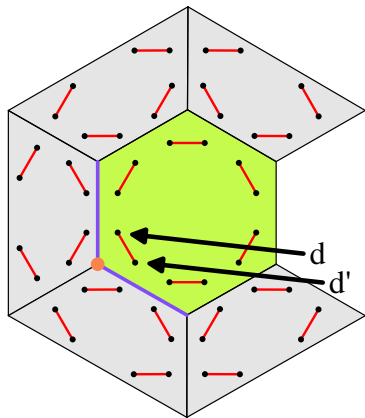
Gmaps built as graphs

0-arc:

- distinct vertices
- same edge and faces

Color legend: 0

A combinatorial description of topology



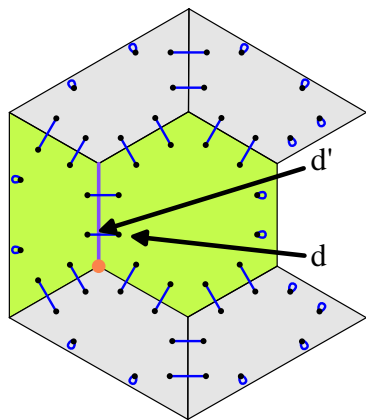
Gmaps built as graphs

1-arc:

- distinct edges
- same vertices and faces

Color legend: 1

A combinatorial description of topology



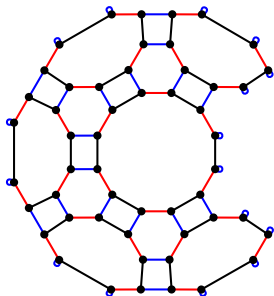
Gmaps built as graphs

2-arc:

- distinct faces
- same vertices and edges

Color legend: 0, 1, 2

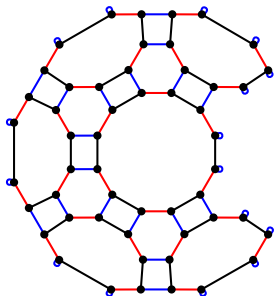
Generalized maps¹ (topology)



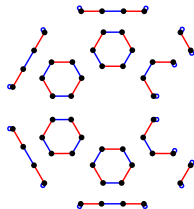
Legend: 0, 1, 2

¹Damiand et al. 2014.

Generalized maps¹ (topology)



Orbit: Sub-graph induced by a subset $\langle o \rangle$ of dimensions

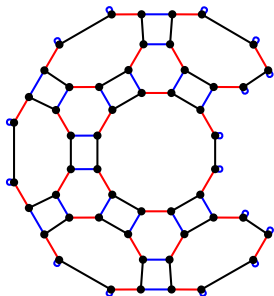


Legend: 0, 1, 2

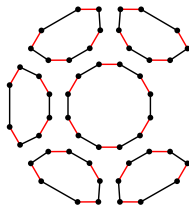
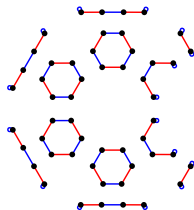
Vertices: orbits $\langle 1, 2 \rangle$

¹Damiand et al. 2014.

Generalized maps¹ (topology)



Orbit: Sub-graph induced by a subset $\langle o \rangle$ of dimensions



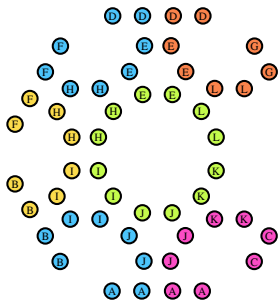
Legend: 0, 1, 2

Vertices: orbits $\langle 1, 2 \rangle$

Faces: orbits $\langle 0, 1 \rangle$

¹Damiand et al. 2014.

Embeddings (geometry)

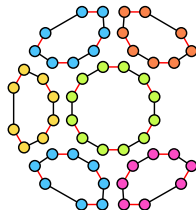
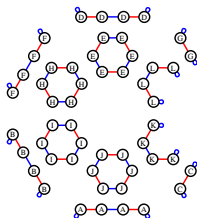
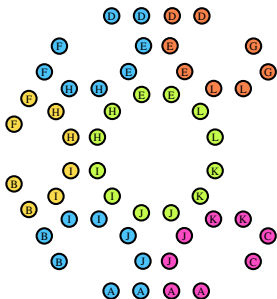


Legend: 0, 1, 2

Embeddings (geometry)

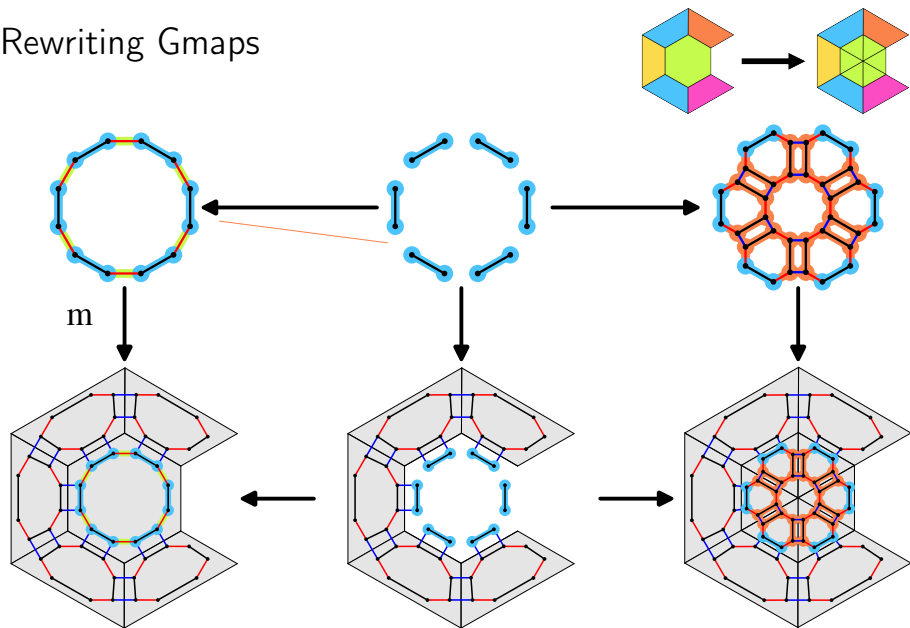


Embedding: function $\pi : \langle o_\pi \rangle \rightarrow \tau_\pi$
with τ_π an abstract data type

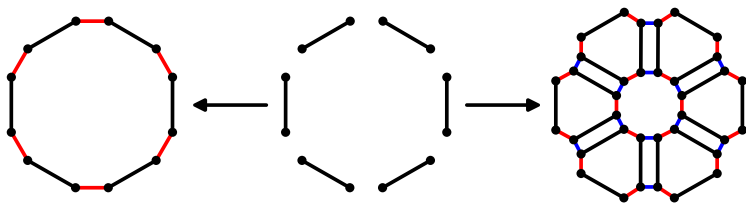
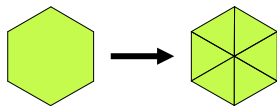


Legend: 0, 1, 2 $position : \langle 1, 2 \rangle \rightarrow \text{Point3}$ $color : \langle 0, 1 \rangle \rightarrow \text{ColorRGB}$

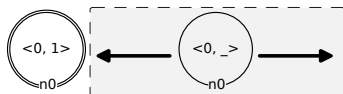
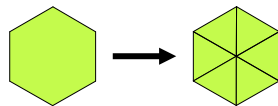
Rewriting Gmaps



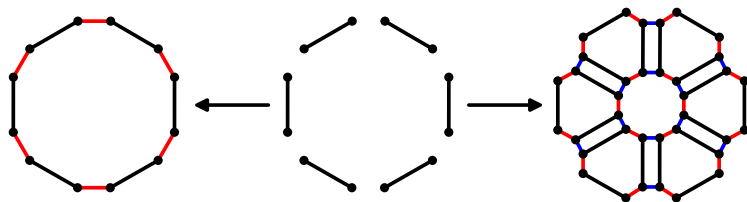
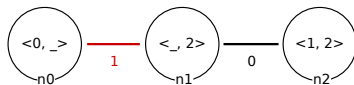
Orbit rewriting



Orbit rewriting

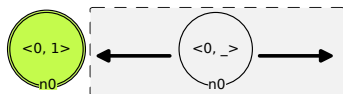
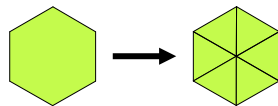


Implicitly
computed

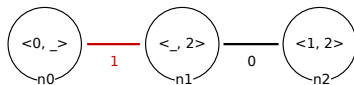


Instantiated
rule

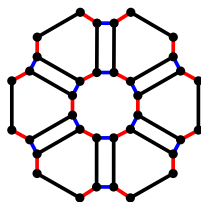
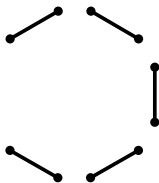
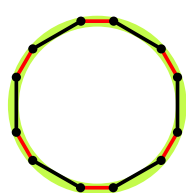
Orbit rewriting



Implicitly
computed

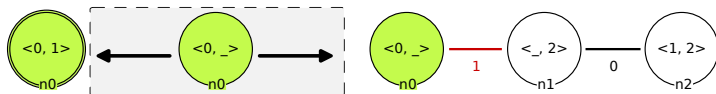
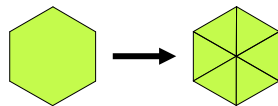


Local



Instantiated
rule

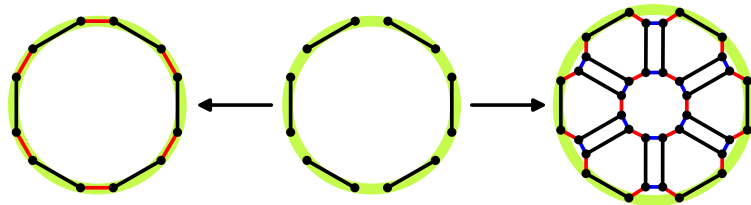
Orbit rewriting



Implicitly
computed

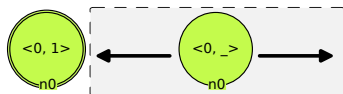
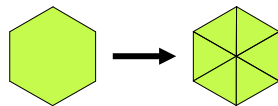


Local

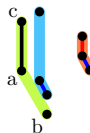


Instantiated
rule

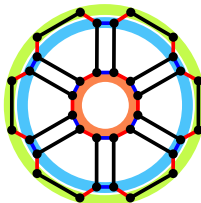
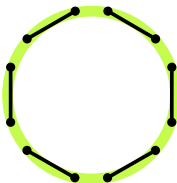
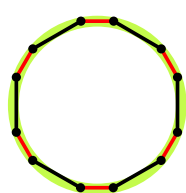
Orbit rewriting



Implicitly
computed

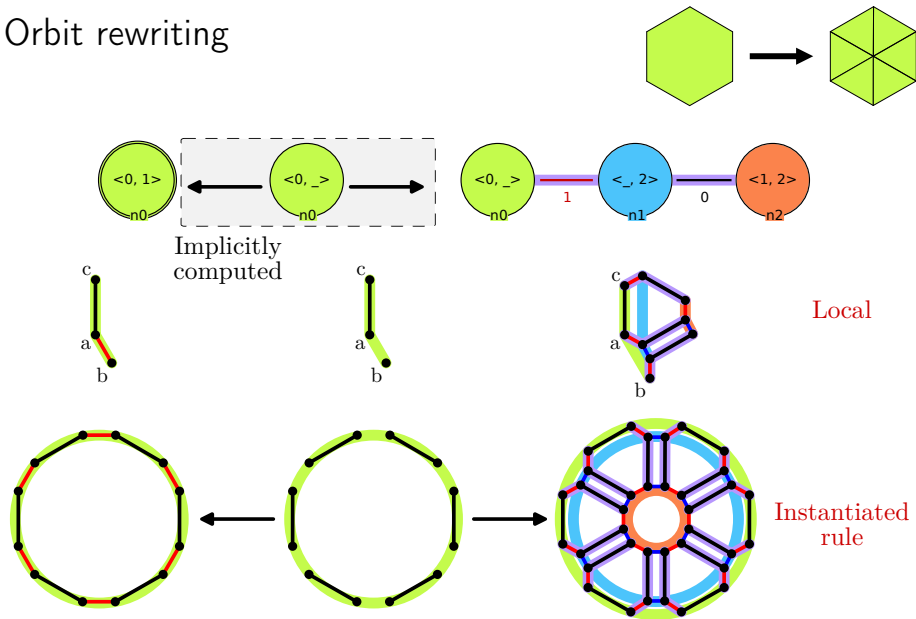


Local

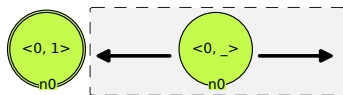
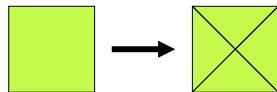


Instantiated
rule

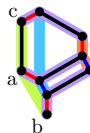
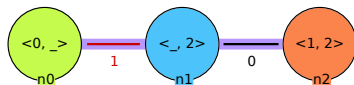
Orbit rewriting



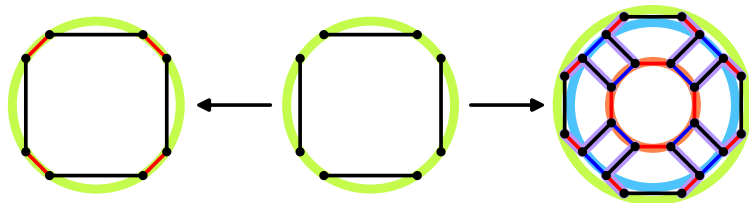
Orbit rewriting



Implicitly
computed



Local



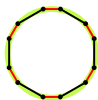
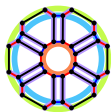
Instantiated
rule

Orbit rewriting explained categorically



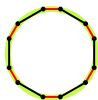
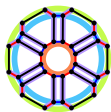
A little digression ...

Graph products¹



¹inspired from Bauderon 1995.

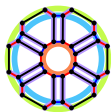
Graph products¹



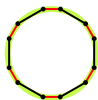
P

¹inspired from Bauderon 1995.

Graph products¹



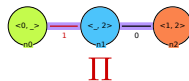
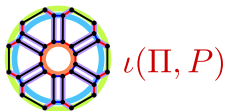
Π



P

¹inspired from Bauderon 1995.

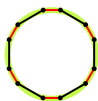
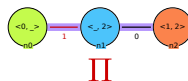
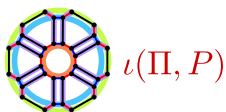
Graph products¹



- $\iota(\Pi, P)$: instantiation

¹inspired from Bauderon 1995.

Graph products¹

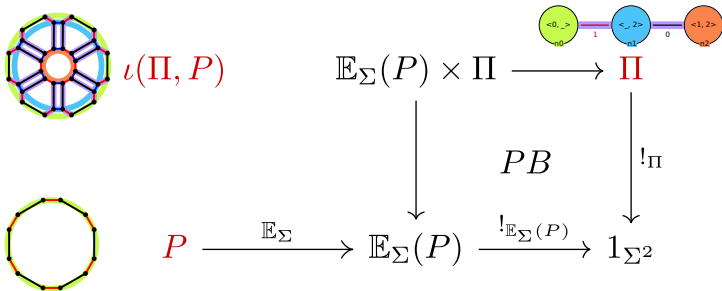


$$P \xrightarrow{\mathbb{E}_\Sigma} \mathbb{E}_\Sigma(P)$$

- $\iota(\Pi, P)$: instantiation
- $\mathbb{E}_\Sigma$: embedding functor

¹inspired from Bauderon 1995.

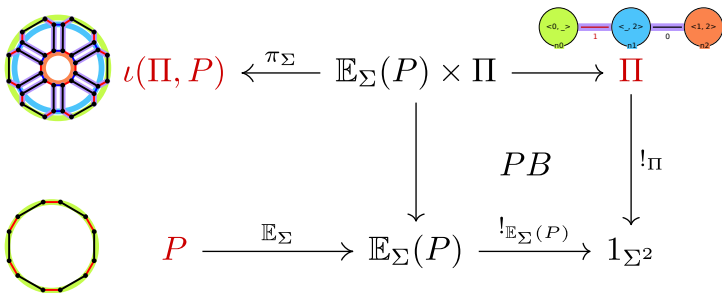
Graph products¹



- $\iota(\Pi, P)$: instantiation
- $\mathbb{E}_\Sigma$: embedding functor

¹inspired from Bauderon 1995.

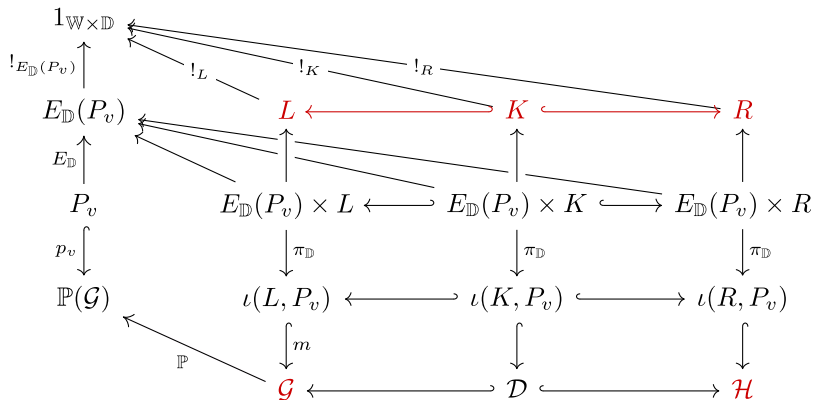
Graph products¹



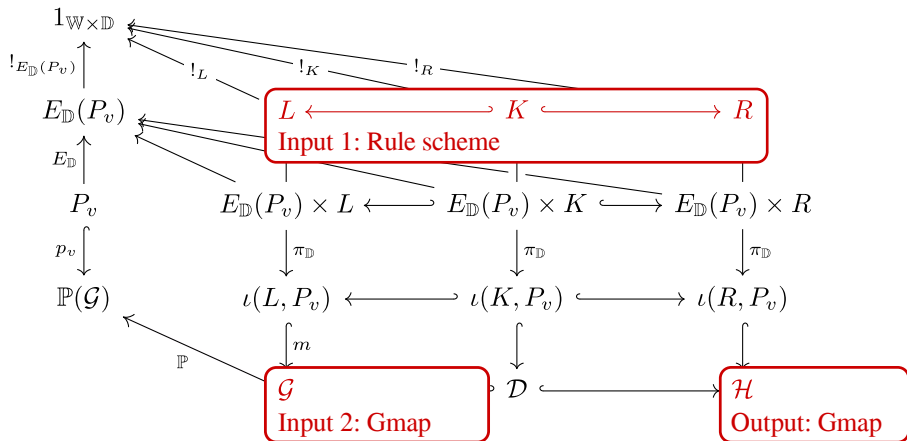
- $\iota(\Pi, P)$: instantiation
- $\mathbb{E}_\Sigma$: embedding functor
- π_Σ : projecting functor

¹inspired from Bauderon 1995.

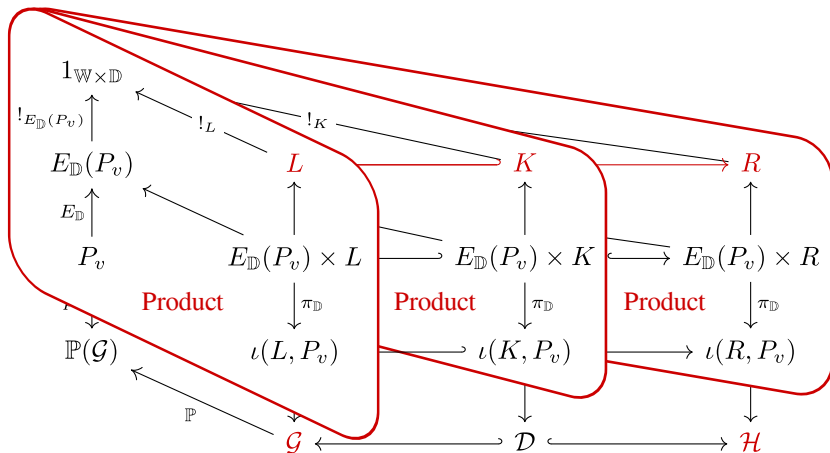
Complete construction



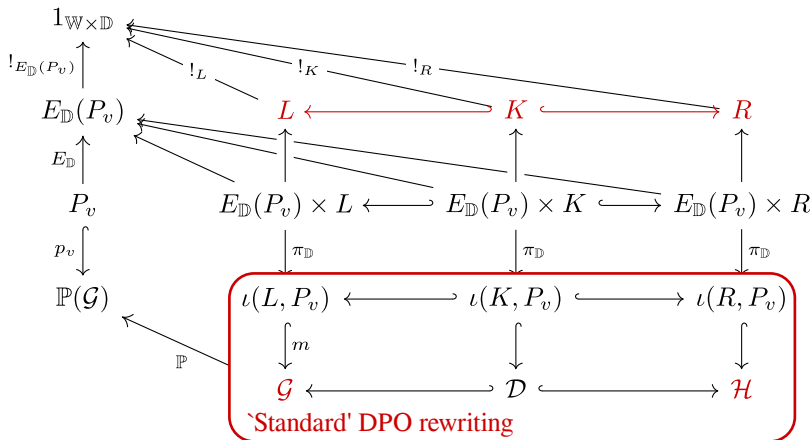
Complete construction



Complete construction



Complete construction



Consistency preservation

Modifications of a well-formed object should produce an equally well-formed object.

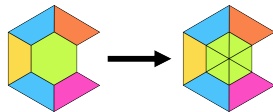
Consistency preservation

Modifications of a well-formed object should produce an equally well-formed object.

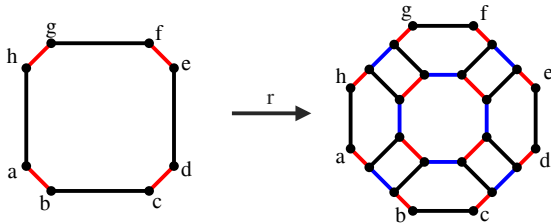
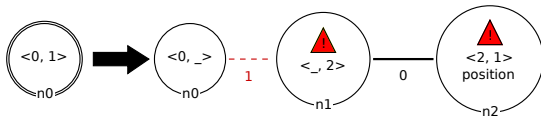
Intuitively, if $G \models c$ and $G \Rightarrow^{p,m} H$, how can we ensure that $H \models c$?

Requirement: Feedback to the rule designer.

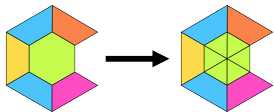
Breaking the topological consistency



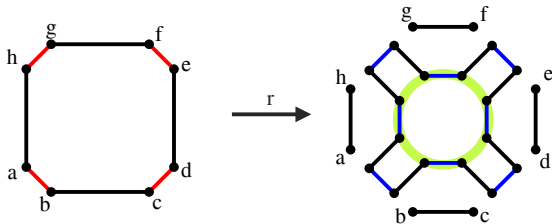
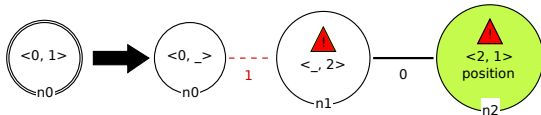
Constraint: 0202 paths should be cycles.



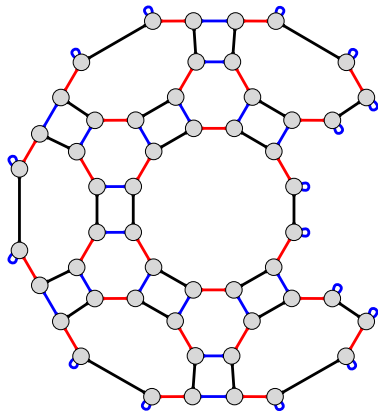
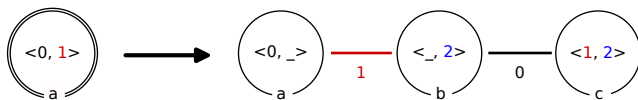
Breaking the topological consistency



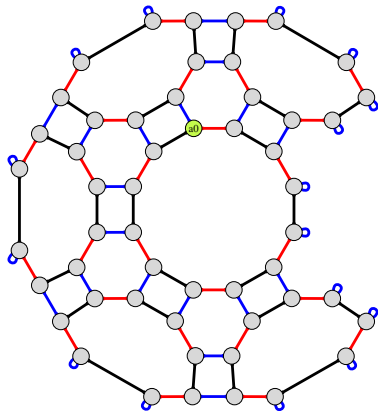
Constraint: 0202 paths should be cycles.



Instantiation: constructive approach

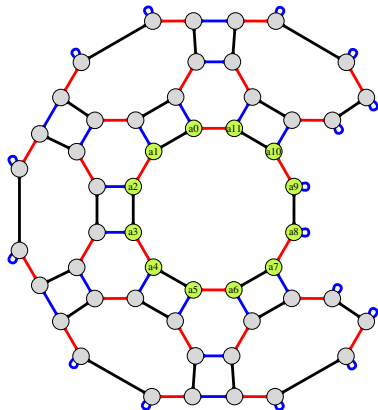


Instantiation: constructive approach



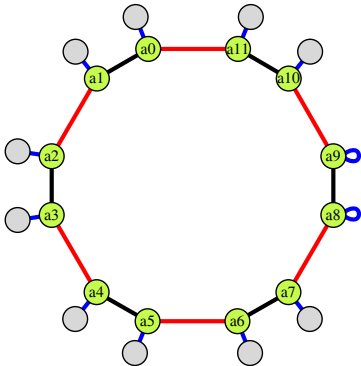
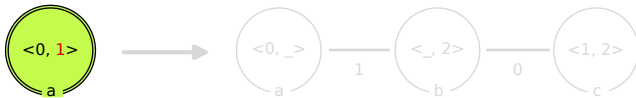
a_0		

Instantiation: constructive approach



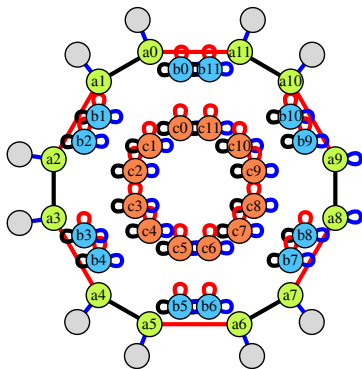
a		
a_0		
a_1		
a_2		
a_3		
a_4		
a_5		
$\vdots$		

Instantiation: constructive approach



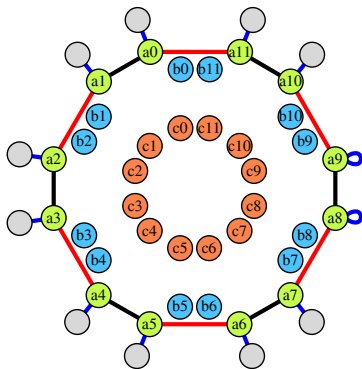
a		
a_0		
a_1		
a_2		
a_3		
a_4		
a_5		
$\vdots$		

Instantiation: constructive approach



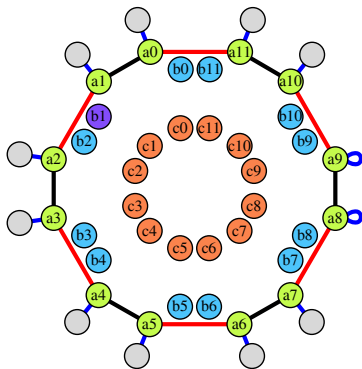
a	b	c
a_0	b_0	c_0
a_1	b_1	c_1
a_2	b_2	c_2
a_3	b_3	c_3
a_4	b_4	c_4
a_5	b_5	c_5
$\vdots$	$\vdots$	$\vdots$

Instantiation: constructive approach



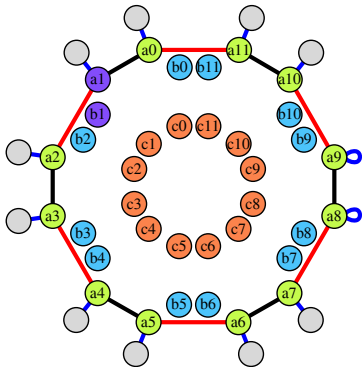
a	b	c
a_0	b_0	c_0
a_1	b_1	c_1
a_2	b_2	c_2
a_3	b_3	c_3
a_4	b_4	c_4
a_5	b_5	c_5
$\vdots$	$\vdots$	$\vdots$

Instantiation: constructive approach



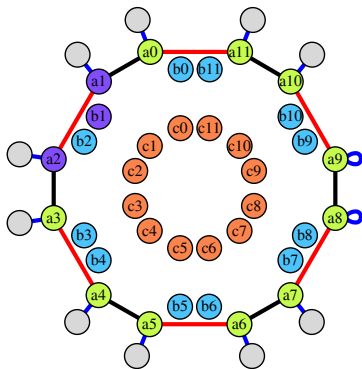
a	b	c
a_0	b_0	c_0
a_1	b_1	c_1
a_2	b_2	c_2
a_3	b_3	c_3
a_4	b_4	c_4
a_5	b_5	c_5
$\vdots$	$\vdots$	$\vdots$

Instantiation: constructive approach



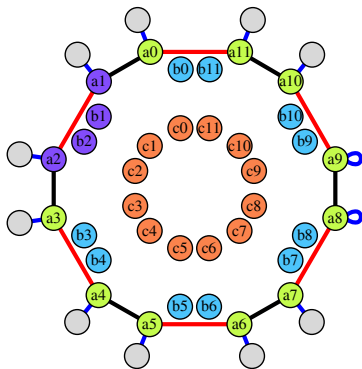
<i>a</i>	<i>b</i>	<i>c</i>
<i>a</i> ₀	<i>b</i> ₀	<i>c</i> ₀
<i>a</i> ₁	<i>b</i> ₁	<i>c</i> ₁
<i>a</i> ₂	<i>b</i> ₂	<i>c</i> ₂
<i>a</i> ₃	<i>b</i> ₃	<i>c</i> ₃
<i>a</i> ₄	<i>b</i> ₄	<i>c</i> ₄
<i>a</i> ₅	<i>b</i> ₅	<i>c</i> ₅
⋮	⋮	⋮

Instantiation: constructive approach



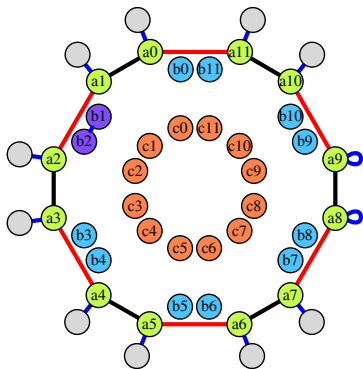
a	b	c
a ₀	b ₀	c ₀
a ₁	b ₁	c ₁
a ₂	b ₂	c ₂
a ₃	b ₃	c ₃
a ₄	b ₄	c ₄
a ₅	b ₅	c ₅
⋮	⋮	⋮

Instantiation: constructive approach



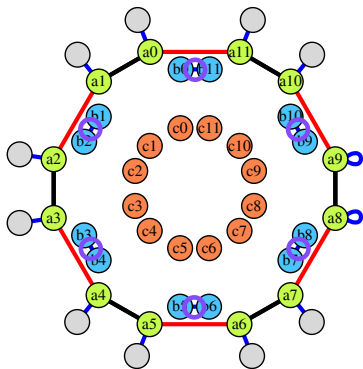
<i>a</i>	<i>b</i>	<i>c</i>
<i>a</i> ₀	<i>b</i> ₀	<i>c</i> ₀
<i>a</i> ₁	<i>b</i> ₁	<i>c</i> ₁
<i>a</i> ₂	<i>b</i> ₂	<i>c</i> ₂
<i>a</i> ₃	<i>b</i> ₃	<i>c</i> ₃
<i>a</i> ₄	<i>b</i> ₄	<i>c</i> ₄
<i>a</i> ₅	<i>b</i> ₅	<i>c</i> ₅
⋮	⋮	⋮

Instantiation: constructive approach



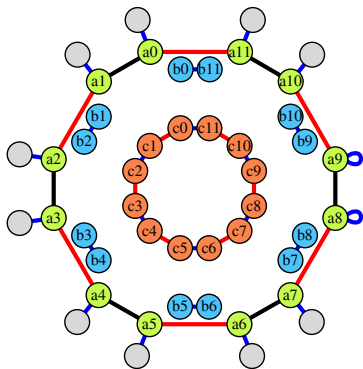
a	b	c
a ₀	b ₀	c ₀
a ₁	b ₁	c ₁
a ₂	b ₂	c ₂
a ₃	b ₃	c ₃
a ₄	b ₄	c ₄
a ₅	b ₅	c ₅
⋮	⋮	⋮

Instantiation: constructive approach



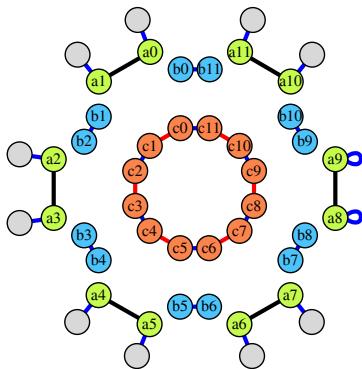
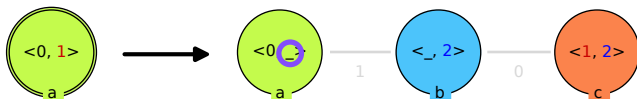
a	b	c
a_0	b_0	c_0
a_1	b_1	c_1
a_2	b_2	c_2
a_3	b_3	c_3
a_4	b_4	c_4
a_5	b_5	c_5
$\vdots$	$\vdots$	$\vdots$

Instantiation: constructive approach



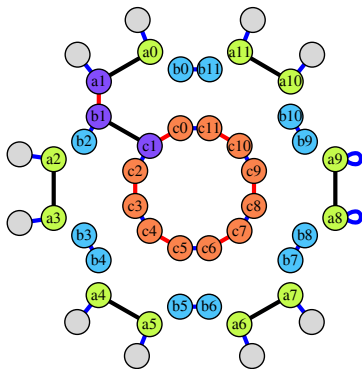
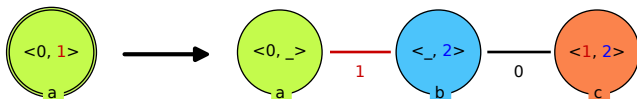
<i>a</i>	<i>b</i>	<i>c</i>
a_0	b_0	c_0
a_1	b_1	c_1
a_2	b_2	c_2
a_3	b_3	c_3
a_4	b_4	c_4
a_5	b_5	c_5
$\vdots$	$\vdots$	$\vdots$

Instantiation: constructive approach



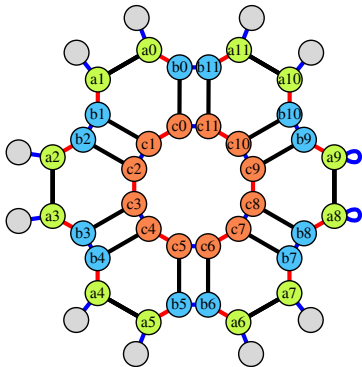
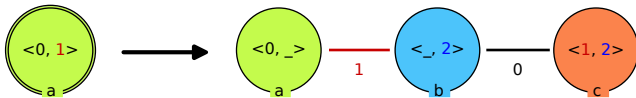
<i>a</i>	<i>b</i>	<i>c</i>
a_0	b_0	c_0
a_1	b_1	c_1
a_2	b_2	c_2
a_3	b_3	c_3
a_4	b_4	c_4
a_5	b_5	c_5
$\vdots$	$\vdots$	$\vdots$

Instantiation: constructive approach



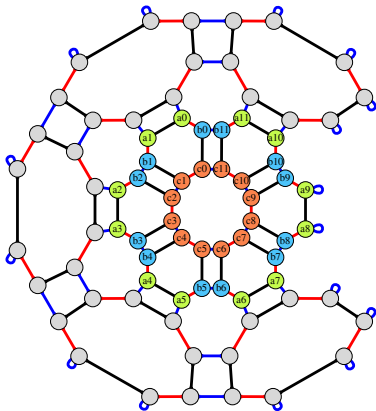
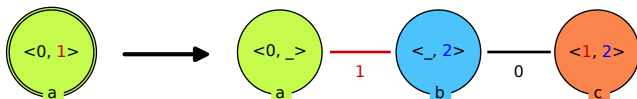
a	b	c
a_0	b_0	c_0
a_1	b_1	c_1
a_2	b_2	c_2
a_3	b_3	c_3
a_4	b_4	c_4
a_5	b_5	c_5
$\vdots$	$\vdots$	$\vdots$

Instantiation: constructive approach



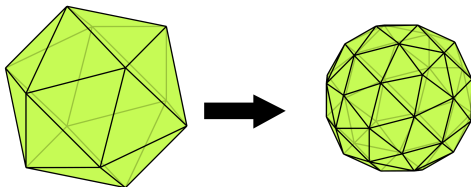
<i>a</i>	<i>b</i>	<i>c</i>
<i>a</i> ₀	<i>b</i> ₀	<i>c</i> ₀
<i>a</i> ₁	<i>b</i> ₁	<i>c</i> ₁
<i>a</i> ₂	<i>b</i> ₂	<i>c</i> ₂
<i>a</i> ₃	<i>b</i> ₃	<i>c</i> ₃
<i>a</i> ₄	<i>b</i> ₄	<i>c</i> ₄
<i>a</i> ₅	<i>b</i> ₅	<i>c</i> ₅
⋮	⋮	⋮

Instantiation: constructive approach



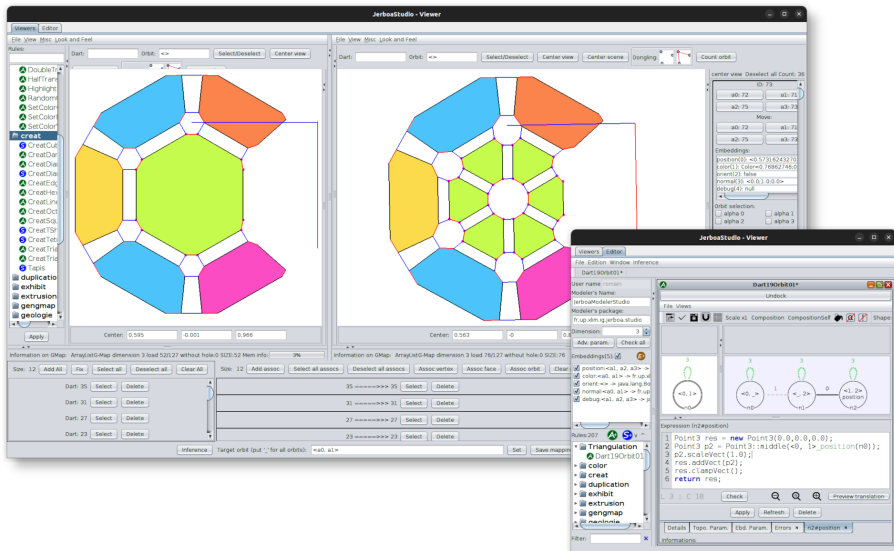
<i>a</i>	<i>b</i>	<i>c</i>
<i>a</i> ₀	<i>b</i> ₀	<i>c</i> ₀
<i>a</i> ₁	<i>b</i> ₁	<i>c</i> ₁
<i>a</i> ₂	<i>b</i> ₂	<i>c</i> ₂
<i>a</i> ₃	<i>b</i> ₃	<i>c</i> ₃
<i>a</i> ₄	<i>b</i> ₄	<i>c</i> ₄
<i>a</i> ₅	<i>b</i> ₅	<i>c</i> ₅
⋮	⋮	⋮

From an abstract language to code



```
int n = 0;
JerboaList <@ebd<position>> sommetsAdjacents ;
for (JerboaDart d : <1,2>_<2>(n0)) {
    sommetsAdjacents.add(d@0.position);
    n = n+1;
}
Point3 bary = Point3::middle(sommetsAdjacents);

if (n==2){
    return Point3::barycenter(n0.position, 3.0/4.0, bary, 1.0/4.0);
}
else if (n==3){
    return Point3::barycenter(n0.position, 7.0/16.0, bary, 9.0/16.0);
}
else {
    return Point3::barycenter(n0.position, 5.0/8.0, bary, 3.0/8.0);
}
```



References I

-  Bauderon, Michel (Jan. 1, 1995). “Parallel Rewriting of Graphs through the Pullback Approach”. In: **Electronic Notes in Theoretical Computer Science**. SEGRAGRA 1995 2, pp. 19–26. ISSN: 1571-0661. DOI: 10.1016/S1571-0661(05)80176-8.
-  Behr, Nicolas et al. (July 3, 2020). “Rule Algebras for Adhesive Categories”. In: **Logical Methods in Computer Science** Volume 16, Issue 3. DOI: 10.23638/LMCS-16(3:2)2020. arXiv: 1807.00785.
-  Belhaouari, Hakim et al. (2014). “Jerboa: A Graph Transformation Library for Topology-Based Geometric Modeling”. In: **Graph Transformation**. ICGT 2014. Ed. by Holger Giese et al. Lecture Notes in Computer Science. Cham: Springer International Publishing, pp. 269–284. ISBN: 978-3-319-09108-2. DOI: 10.1007/978-3-319-09108-2_18.

References II



Brown, Kristopher et al. (2022). “Computational Category-Theoretic Rewriting”. In: **Graph Transformation**. Ed. by Nicolas Behr et al. Lecture Notes in Computer Science. Cham: Springer International Publishing, pp. 155–172. ISBN: 978-3-031-09843-7. DOI: 10.1007/978-3-031-09843-7_9.



Damiand, Guillaume et al. (Sept. 19, 2014). **Combinatorial Maps: Efficient Data Structures for Computer Graphics and Image Processing**. CRC Press. 407 pp. ISBN: 978-1-4822-0652-4.



Ehrig, H. et al. (Feb. 1, 1997). “Algebraic Approaches to Graph Transformation, Part II: Single Pushout Approach and Comparison with Double Pushout Approach”. In: **Handbook of Graph Grammars and Computing by Graph Transformation: Volume I. Foundations**. USA: World Scientific, pp. 247–312. ISBN: 978-981-02-2884-2. DOI: 10.5555/278918.278930.

References III



Ehrig, Hartmut et al. (2006). **Fundamentals of Algebraic Graph Transformation**. Monographs in Theoretical Computer Science. An EATCS Series. Berlin Heidelberg: Springer-Verlag. ISBN: 978-3-540-31187-4. DOI: 10.1007/3-540-31188-2.



Ghamarian, Amir Hossein et al. (Feb. 1, 2012). “Modelling and analysis using GROOVE”. In: **International Journal on Software Tools for Technology Transfer** 14.1, pp. 15–40. ISSN: 1433-2787. DOI: 10.1007/s10009-011-0186-x.



Habel, Annegret et al. (Oct. 1, 2001). “Double-pushout graph transformation revisited”. In: **Mathematical Structures in Computer Science** 11.5, pp. 637–688. ISSN: 0960-1295. DOI: 10.1017/S0960129501003425.

References IV



Heckel, Reiko et al. (2020). **Graph Transformation for Software Engineers: With Applications to Model-Based Development and Domain-Specific Language Engineering**. Cham: Springer International Publishing. ISBN: 978-3-030-43915-6. DOI: 10.1007/978-3-030-43916-3.



Johnstone, Peter T. et al. (2007). “Quasitoposes, Quasiadhesive Categories and Artin Glueing”. In: **Algebra and Coalgebra in Computer Science**. Ed. by Till Mossakowski et al. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, pp. 312–326. ISBN: 978-3-540-73859-6. DOI: 10.1007/978-3-540-73859-6_21.

References V

-  Lack, Stephen et al. (2004). “Adhesive Categories”. In: **Foundations of Software Science and Computation Structures**. Ed. by Igor Walukiewicz. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, pp. 273–288. ISBN: 978-3-540-24727-2. DOI: 10.1007/978-3-540-24727-2_20.
-  — (July 1, 2005). “Adhesive and quasiadhesive categories”. In: **RAIRO - Theoretical Informatics and Applications** 39.3, pp. 511–545. ISSN: 0988-3754, 1290-385X. DOI: 10.1051/ita:2005028.
-  Reyes, Marie La Palme et al. (2004). “Generic Figures and Their Glueings: A Constructive Approach to Functor Categories”. In: p. 267.
-  Rozenberg, Grzegorz, ed. (Feb. 1, 1997). **Handbook of Graph Grammars and Computing by Graph Transformation: Volume I. Foundations**. Vol. Foundations. 1 vols. USA: World Scientific Publishing Co., Inc. 545 pp. ISBN: 978-981-02-2884-2.

References VI



Serre, Jean-Pierre (1980). **Trees**. Springer. ISBN: 978-3-642-61856-7.



Söldner, Robert et al. (Dec. 22, 2022). “Towards Mechanised Proofs in Double-Pushout Graph Transformation”. In: **Electronic Proceedings in Theoretical Computer Science** 374, pp. 59–75. ISSN: 2075-2180. DOI: 10.4204/EPTCS.374.6.