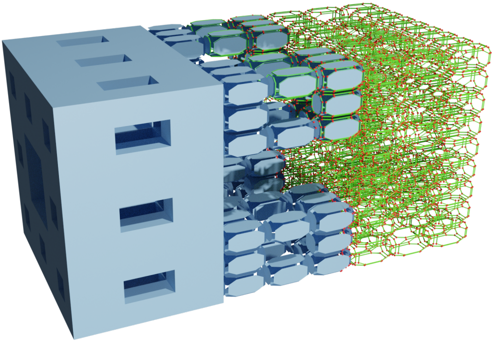
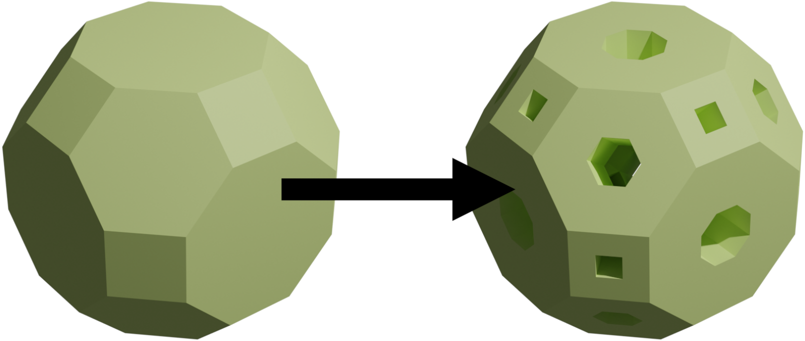
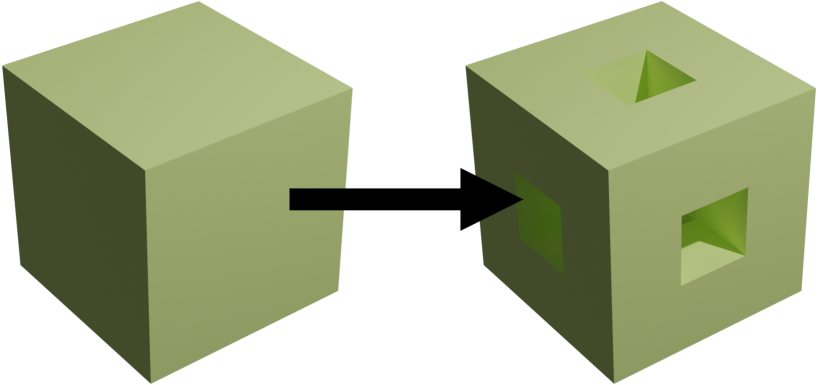
Inference of geometric modeling operations using generalized maps



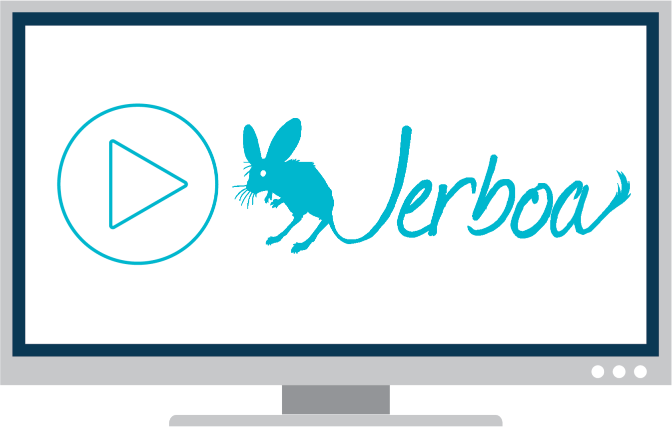
Romain Pascual, Hakim Belhaouari, Agnès Arnould, Pascale Le Gall

June 16th 2022





Demo

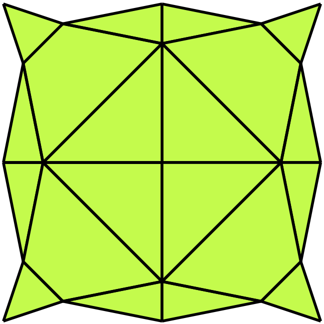


1. Generalized maps
   - Generalized maps
   - Embedded G-maps
2. Graph rewriting
   - Graph Rewriting
   - G-map rewriting
3. Topological inference
   - Main objective
   - Method
   - Examples and results
4. Geometric inference
   - Main objective
   - Method
   - Examples and results

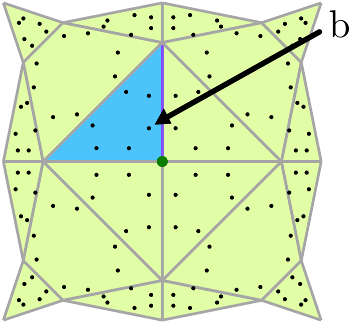
Generalized maps

- Geometric objects are represented with embedded generalize maps.

Generalized maps [Damiand and Lienhardt 2004]



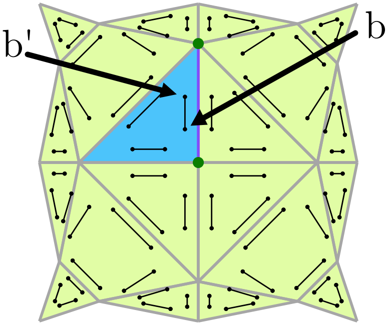
Generalized maps [Damiand and Lienhardt 2004]



G-maps built as graphs.

b identifies the green vertex, the purple edge and the blue face.

Generalized maps [Damiand and Lienhardt 2004]

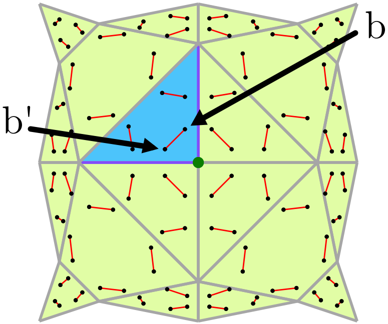


G-maps built as graphs.

0-arc :

- distinct vertices.
- same edges and faces.

Generalized maps [Damiand and Lienhardt 2004]

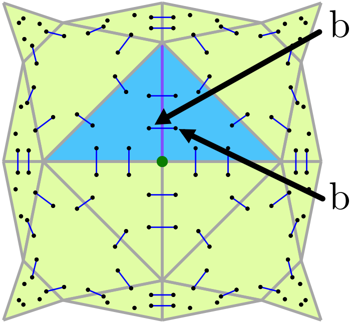


G-maps built as graphs.

1-arc :

- distinct edges.
- same vertices and faces.

Generalized maps [Damiand and Lienhardt 2004]

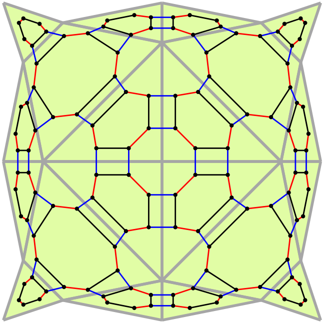


G-maps built as graphs.

2-arc :

- distinct faces.
- same vertices and edges.

Generalized maps [Damiand and Lienhardt 2004]

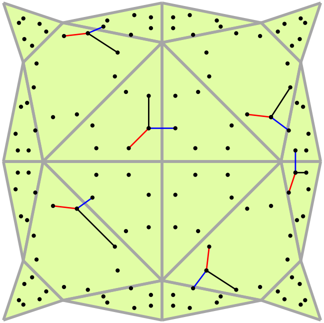


G-maps built as graphs.

n-G-map (union of the graphs)

Undirected graph labeled on the arcs with dimensions from $[\![0, n]\!]$ such that :

Generalized maps [Damiand and Lienhardt 2004]



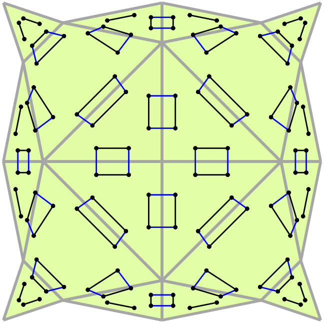
G-maps built as graphs.

n-G-map (union of the graphs)

Undirected graph labeled on the arcs with dimensions from $[\![0, n]\!]$ such that :

- Incidence : every dart (node) if the source of a unique arc per dimension.

Generalized maps [Damiand and Lienhardt 2004]



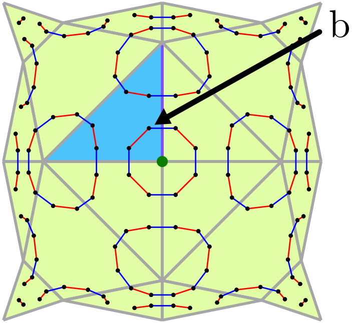
G-maps built as graphs.

n-G-map (union of the graphs)

Undirected graph labeled on the arcs with dimensions from $[\![0, n]\!]$ such that :

- Incidence : every dart (node) if the source of a unique arc per dimension.
- Cycles : any $ijij$-path is a cycle whenever $i + 2 \leq j$

Topological cells and orbits

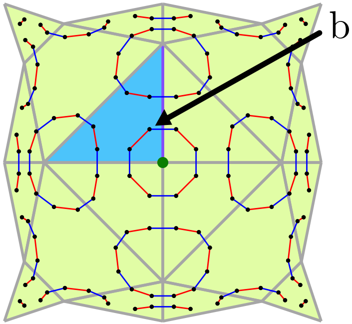


Topological cells correspond to subgraphs.

Orbit

Graph induced by a subset $\langle o \rangle \subseteq [\![0, n]\!]$ of dimensions.

Topological cells and orbits

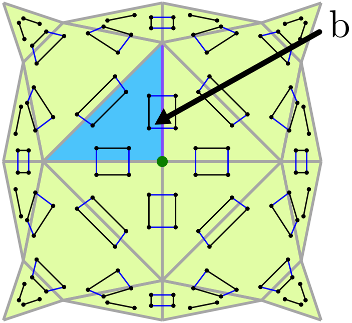


Topological cells correspond to subgraphs.

b belongs to the green vertex.

- $\langle 1, 2 \rangle$-orbit .

Topological cells and orbits

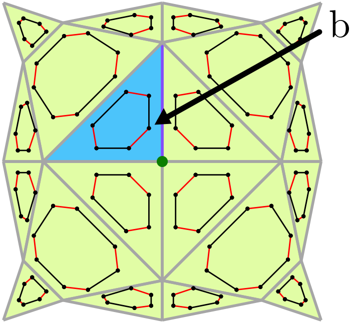


Topological cells correspond to subgraphs.

b belongs to the purple edge.

- $\langle 0, 2 \rangle$-orbit .

Topological cells and orbits



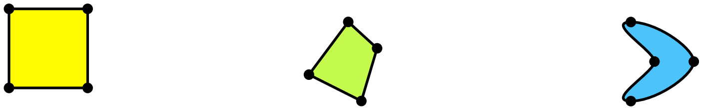
Topological cells correspond to subgraphs.

b belongs to the blue face.

- $\langle 0, 1 \rangle$-orbit .

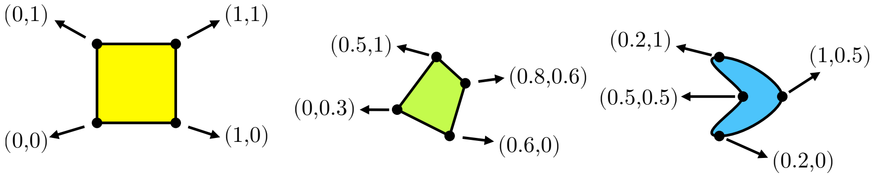
Geometry

- An object's display rely on it's geometry.



Geometry

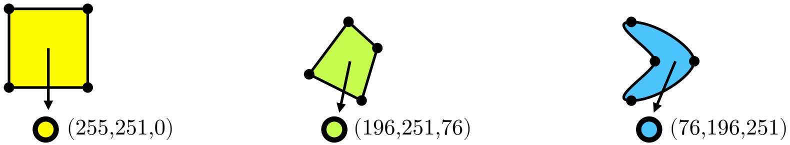
- An object's display rely on it's geometry.



An embedding is defined for an orbit type : position.

Geometry

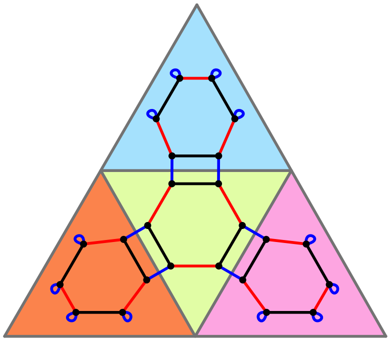
- An object's display rely on it's geometry.



An embedding is defined for an orbit type : color.

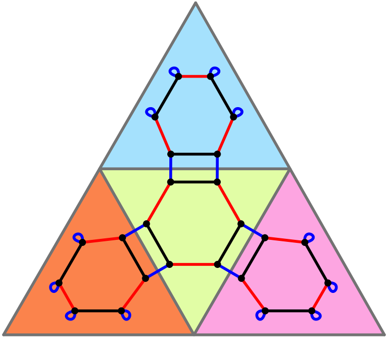
Embedded G-maps

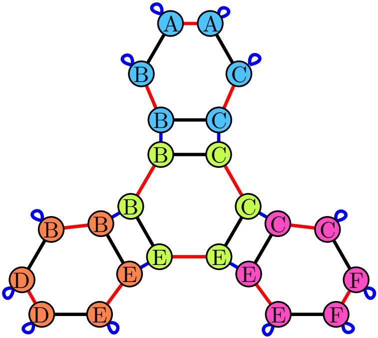
▶ Embedding values are defined for each dart, with some consistency constraints.



Embedded G-maps

► Embedding values are defined for each dart, with some consistency constraints.

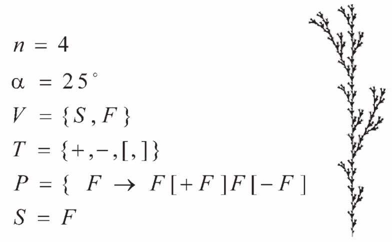




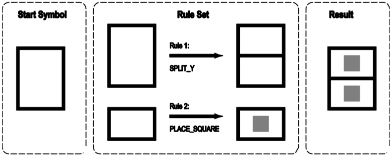
Graph rewriting

- Operations on G-maps are designed as graph rewriting rules.

Rewriting



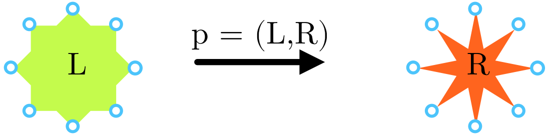
L-systems
([Santos et Coelho 2009])



Shape grammars
([Di Angelo et al. 2012])

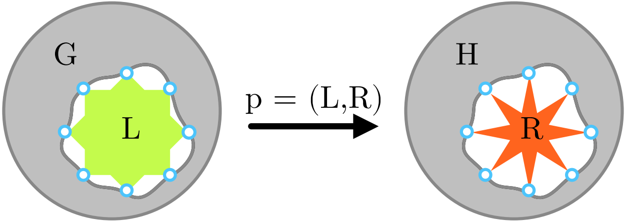
Graph transformation rules

- Goal : Generalise from strings to graphs.



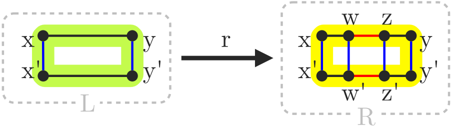
Graph transformation rules

- Goal : Generalise from strings to graphs.

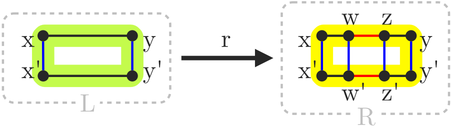


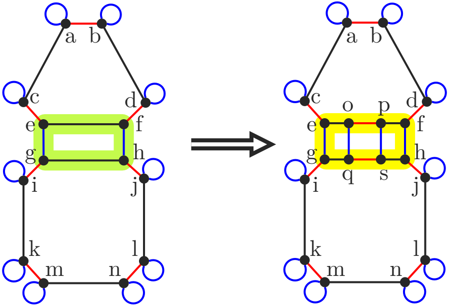
- Find in G a graph similar to L,
- Remove it from G,
- Reconnect R within the more global context.

G-map rewriting : vertex insertion

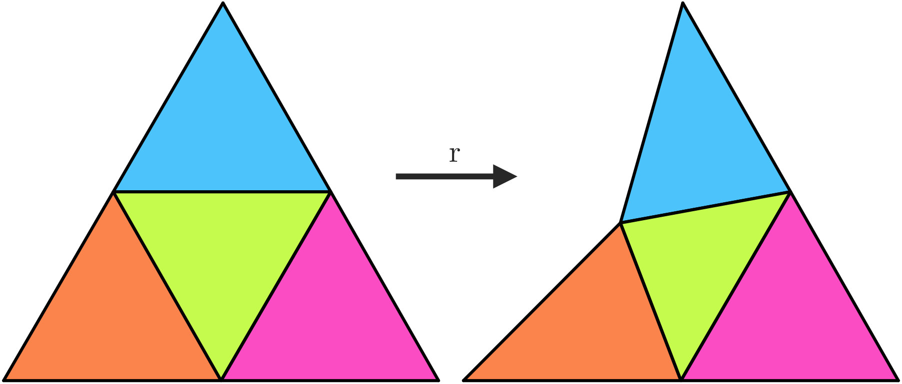


G-map rewriting : vertex insertion

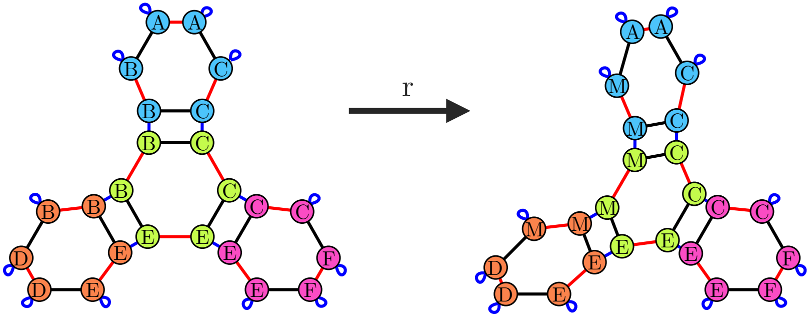




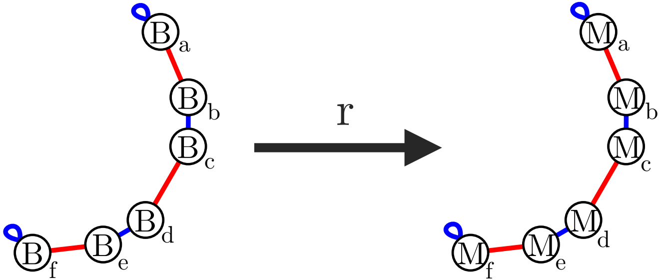
Embedding expressions : vertex translation



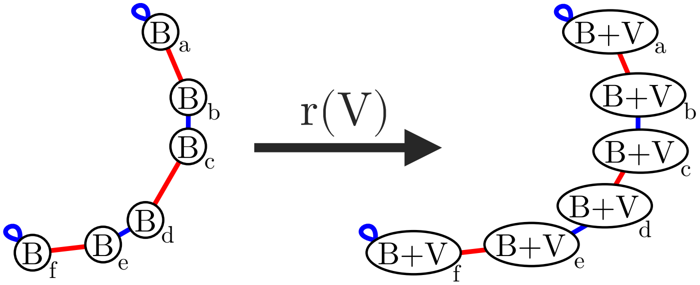
Embedding expressions : vertex translation



Embedding expressions : vertex translation

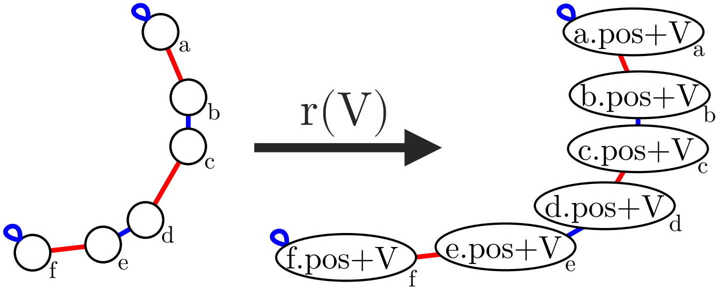


Embedding expressions : vertex translation



- Computations in the embedding space
  - $B + V = M$

Embedding expressions : vertex translation

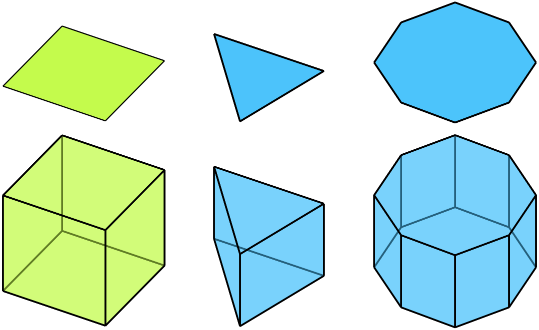


- Computations in the embedding space
- Value accessors
  - $a.position = B$

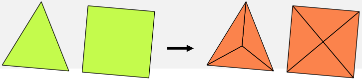
Need for genericity (1) : Topology

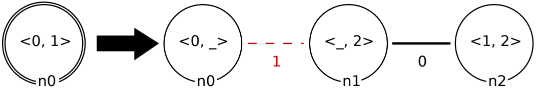
Exploit the homogeneity of G-maps.

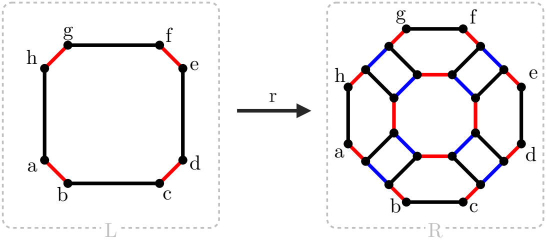
- Key idea : modeling operation are parameterized by the topology (more precisely orbits).



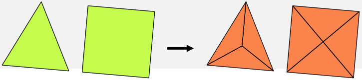
Orbit rewriting

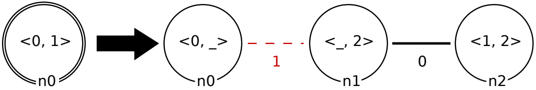


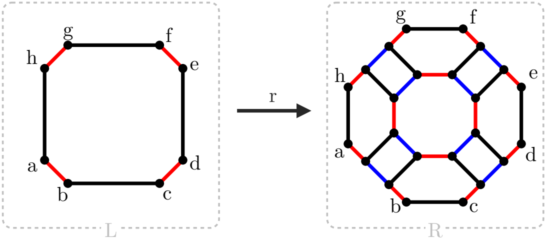




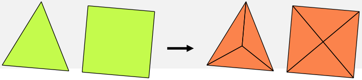
Orbit rewriting

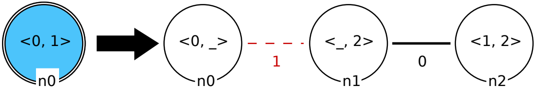


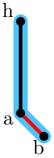


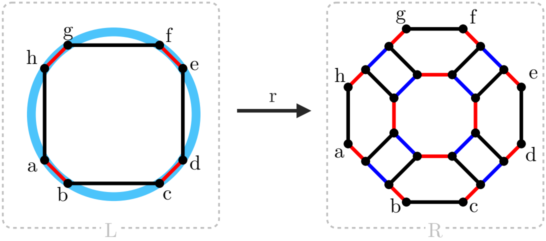


Orbit rewriting

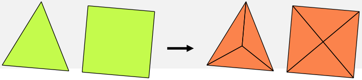


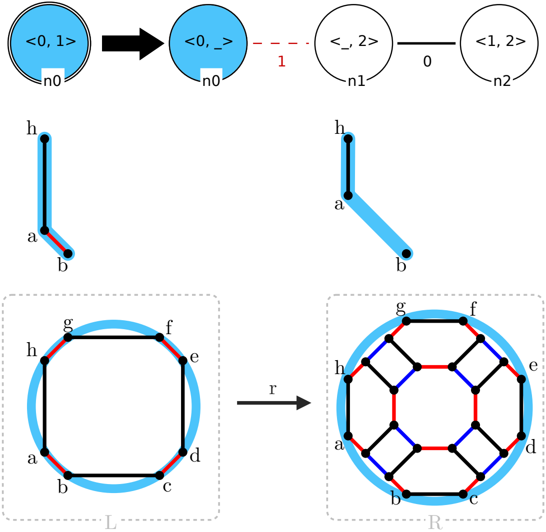




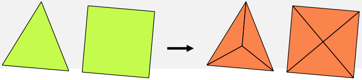


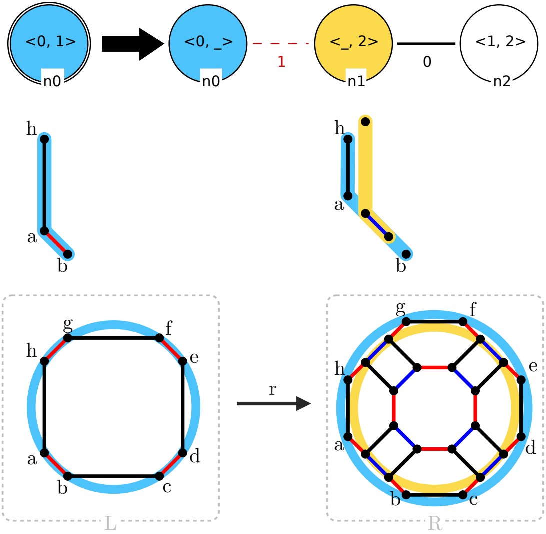
Orbit rewriting



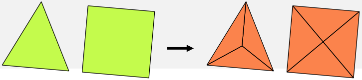


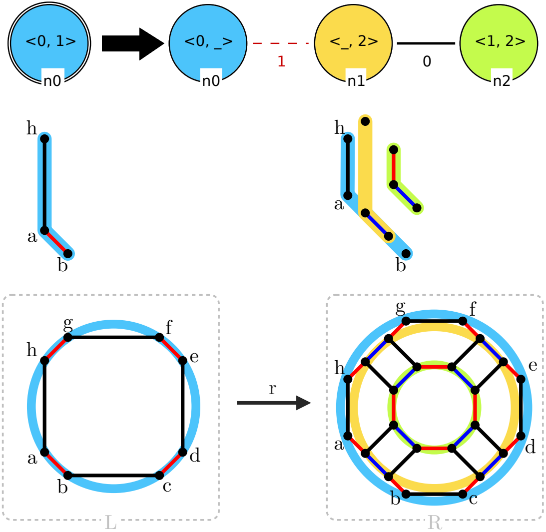
Orbit rewriting



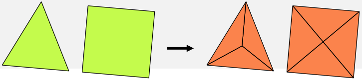


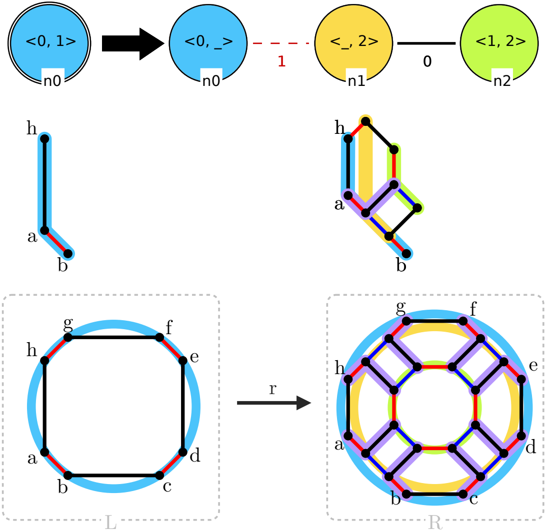
Orbit rewriting



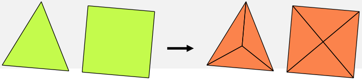


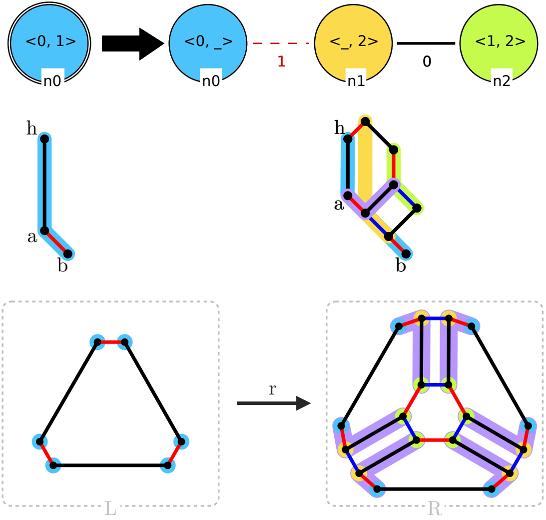
Orbit rewriting



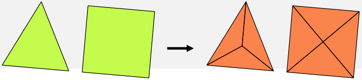


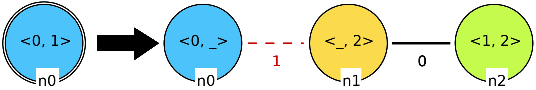
Orbit rewriting

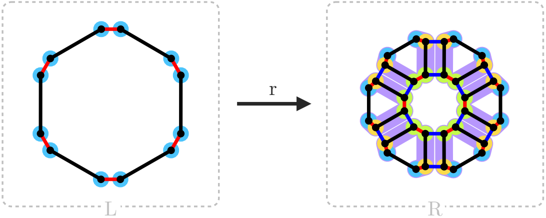




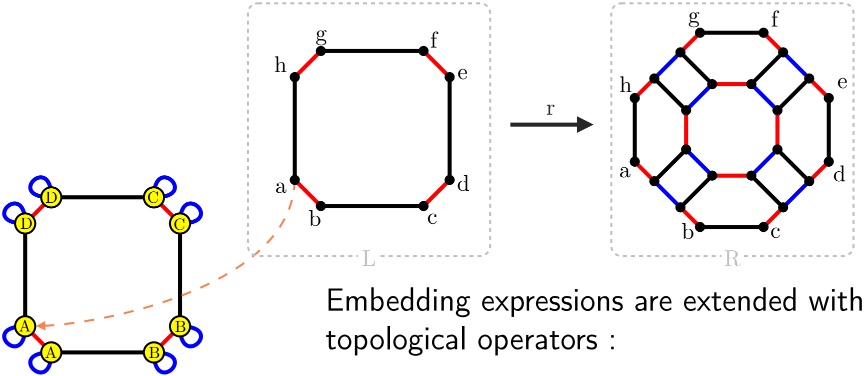
Orbit rewriting





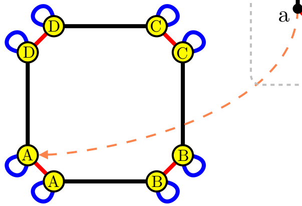


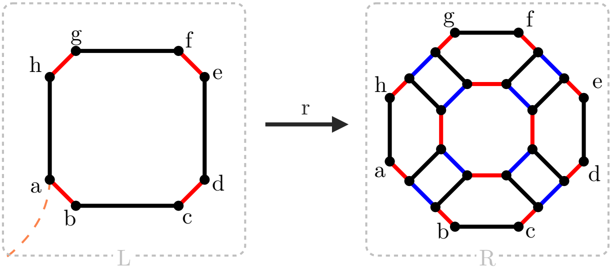
Need for genericity (2) : Geometry



Embedding expressions are extended with topological operators :

Need for genericity (2) : Geometry

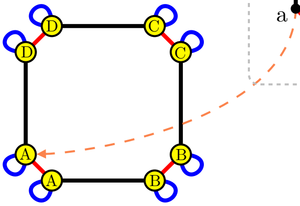


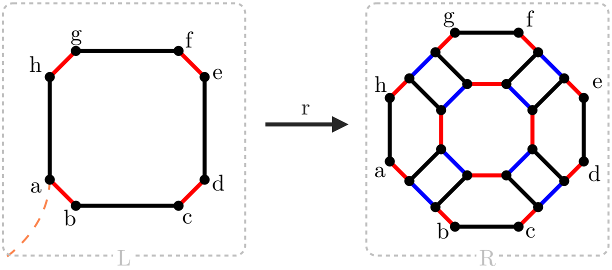


Embedding expressions are extended with topological operators :

- Neighbor operator :
  - a@0.position = D
  - a@0@1@0.position = C

Need for genericity (2) : Geometry

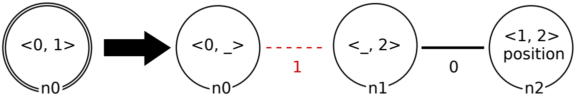


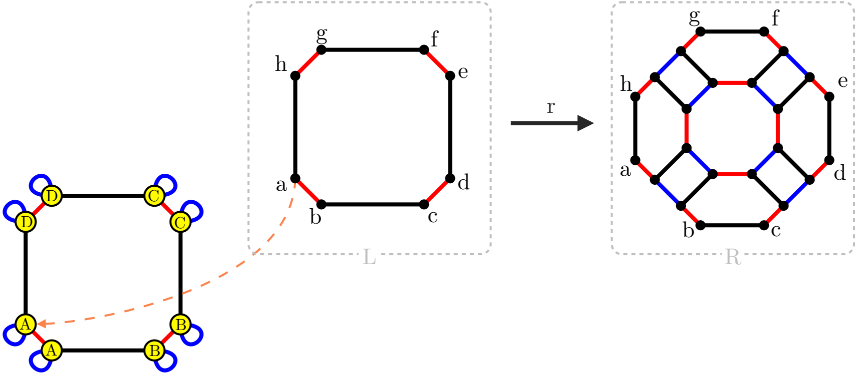


Embedding expressions are extended with topological operators :

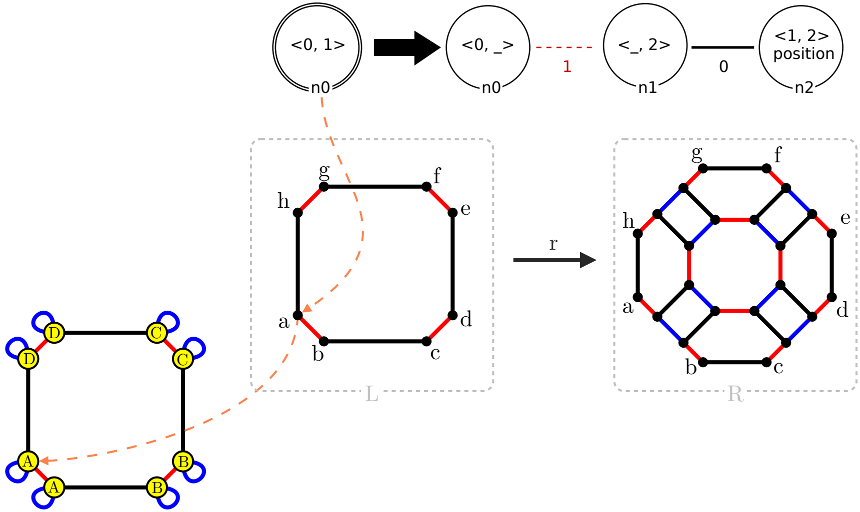
- Neighbor operator :
- Collect operator :
  - $position_{\langle 0,2 \rangle}(a) = \{A, D\}$
  - $position_{\langle 0,1 \rangle}(a) = \{A, B, C, D\}$

Extension to schemes

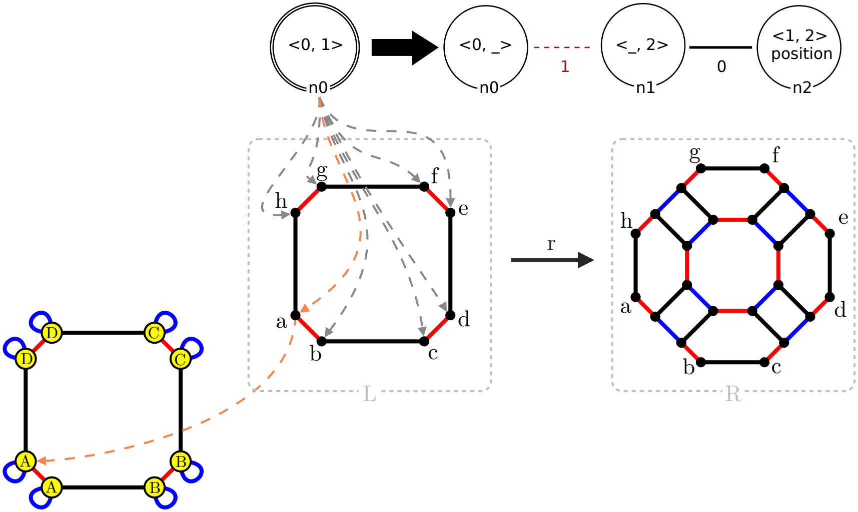




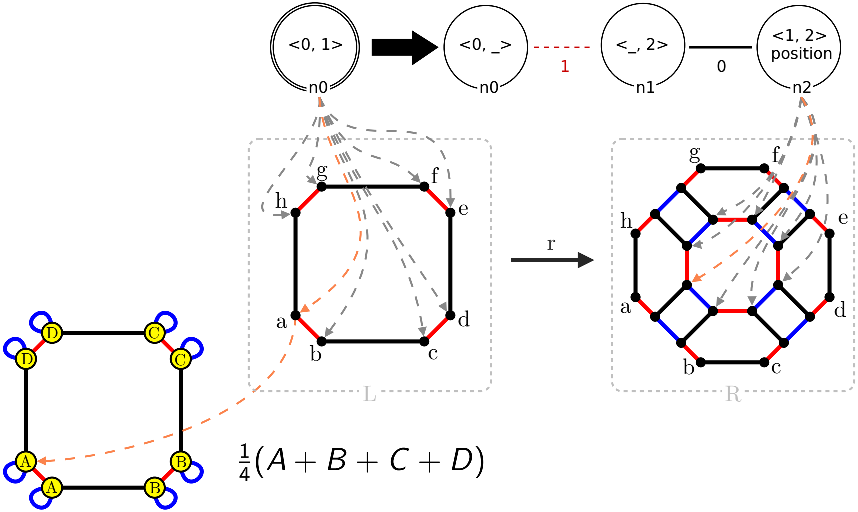
Extension to schemes



Extension to schemes

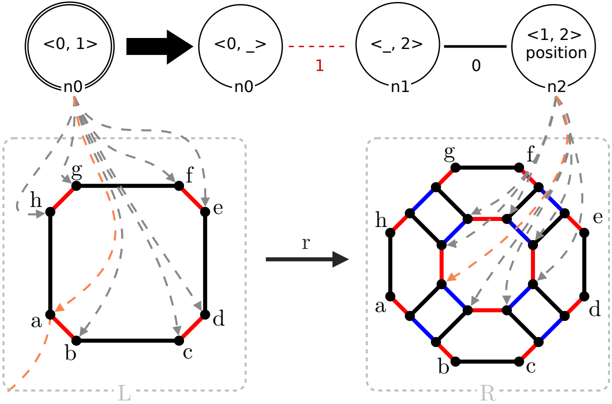


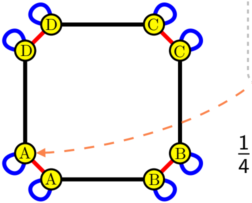
Extension to schemes



$\frac{1}{4}(A + B + C + D)$

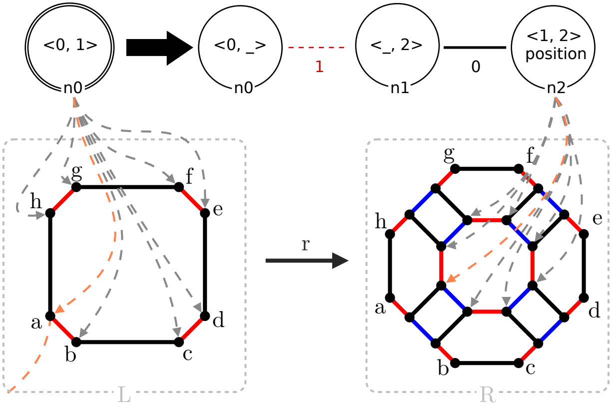
Extension to schemes

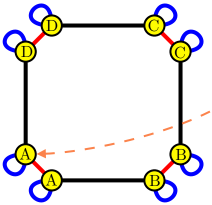




$$\frac{1}{4}(A + B + C + D) = middle(\{A, B, C, D\})$$

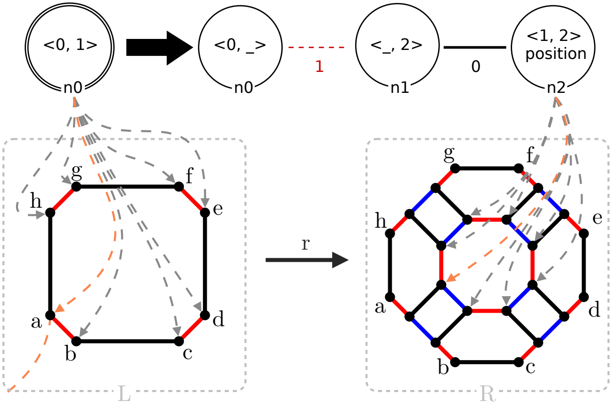
Extension to schemes

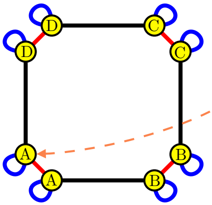




$$\frac{1}{4}(A + B + C + D) = middle(\{A, B, C, D\})$$
$$= middle(position_{\langle 0,1\rangle}(a))$$

Extension to schemes

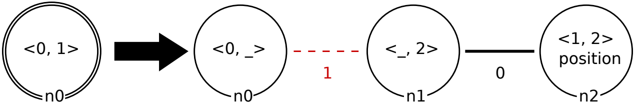




$$\frac{1}{4}(A+B+C+D) = middle(\{A,B,C,D\})$$
$$= middle(position_{\langle 0,1\rangle}(a))$$
$$= middle(position_{\langle 0,1\rangle}(n0))$$

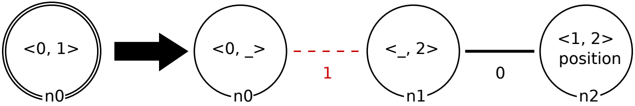
Key points

We describe geometric modeling operations with rule schemes.



Key points

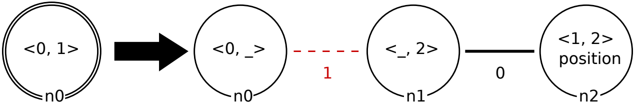
We describe geometric modeling operations with rule schemes.



- Operations described in a domain-specific language

Key points

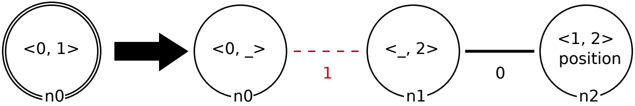
We describe geometric modeling operations with rule schemes.



- Operations described in a domain-specific language
- Each node in a pattern of a rule encodes for multiple darts in the G-maps.

Key points

We describe geometric modeling operations with rule schemes.



- Operations described in a domain-specific language
- Each node in a pattern of a rule encodes for multiple darts in the G-maps.
- Node names in the embedding expressions are substituted during the instantiation.

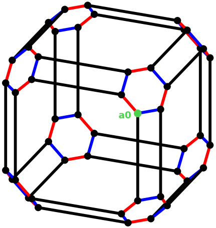
Topological inference

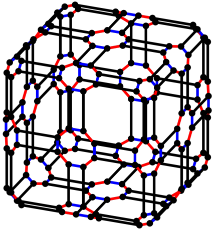
- Graph traversal algorithm to infer the topological part of modeling operations.

Objective

Retrieve a scheme rule from an instance of an object before modification and one after modification.

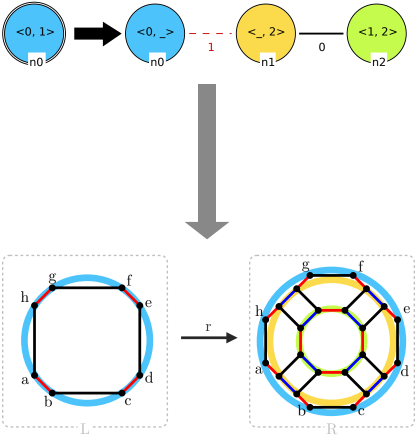
- Input : Before G-map, After G-map and some additional information.
- Output : Rule scheme(s) describing the modeling operation.



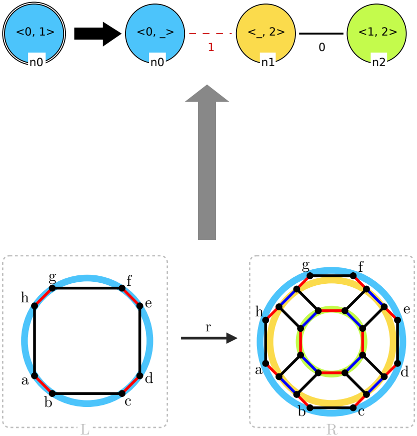


- Dart $a0$
- Orbit type $\langle 1,\ 2 \rangle$

Reversing the instantiation process



Reversing the instantiation process



Algorithm : Topological folding algorithm

Input: A graph G encoding the preservation relation between two partial G-maps, an orbit type $\langle o \rangle$, and a dart a of G.

Output: A graph $\mathcal{S}$ that encodes the Jerboa rule with $\langle o \rangle$ as variable, given that the operation is applied at the dart a.

$Q \leftarrow \emptyset,\ \mathcal{S} \leftarrow \emptyset$ // empty queue and empty ‘rule‘ graph
$h \leftarrow Node(G, \langle o \rangle, a)$ // build the hook node
$add\_node(\mathcal{S}, h)$ // add h to the ‘rule‘ graph
$enqueue(Q, h)$
while $Q \neq \emptyset$ **do**
  $m \leftarrow dequeue(Q)$
  foreach $d \in [\![0, n]\!] \setminus label(m)$ **do**
    $v \leftarrow arc\_expansion(G, m, d)$ // extend arcs
    $build\_label(G, v)$ // deduce the relabeling function
    $add\_node(\mathcal{S}, v)$
    $enqueue(Q, v)$
return S

R. Pascual, H. Belhaouari, A. Arnould, and P. Le Gall, ‘Inferring topological operations on generalized maps : Application to subdivision schemes’, Graphics and Visual Computing, 2022.

Folding a G-map

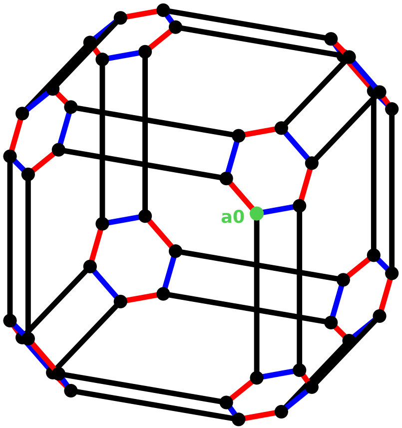
Besides a G-map, we whose a dart in the G-map and an orbit type.

Graph traversal algorithm

Iteratively apply two foldings :

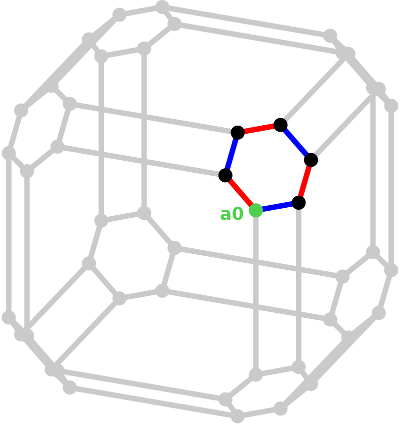
- Folding of a node.
- Folding of the arcs.

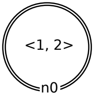
▶ Illustration on the cube with the orbit type $\langle 1, 2 \rangle$.



Execution

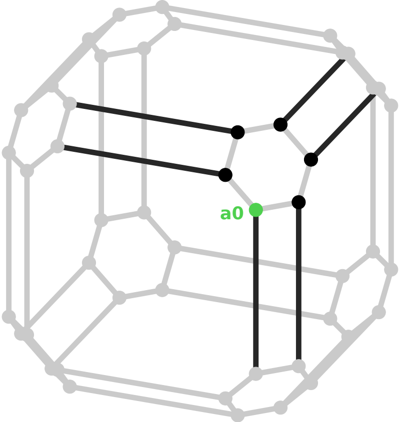
Folding of a node (hook case with the orbit type $\langle 1, 2 \rangle$).

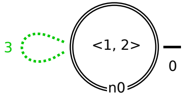




Execution

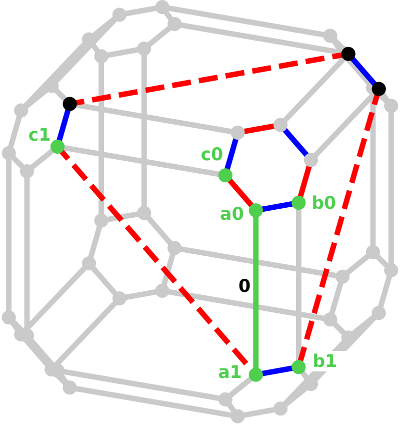
Folding of the arcs.

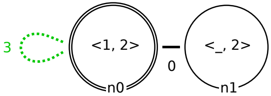




Execution

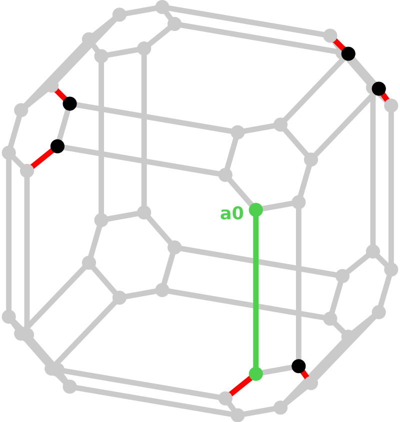
Folding of a node.

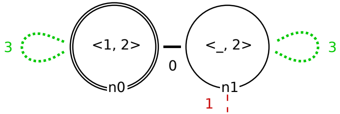




Execution

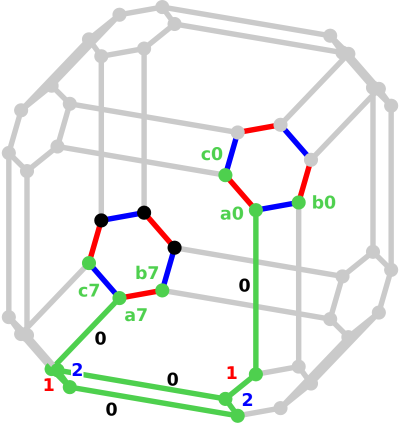
Folding of the arcs.

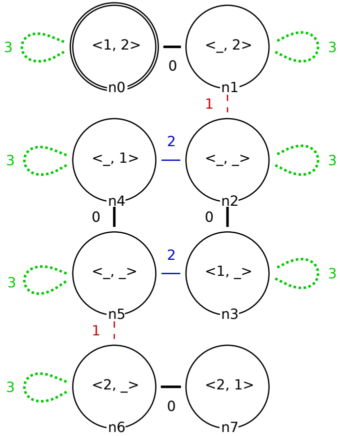




Execution

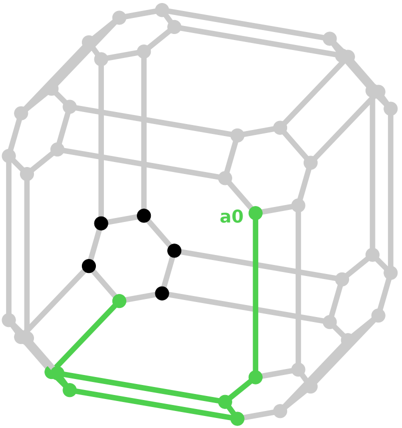
Folding of a node.

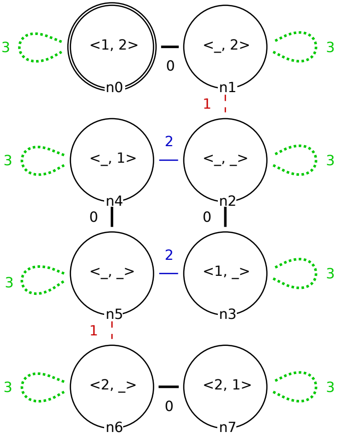




Execution

Folding of the arcs.

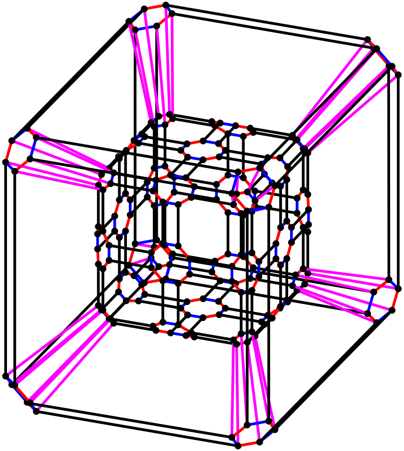




Folding of a rule

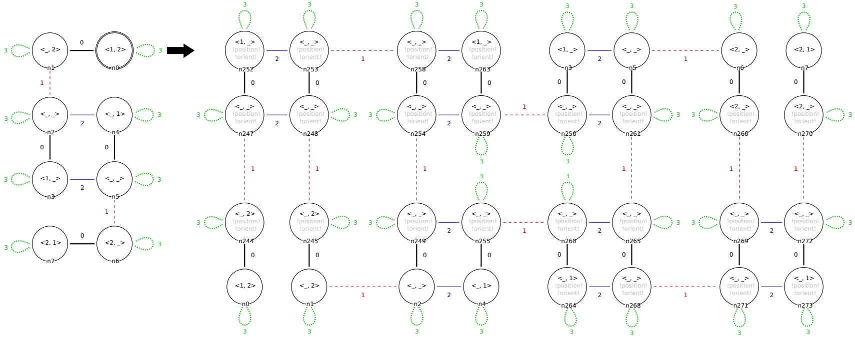
Partial mapping on darts from the before instance to the after instance.

Build a graph where the preserved darts are linked with κ-arcs (in pink).



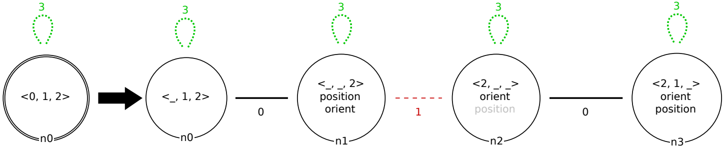
Folding the quad subdivision

- Rule scheme with the orbit type $\langle 1, 2 \rangle$ on the cube :

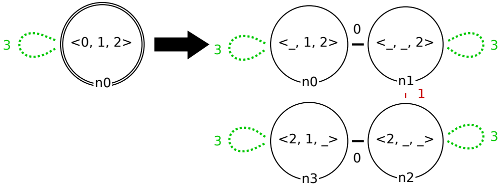


Folding the quad subdivision

- Used rule scheme :



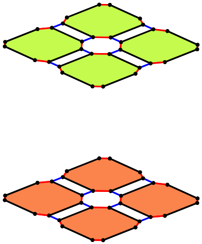
- Among the inferred rules, we retrieve the one we used.



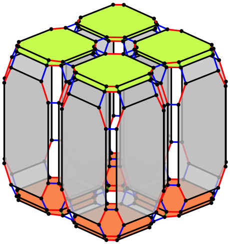
  - 768 possible schemes
  - 14 distinct schemes (cube symmetry).
  - 48 schemes tried (marking).
  - 14 schemes built (removal of isomorphic rules).

Example in geology (pure topology)

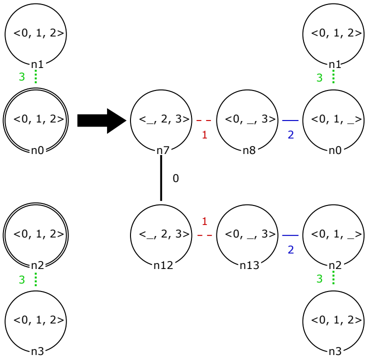
Before



After



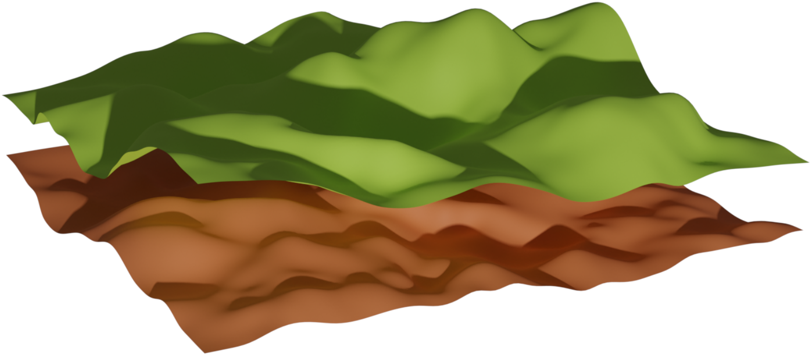
Operation



Inference time : ~ 3 ms

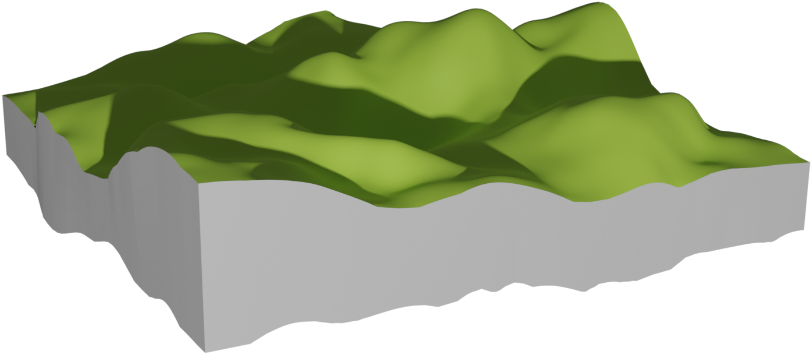
Example in geology (pure topology)

Before



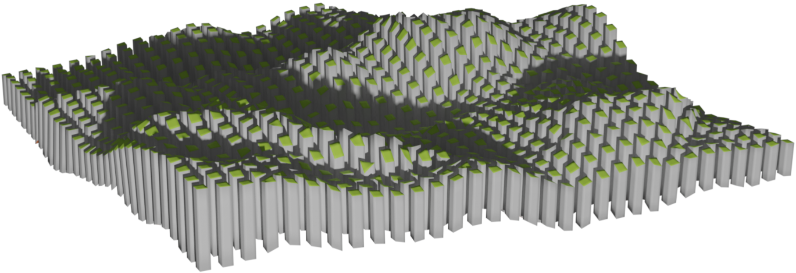
Example in geology (pure topology)

After



Example in geology (pure topology)

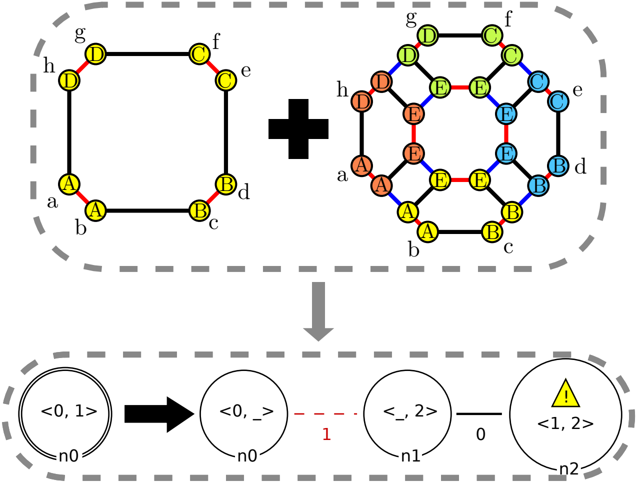
After



Geometric inference

- Computing embedding expressions is solved as a constraint solving problem.

Objective



The rule is missing its embedding expressions.

Method (inference of positions)

▶ Hypothesis : The vertex positions of the target object C are obtained as affine combinations of vertex positions in the initial object O.

Method (inference of positions)

▶ Hypothesis : The vertex positions of the target object C are obtained as affine combinations of vertex positions in the initial object O.

For each vertex in C, we want a position p expressed as :

$$p = \sum_{i=0}^{k} a_i p_i + t$$

where :

- p : target position (known)

Method (inference of positions)

▶ Hypothesis : The vertex positions of the target object C are obtained as affine combinations of vertex positions in the initial object O.

For each vertex in C, we want a position p expressed as :

$$p = \sum_{i=0}^{k} a_i p_i + t$$

where :

- p : target position (known)
- p_i : position of the initial vertex i (known)

Method (inference of positions)

▶ Hypothesis : The vertex positions of the target object C are obtained as affine combinations of vertex positions in the initial object O.

For each vertex in C, we want a position p expressed as :

$$p = \sum_{i=0}^{k} a_i p_i + t$$

where :

- p : target position (known)
- p_i : position of the initial vertex i (known)
- a_i : weight (unknown)

Method (inference of positions)

▶ Hypothesis : The vertex positions of the target object C are obtained as affine combinations of vertex positions in the initial object O.

For each vertex in C, we want a position p expressed as :

$$p = \sum_{i=0}^{k} a_i p_i + t$$

where :

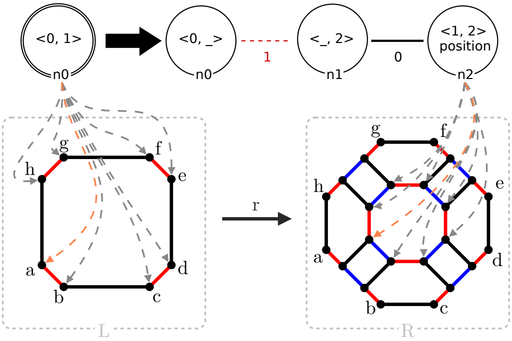
- p : target position (known)
- p_i : position of the initial vertex i (known)
- a_i : weight (unknown)
- t : translation (unknown)

Need for abstraction on schemes

We want $(a_i)_{0 \leq i \leq k}$ such that :
$p = \sum_{i=0}^{k} a_i p_i + t$

Need for abstraction on schemes

We want $(a_i)_{0 \leq i \leq k}$ such that :
$p = \sum_{i=0}^{k} a_i p_i + t$

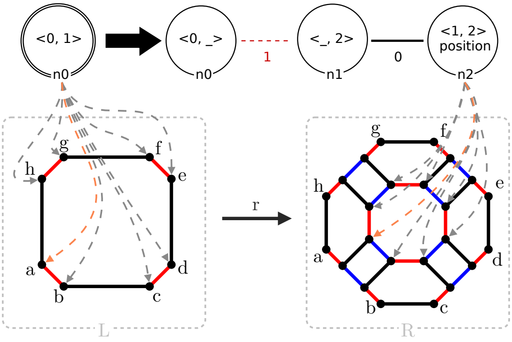


Issue : darts in the G-map will share the same expression.

- Because rule schemes abstract topological cells.

Need for abstraction on schemes

We want $(a_i)_{0 \leq i \leq k}$ such that :
$p = \sum_{i=0}^{k} a_i p_i + t$



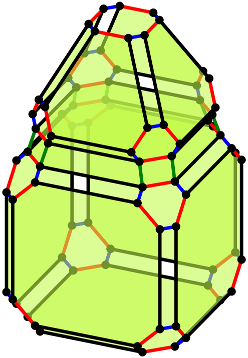
Issue : darts in the G-map will share the same expression.

- Because rule schemes abstract topological cells.

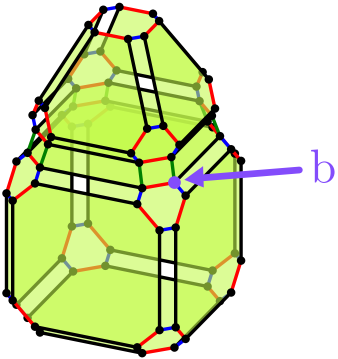
Solution : Exploit the topology.

- Use points of interests that share the same expression.

Points of interests



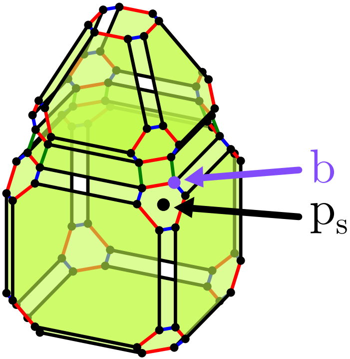
Points of interests



Points of interests

avec

- p_s : vertex

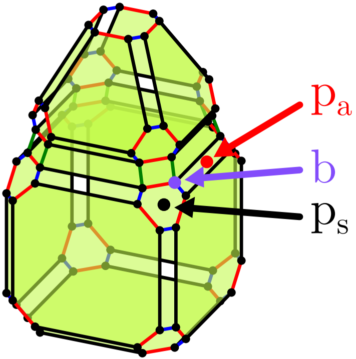


$$p_s = middle(position_{\langle 1,2,3 \rangle}(b))$$

Points of interests

avec

- p_s : vertex
- p_a : edge midpoint

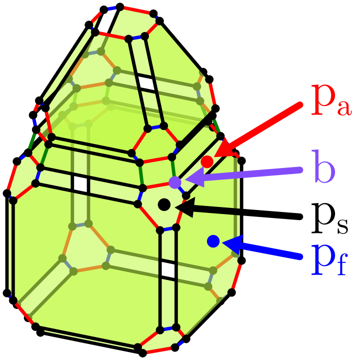


$$p_a = middle(position_{\langle 0,2,3 \rangle}(b))$$

Points of interests

avec

- p_s : vertex
- p_a : edge midpoint
- p_f : face barycenter

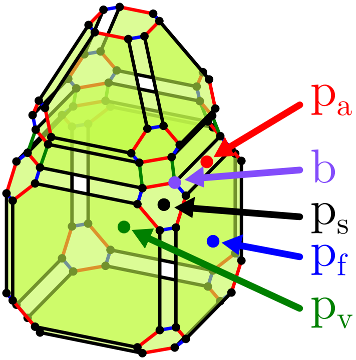


$$p_f = middle(position_{\langle 0,1,3 \rangle}(b))$$

Points of interests

avec

- p_s : vertex
- p_a : edge midpoint
- p_f : face barycenter
- p_v : volume barycenter

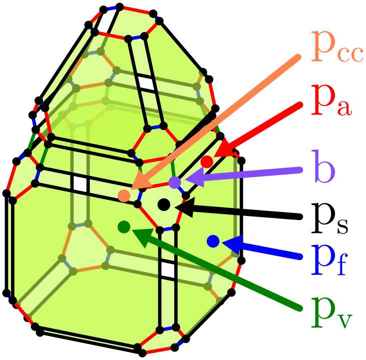


$$p_v = middle(position_{\langle 0,1,2\rangle}(b))$$

Points of interests

avec

- p_s : vertex
- p_a : edge midpoint
- p_f : face barycenter
- p_v : volume barycenter
- p_{cc} : CC barycenter

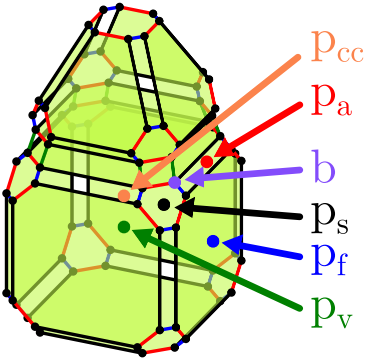


$$p_{cc} = middle(position_{\langle 0,1,2,3\rangle}(b))$$

Points of interests

avec

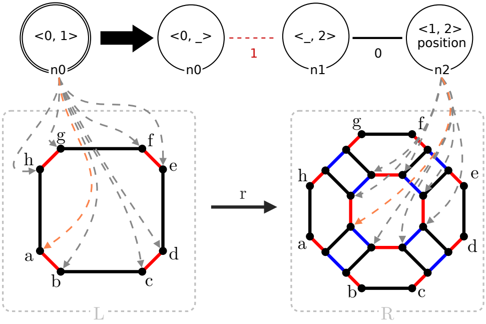
- p_s : vertex
- p_a : edge midpoint
- p_f : face barycenter
- p_v : volume barycenter
- p_{cc} : CC barycenter



Thanks to the points of interests, the systems is rewritten as :

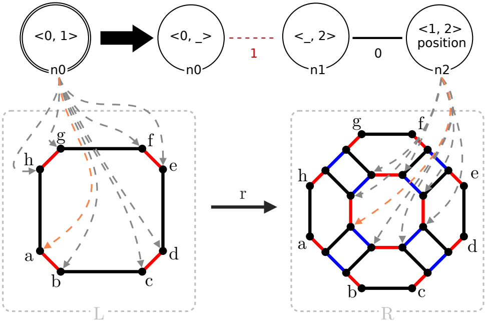
$$p = a_s p_s + a_a p_a + a_f p_f + a_v p_v + a_{cc} p_{cc} + t$$

Illustration



The position expression of $n2$ only depends of $n0$.

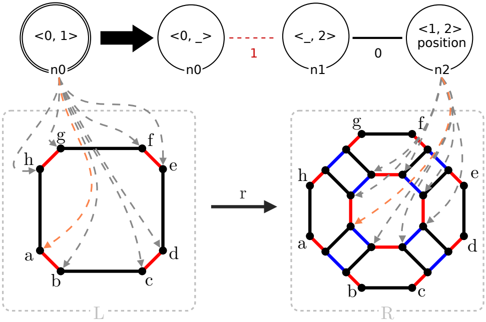
Illustration



The position expression of $n2$ only depends of $n0$.

$$n2.position = a_s n0.p_s + a_a n0.p_a + a_f n0.p_f + a_v n0.p_v + a_{cc} n0.p_{cc} + t$$

Illustration

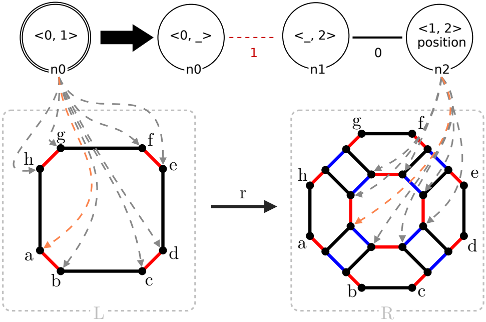


The position expression of $n2$ only depends of $n0$.

- One equation per dart (8 darts.)

$$n2.position = a_s n0.p_s + a_a n0.p_a + a_f n0.p_f + a_v n0.p_v + a_{cc} n0.p_{cc} + t$$

Illustration

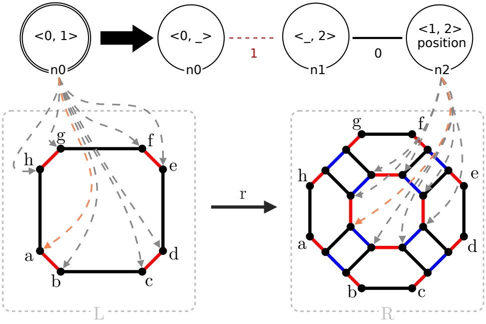


The position expression of $n2$ only depends of $n0$.

- One equation per dart (8 darts.)
- Split per coordinate (on x, y, z).

$$n2.position = a_s n0.p_s + a_a n0.p_a + a_f n0.p_f + a_v n0.p_v + a_{cc} n0.p_{cc} + t$$

Illustration

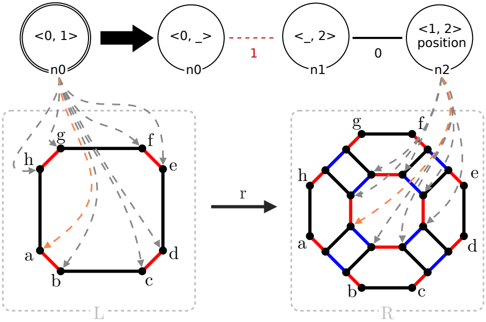


The position expression of $n2$ only depends of $n0$.

- One equation per dart (8 darts.)
- Split per coordinate (on x, y, z).
- 24 equations and 8 variables.

$$n2.position = a_s n0.p_s + a_a n0.p_a + a_f n0.p_f + a_v n0.p_v + a_{cc} n0.p_{cc} + t$$

Illustration



The position expression of *n*2 only depends of *n*0.

- One equation per dart (8 darts.)
- Split per coordinate (on *x*, *y*, *z*).
- 24 equations and 8 variables.

$$n2.position = a_s n0.p_s + a_a n0.p_a + a_f n0.p_f + a_v n0.p_v + a_{cc} n0.p_{cc} + t$$

▶ Solved as a CSP (OR-Tools, Z3)

Solving the barycentric triangulation

- Global equation :

$n2.position = a_s n0.p_s + a_a n0.p_a + a_f n0.p_f + a_v n0.p_v + a_{cc} n0.p_{cc} + t$

Solving the barycentric triangulation

- Global equation :

$$n2.position = a_s n0.p_s + a_a n0.p_a + a_f n0.p_f + a_v n0.p_v + a_{cc} n0.p_{cc} + t$$

- Generated system (only on x and y)

$$\begin{cases}
(0.5;0.5) = a_s*(0;0) + a_a*(0.5;0) + a_f*(0.5;0.5) + a_v*(0.5;0.5) + a_{cc}*(0.5;0.5) + (tx;ty) \\
(0.5;0.5) = a_s*(1;0) + a_a*(0.5;0) + a_f*(0.5;0.5) + a_v*(0.5;0.5) + a_{cc}*(0.5;0.5) + (tx;ty) \\
(0.5;0.5) = a_s*(1;0) + a_a*(1;0.5) + a_f*(0.5;0.5) + a_v*(0.5;0.5) + a_{cc}*(0.5;0.5) + (tx;ty) \\
(0.5;0.5) = a_s*(1;1) + a_a*(1;0.5) + a_f*(0.5;0.5) + a_v*(0.5;0.5) + a_{cc}*(0.5;0.5) + (tx;ty) \\
(0.5;0.5) = a_s*(1;1) + a_a*(0.5;1) + a_f*(0.5;0.5) + a_v*(0.5;0.5) + a_{cc}*(0.5;0.5) + (tx;ty) \\
(0.5;0.5) = a_s*(0;1) + a_a*(0.5;1) + a_f*(0.5;0.5) + a_v*(0.5;0.5) + a_{cc}*(0.5;0.5) + (tx;ty) \\
(0.5;0.5) = a_s*(0;1) + a_a*(0;0.5) + a_f*(0.5;0.5) + a_v*(0.5;0.5) + a_{cc}*(0.5;0.5) + (tx;ty) \\
(0.5;0.5) = a_s*(0;0) + a_a*(0;0.5) + a_f*(0.5;0.5) + a_v*(0.5;0.5) + a_{cc}*(0.5;0.5) + (tx;ty)
\end{cases}$$

Solving the barycentric triangulation

- Global equation :

$n2.position = a_s n0.p_s + a_a n0.p_a + a_f n0.p_f + a_v n0.p_v + a_{cc} n0.p_{cc} + t$

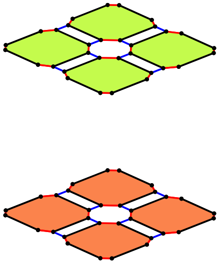
- Generated system (only on x and y)

$$\begin{cases} (0.5;0.5) = a_s*(0;0) + a_a*(0.5;0) + a_f*(0.5;0.5) + a_v*(0.5;0.5) + a_{cc}*(0.5;0.5) + (tx;ty) \\ (0.5;0.5) = a_s*(1;0) + a_a*(0.5;0) + a_f*(0.5;0.5) + a_v*(0.5;0.5) + a_{cc}*(0.5;0.5) + (tx;ty) \\ (0.5;0.5) = a_s*(1;0) + a_a*(1;0.5) + a_f*(0.5;0.5) + a_v*(0.5;0.5) + a_{cc}*(0.5;0.5) + (tx;ty) \\ (0.5;0.5) = a_s*(1;1) + a_a*(1;0.5) + a_f*(0.5;0.5) + a_v*(0.5;0.5) + a_{cc}*(0.5;0.5) + (tx;ty) \\ (0.5;0.5) = a_s*(1;1) + a_a*(0.5;1) + a_f*(0.5;0.5) + a_v*(0.5;0.5) + a_{cc}*(0.5;0.5) + (tx;ty) \\ (0.5;0.5) = a_s*(0;1) + a_a*(0.5;1) + a_f*(0.5;0.5) + a_v*(0.5;0.5) + a_{cc}*(0.5;0.5) + (tx;ty) \\ (0.5;0.5) = a_s*(0;1) + a_a*(0;0.5) + a_f*(0.5;0.5) + a_v*(0.5;0.5) + a_{cc}*(0.5;0.5) + (tx;ty) \\ (0.5;0.5) = a_s*(0;0) + a_a*(0;0.5) + a_f*(0.5;0.5) + a_v*(0.5;0.5) + a_{cc}*(0.5;0.5) + (tx;ty) \end{cases}$$

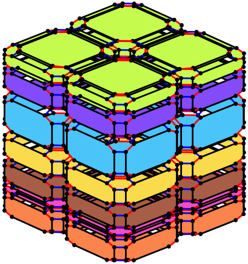
- Solution found :
  - $a_s = -6.601425600620388E-17$
  - $a_a = 0.0$
  - $a_f = 1.0$
  - $a_v = 0.0$
  - $a_{cc} = 0.0$
  - $t = (0.0, 0.0)$

Example in geology (with the geometry)

Before

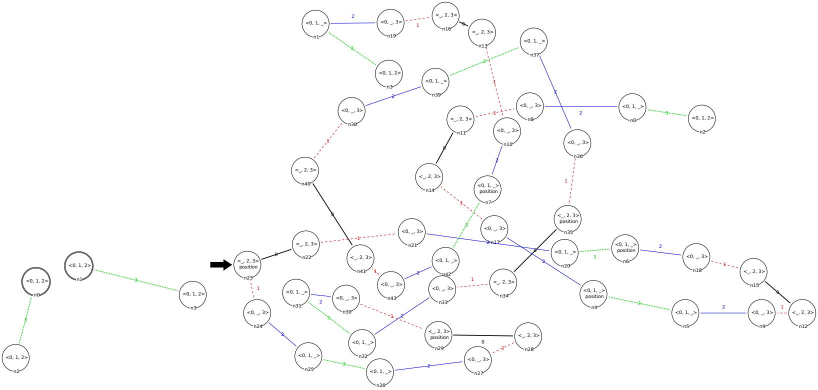


After



Example in geology (with the geometry)

Operation



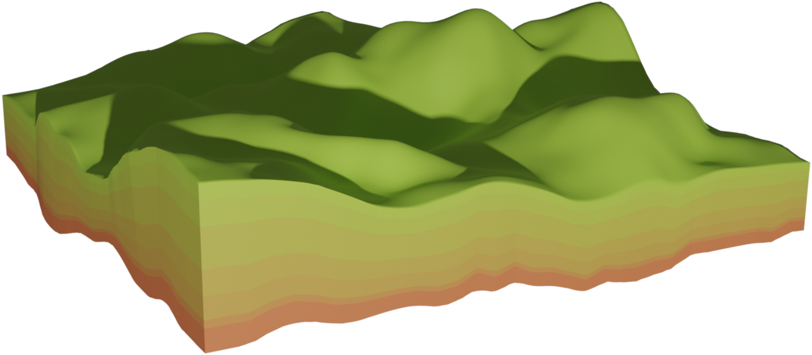
Inference time : $\sim$ 26 ms for the topology,
$\sim$ 549 ms for the embedding expressions

Example in geology (with the geometry)

Before

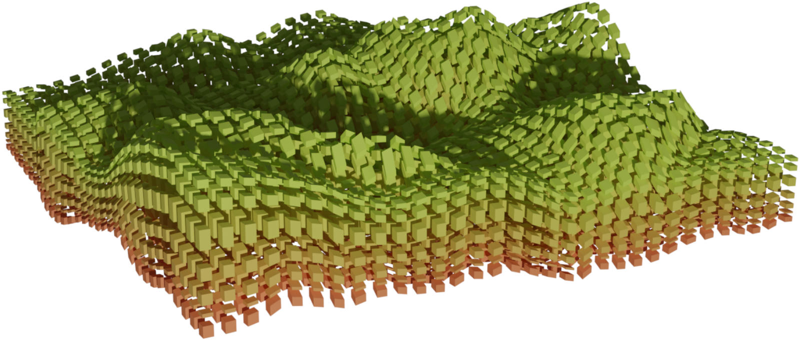
Example in geology (with the geometry)

After



Example in geology (with the geometry)

After

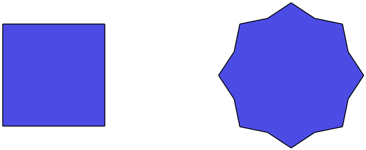


Limits

- 1. The solution does not admit any solution.

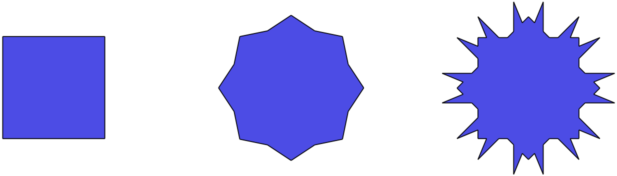
Limits

- 1. The solution does not admit any solution.
- 2. We do not find the desired solution.



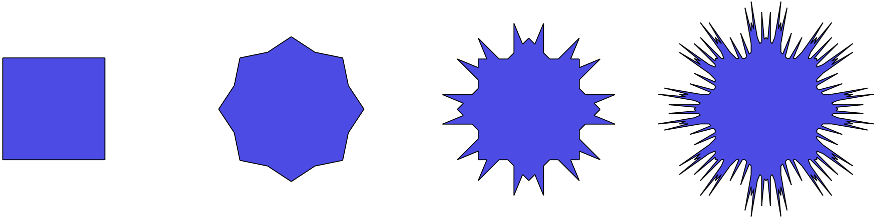
Limits

- 1. The solution does not admit any solution.
- 2. We do not find the desired solution.



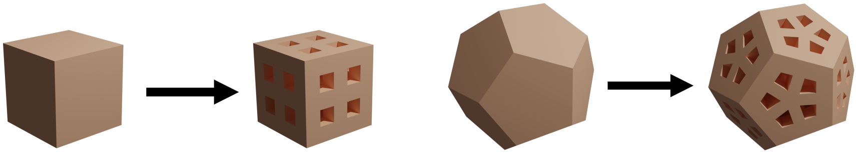
Limits

- 1. The solution does not admit any solution.
- 2. We do not find the desired solution.



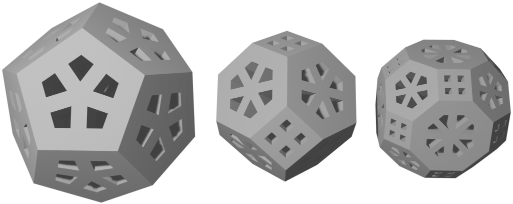
Conclusion

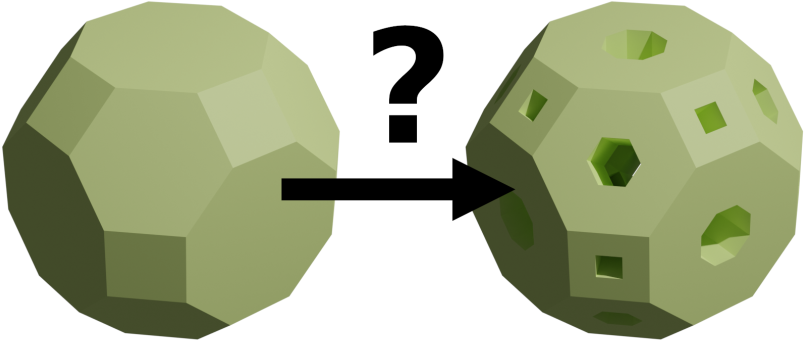
- We presented a method to infer a geometric modeling operation from two instances of an object before and after modification.
- Topology : Graph traversal algorithm.
- Geometry : Affine combinations of points of interests.

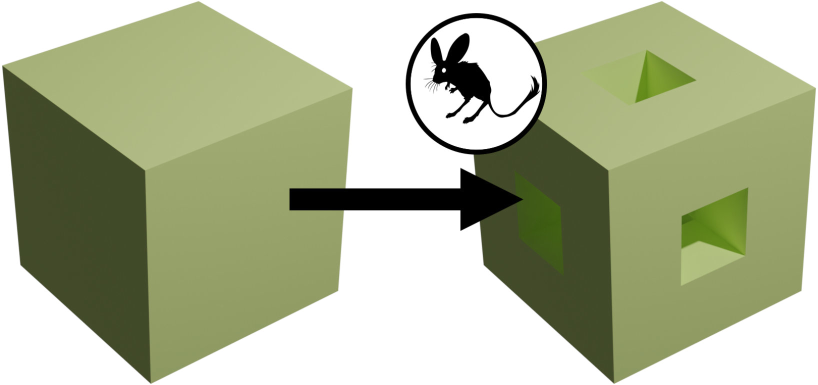


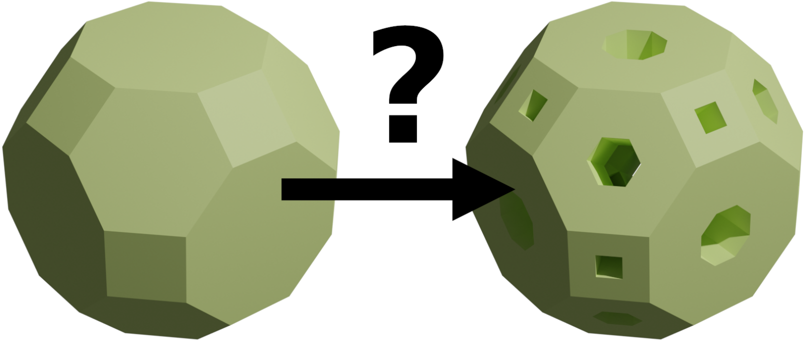
Future works

- Geometric inference :
  - Other points of interest (exploiting the neighboring operator).
  - Other kind of functions (instead of affine combinations).
  - Other embeddings.
- Support in the design of operations
  - Automated generation of instances for a given operation.

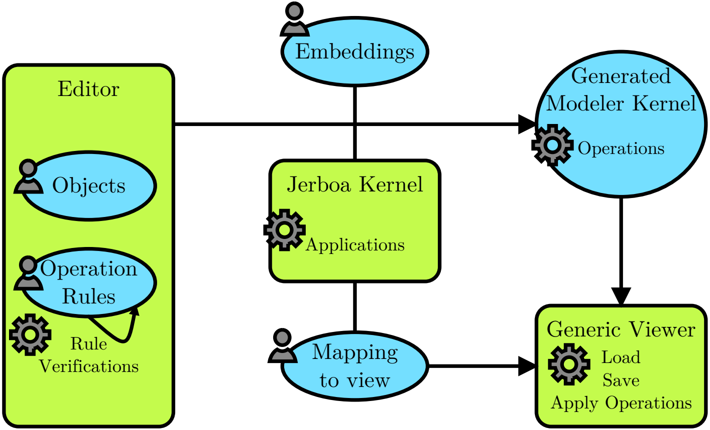








Jerboa's architecture



How to preserve the consistency of the model ?

- Topological constraints : structure,
  - E.g., vertices are incident to edges thats are incident to faces.
- Embedding constraints : geometry,
  - E.g., all elements defining the same vertex should share the same position.

Goal : The modification of a well-formed object should provide a well-formed object.

Rewriting

String rewriting :

- an alphabet Σ
- a set of rewriting rules $u \rightarrow v$ (u and v are words on Σ^*)

Rewriting

String rewriting :

- an alphabet Σ
- a set of rewriting rules $u \rightarrow v$ (u and v are words on Σ^*)

sleep $\rightarrow$ follow

Rewriting

String rewriting :

- an alphabet Σ
- a set of rewriting rules $u \rightarrow v$ (u and v are words on Σ^*)

$$\text{sleep} \rightarrow \text{follow}$$

You␣are␣all␣sleeping␣!

Rewriting

String rewriting :

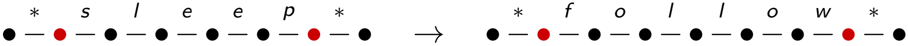
- an alphabet Σ
- a set of rewriting rules $u \to v$ (u and v are words on Σ^*)

sleep $\to$ follow

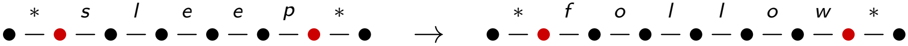
You␣are␣all␣sleeping␣! $\to$ You␣are␣all␣following␣!

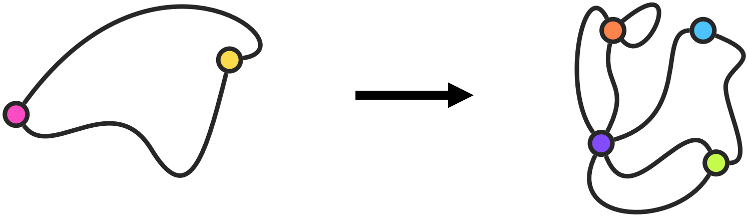
How to rewrite graphs ?

How to rewrite graphs ? (based on [Ehrig 1979])

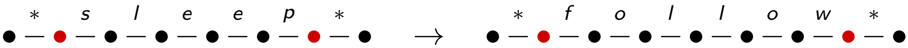


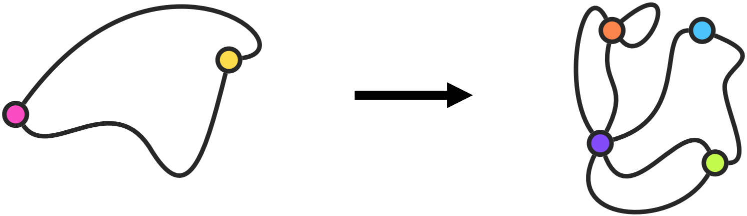
How to rewrite graphs ? (based on [Ehrig 1979])





How to rewrite graphs ? (based on [Ehrig 1979])



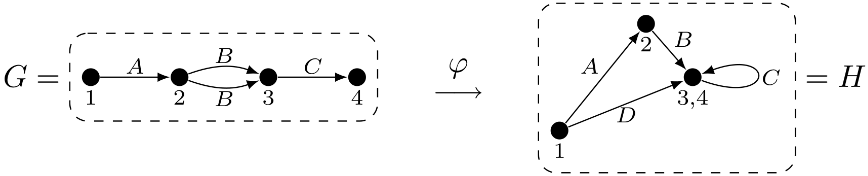


- No notion of beginning and end in a graph.

→ Identify the "gluing" elements.

How to map graphs ? (from [König 18])

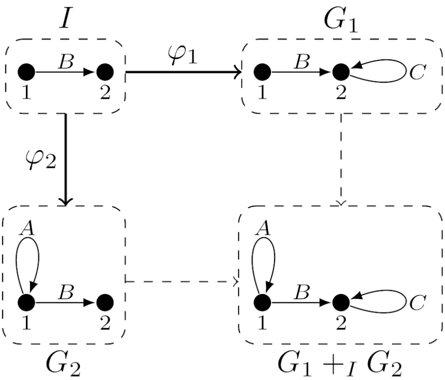
Graph morphism :



- Functions on nodes ands arcs that preserve structure.

How to glue graphs ? (from [König 18])

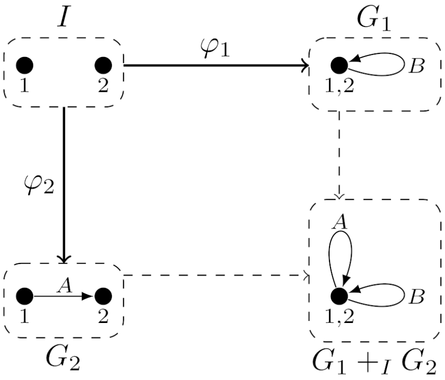
Graph gluing :



- $\sim$ Quotiented disjoint union.

How to glue graphs ? (from [König 18])

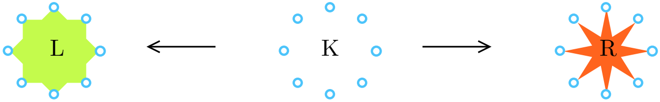
Graph gluing :



- $\sim$ Quotiented disjoint union.

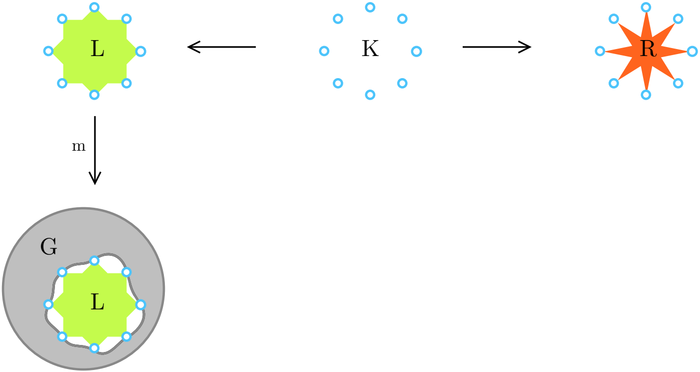
Back to graph transformation rules

- L, K, R, G, D, H are graphs.
- Arrows are graph morphismses.
- Squares are graph gluings.



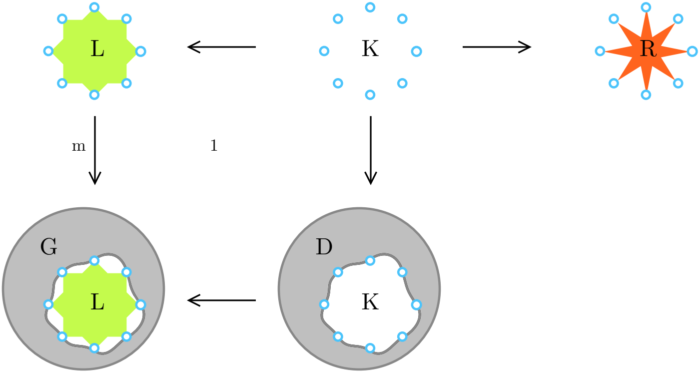
Back to graph transformation rules

- L, K, R, G, D, H are graphs.
- Arrows are graph morphismses.
- Squares are graph gluings.



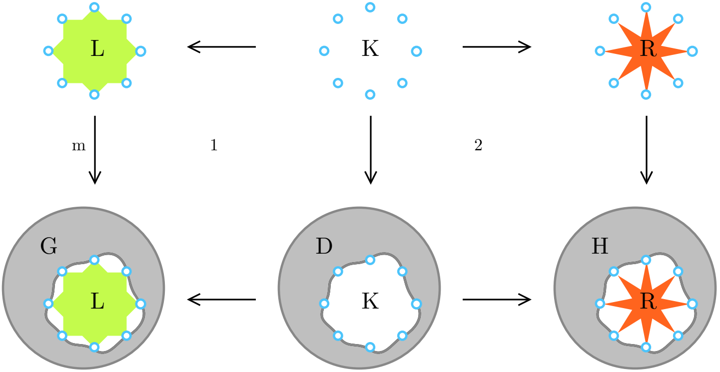
Back to graph transformation rules

- L, K, R, G, D, H are graphs.
- Arrows are graph morphismses.
- Squares are graph gluings.

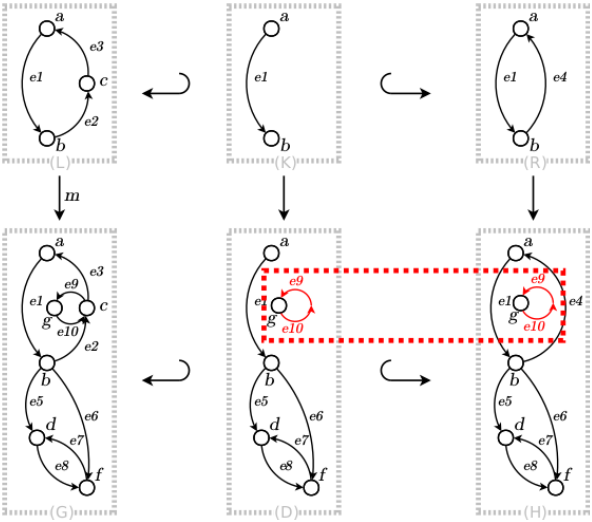


Back to graph transformation rules

- L, K, R, G, D, H are graphs.
- Arrows are graph morphismses.
- Squares are graph gluings.



Why does it has to be so complicated ?



Rewriting G-maps

- Most conservative framework (all morphisms are injectives).