Category Theory 101
Graph Transformations
Discrete Structures Day

F. Prost

PLUME team, LIP - ENS-Lyon

17th of December 2015

Introduction

- High level approach to programming: graph rewriting based on category theory.
- Much more difficult than term rewriting (which are just trees).

Introduction

- High level approach to programming: graph rewriting based on category theory.
- Much more difficult than term rewriting (which are just trees).
- Simulation of biological phenomenons.
- Simulation of chemical reactions.
- Study of cloning:

  $\Longrightarrow$ Typically to produce a web site one starts to copy an existing one, then one modifies it accordingly to its will.

  $\Longrightarrow$ Social Data Anonymization techniques rely on finely tuned cloning operations.

Plan

Plan

Category Theory

- Early 40's by MacLane and Eilenberg with a unifying aim: topology and algebra.

$\Longrightarrow$ What are the fundamental structures of those two fields ?

Category Theory

- Early 40's by MacLane and Eilenberg with a unifying aim: topology and algebra.

$\Longrightarrow$ What are the fundamental structures of those two fields ?

- Results much more general than thought at first.
- Set theory is just a special case of category (Lawvere).
- In computer science E. Moggi was able to capture ideas previously thought to be outside of reach with the monads.
- In logic J.-Y. Girard and the linear logic.
- etc.

Category Theory

Definition

A category $\mathcal{C}$ is made of

- A collection of object : $Obj(\mathcal{C})$
- $\forall x, y \in Obj(\mathcal{C})$ a set $Hom_{\mathcal{C}}(x, y)$
- $\forall x \in Obj(\mathcal{C})$ there is $id_x \in Hom_{\mathcal{C}}(x, x)$
- $\forall x, y, z \in Obj(\mathcal{C})$ a function $\circ : Hom_{\mathcal{C}}(x, y) \times Hom_{\mathcal{C}}(y, z) \to Hom_{\mathcal{C}}(y, z)$

Category Theory

Definition

A category $\mathcal{C}$ is made of

- A collection of object : $Obj(\mathcal{C})$
- $\forall x, y \in Obj(\mathcal{C})$ a set $Hom_{\mathcal{C}}(x, y)$
- $\forall x \in Obj(\mathcal{C})$ there is $id_x \in Hom_{\mathcal{C}}(x, x)$
- $\forall x, y, z \in Obj(\mathcal{C})$ a function
  $\circ : Hom_{\mathcal{C}}(x, y) \times Hom_{\mathcal{C}}(y, z) \rightarrow Hom_{\mathcal{C}}(y, z)$

such that

1. Identity: $f \circ id = id \circ f = f$
2. Associativity: $(h \circ g) \circ f = h \circ (g \circ f)$

Example: Category of graphs

- Objects: $G = (V, E, s, t)$ with $s, t : E \to V$
- Morphisms: $f : G \to H$ must respect source and target functions, ie:

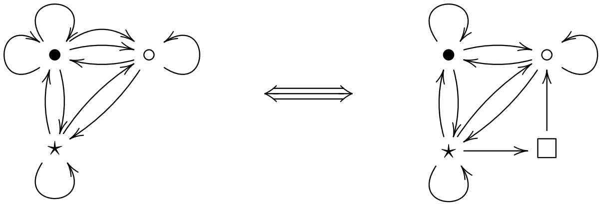
$$\forall e \in E. f(s(e)) = s(f(e))$$
$$\forall e \in E. f(t(e)) = t(f(e))$$

Example: Category of graphs

- Objects: $G = (V, E, s, t)$ with $s, t : E \to V$
- Morphisms: $f : G \to H$ must respect source and target functions, ie:

$$\forall e \in E.f(s(e)) = s(f(e))$$
$$\forall e \in E.f(t(e)) = t(f(e))$$

- Exemples:



Pullback

- Lets have : $f : X \to Z$ and $g : Y \to Z$
- Fiber product: $X \times_Z Y := \{(x, w, y) \mid f(x) = w = g(y)\}$

$$
\begin{tikzcd}
A \arrow[dr, dashed, "\exists !"] \arrow[drr, bend left, "\forall t"] \arrow[ddr, bend right, "\forall u"'] & & \\
& X \times_Z Y \arrow[r, dashed, "\pi_1"] \arrow[d, dashed, "\pi_2"'] & X \arrow[d, "f"] \\
& Y \arrow[r, "g"'] & Z
\end{tikzcd}
$$

Pushout

- Co-construction of the pullback.
- Lets have : $f : X \to Z$ and $g : Y \to Z$
- disjoint sum with gluing: $X +_Z Y := X + Y + Z/ \sim$
- With $\sim$ generated by $f(z) \sim z \sim g(z)$

$$\begin{tikzcd}
Z \arrow[r, "f"] \arrow[d, "g"'] & X \arrow[d, dashed, "i_1"] \arrow[ddr, bend left, "\forall u"] & \\
Y \arrow[r, dashed, "i_2"'] \arrow[drr, bend right, "\forall t"'] & X +_Z Y \arrow[dr, dashed, "\exists!"] & \\
 & & A
\end{tikzcd}$$

Plan

Rule based transformations

- Rule-based term rewriting is easy: replace a tree by another one.
- Much more difficult with graphs (multiple incident edges).
- Categorical frameworks make it clean to express graph transformations systematically.

PB	PO
clone	merge
delete	add
comatch	match
global	local

AGREE extended rule

Extension of a framework proposed by A. Corradini, D. Duval, R. Echahed, F. Prost and L. Ribeiro [ICGT15].

Definition (AGREE rules and matches)

- A *rule* is

$$\begin{tikzcd} L & K \arrow[l, "l"'] \arrow[r, "r"] \arrow[d, "t"] & R \\ T_L & T_K \arrow[l, "l'"'] & \end{tikzcd}$$

- A *match* of such a rule is composed of a mono $L \overset{m}{\rightarrowtail} G$ and a typing morphism $G \xrightarrow{\overline{m}} T_L$ and is such that $l' \circ t = (\overline{m} \circ m) \circ l$.

AGREE rewrite step

Definition (AGREE rewriting)

Given $\rho = (K \xrightarrow{l} L, K \xrightarrow{r} R, K \overset{t}{\rightarrowtail} T_K, T_K \xrightarrow{l'} T_L)$ and a match $L \overset{m}{\rightarrowtail} G, G \xrightarrow{\overline{m}} T_L$: $G \Rightarrow_{\rho,m} H$ is computed as follows:

1. Span $G \xleftarrow{g} D \xrightarrow{n'} T_K$ is the pullback of $G \xrightarrow{\overline{m}} T(L) \xleftarrow{l'} T_K$. Since $l' \circ t = \eta_L \circ l$ there is a unique $K \overset{n}{\rightarrowtail} D$.
2. $R \xrightarrow{p} H \xleftarrow{h} D$ is the pushout of $D \xleftarrow{n} K \xrightarrow{r} R$.

AGREE rewrite step

Definition (AGREE rewriting)

Given $\rho = (K \xrightarrow{l} L, K \xrightarrow{r} R, K \xrightarrowtail{t} T_K, T_K \xrightarrow{l'} T_L)$ and a match $L \xrightarrowtail{m} G, G \xrightarrow{\overline{m}} T_L : G \Rightarrow_{\rho,m} H$ is computed as follows:

1. Span $G \xleftarrow{g} D \xrightarrow{n'} T_K$ is the pullback of $G \xrightarrow{\overline{m}} T(L) \xleftarrow{l'} T_K$. Since $l' \circ t = \eta_L \circ l$ there is a unique $K \xrightarrowtail{n} D$.
2. $R \xrightarrow{p} H \xleftarrow{h} D$ is the pushout of $D \xleftarrow{n} K \xrightarrow{r} R$.

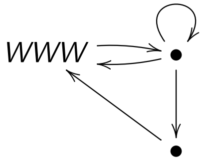
$$
\begin{tikzcd}
L \arrow[d, "m"] & K \arrow[l, tail, "l"'] \arrow[r, "r"] \arrow[d, "n"'] \arrow[dd, bend left, "t"] \arrow[dr, phantom, "PO\ (3)"] & R \arrow[d, "p"] \\
G \arrow[d, "\overline{m}"] \arrow[dr, phantom, "PB\ (1)"] & D \arrow[l, "g"'] \arrow[r, "h"] \arrow[d, "n'"] \arrow[ul, phantom, "(2)"] & H \\
T(L) & T_K \arrow[l, "l'"] &
\end{tikzcd}
$$

Example : copy of web pages

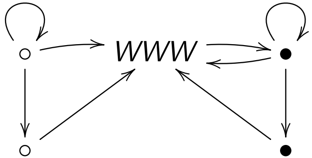
- The structure of a web site typically as two kind of links :
  - Internal links: file hierarchy (indirect link)
  - External links: references pointing outside of the site.

Example : copy of web pages

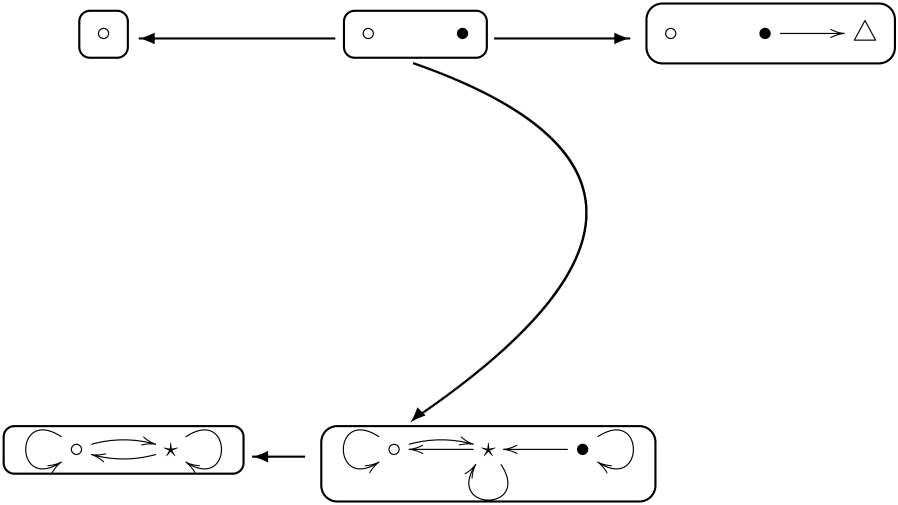
- The structure of a web site typically as two kind of links :
  - Internal links: file hierarchy (indirect link)
  - External links: references pointing outside of the site.
- The cloning of a web site consists in duplicating all local files and keeping external links shared between the two copies.



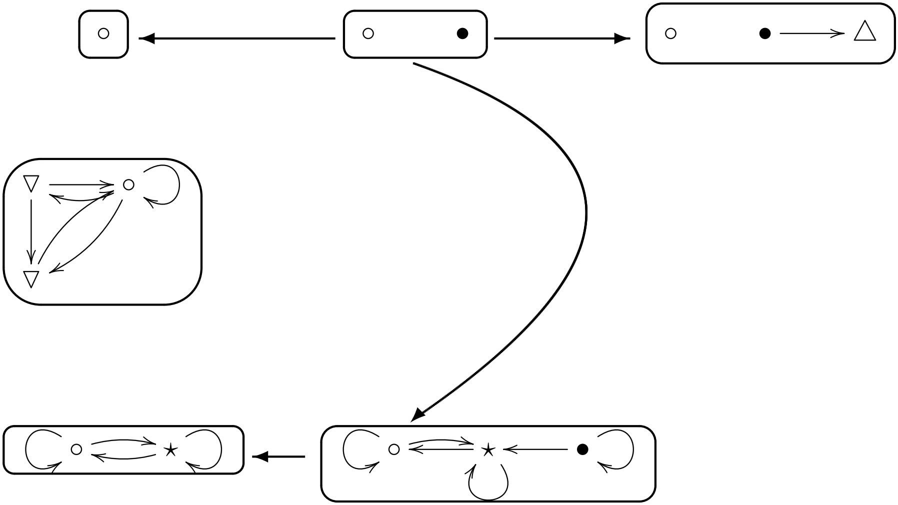
should be cloned as follows



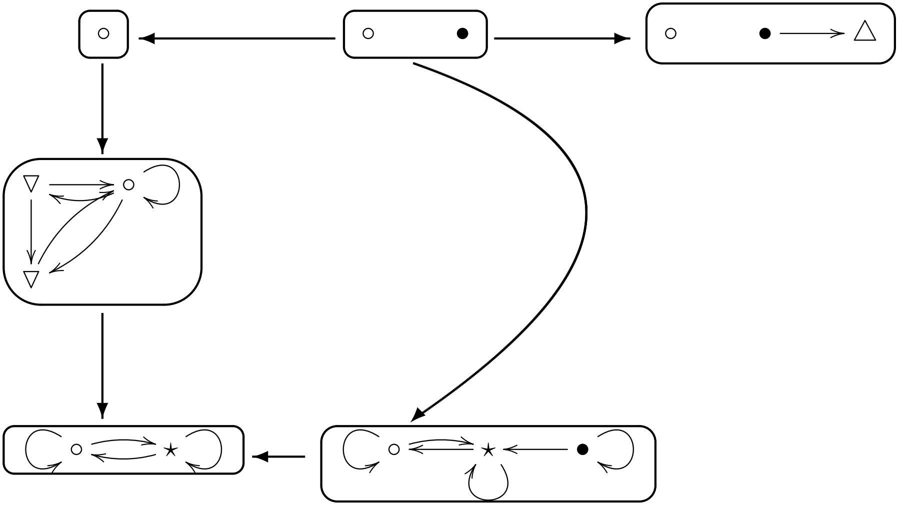
Web copy with AGREE rewriting



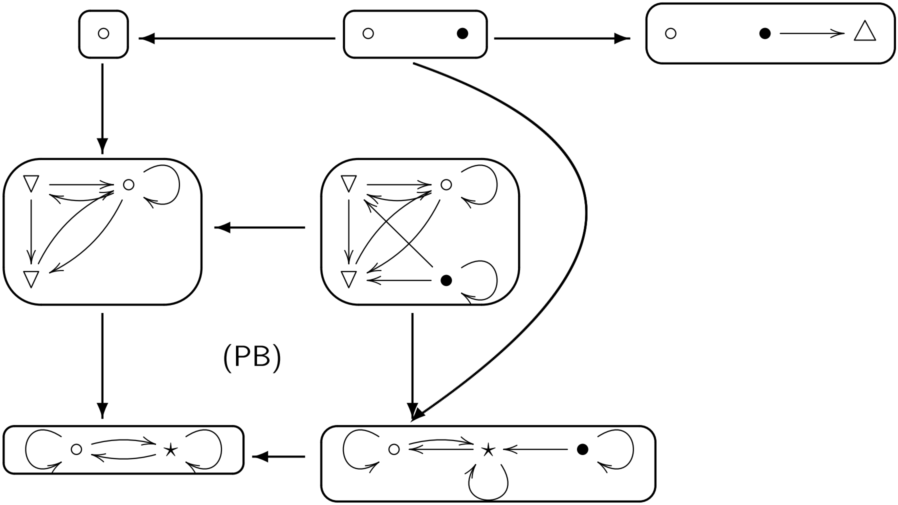
Web copy with AGREE rewriting



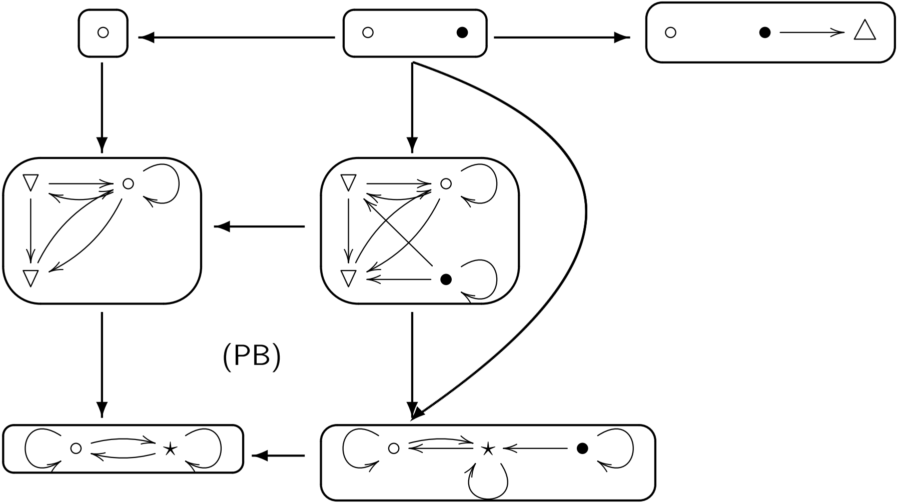
Web copy with AGREE rewriting



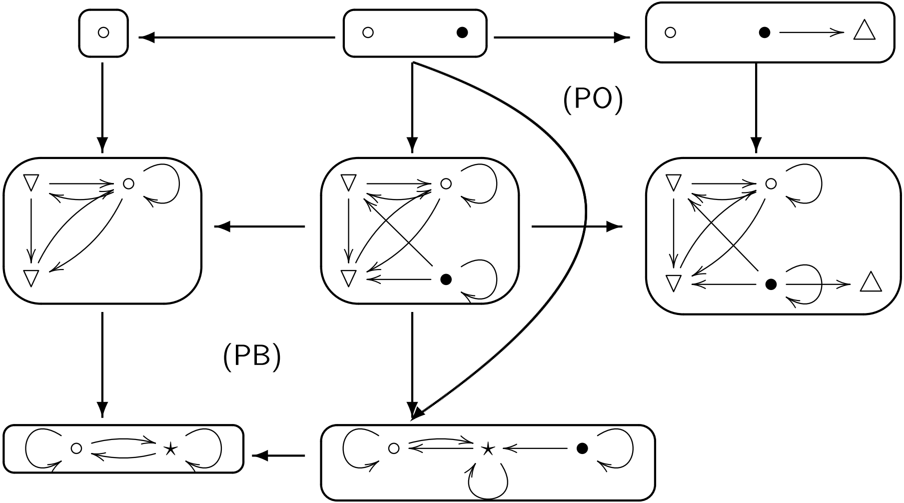
Web copy with AGREE rewriting



Web copy with AGREE rewriting



Web copy with AGREE rewriting



Plan

Social Data Anonymization: concepts and challenges

- Big economical issue: more or less the backbone of the business models of internet giants (Google, Facebook, Yahoo etc.).
- Big political issue: Open Data Policy.

Social Data Anonymization: concepts and challenges

- Big economical issue: more or less the backbone of the business models of internet giants (Google, Facebook, Yahoo etc.).
- Big political issue: Open Data Policy.
- Raw problem: given a graph G we would like to produce G' such that
  - $Stat(G) \simeq Stat(G')$
  - It is not possible to reidentify nodes (or edges) of G from knowing G' (and some extra informations...).

Social Data Anonymization: concepts and challenges

- Big economical issue: more or less the backbone of the business models of internet giants (Google, Facebook, Yahoo etc.).
- Big political issue: Open Data Policy.
- Raw problem: given a graph G we would like to produce G' such that
  - $Stat(G) \simeq Stat(G')$
  - It is not possible to reidentify nodes (or edges) of G from knowing G' (and some extra informations...).
- Naïve approach doesn't work : Netflix [NarayanShmatikov06].
- Anonymization is an active research field ... rather artistic at the time: approaches validated through experiments.

Social Data Anonymization: Dimensions and Principles

- Problem more down to the earth than non-interference:
  - Partial knowledge of the graph by the opponent.
  - Active attacker (embedding fake sub graphs to re-identify them).
  - Object of interests vary from one data set to another.

Social Data Anonymization: Dimensions and Principles

- Problem more down to the earth than non-interference:
  - Partial knowledge of the graph by the opponent.
  - Active attacker (embedding fake sub graphs to re-identify them).
  - Object of interests vary from one data set to another.
- Hence three important points to consider:
  1. Background Knowledge: What does the opponent know ? Model of the opponent.
  2. Privacity: what is attacked ?
  3. Usage: How the data is going to be analyzed ?

$\Longrightarrow$ Anonymizing techniques

Social Data Anonymization: Techniques

- Two families:
  - Clustering: group together edges and nodes.
  - k-anonymity (and l-diversity): there should be at least k-1 other candidates with similar features.

Social Data Anonymization: Techniques

- Two families:
  - Clustering: group together edges and nodes.
  - k-anonymity (and l-diversity): there should be at least k-1 other candidates with similar features.
- We focus on the k-anonymity approach: the problem amounts to create G' such that $G' = G1 \oplus G2 \oplus ... \oplus G_k$ such that G_is are isomorphic graphs.

Social Data Anonymization: Techniques

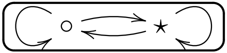
- Two families:
  - Clustering: group together edges and nodes.
  - k-anonymity (and l-diversity): there should be at least k-1 other candidates with similar features.
- We focus on the k-anonymity approach: the problem amounts to create G' such that $G' = G1 \oplus G2 \oplus ... \oplus G_k$ such that G_is are isomorphic graphs.
- It is NP-hard to find graph transformations minimizing the editing distance between a graph and a k-isomorphic graph.

Social Data Anonymization: Techniques

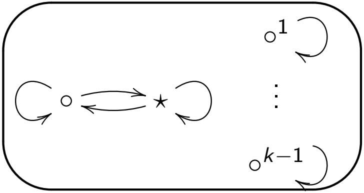
- Two families:
  - Clustering: group together edges and nodes.
  - k-anonymity (and l-diversity): there should be at least k-1 other candidates with similar features.
- We focus on the k-anonymity approach: the problem amounts to create G' such that $G' = G1 \oplus G2 \oplus ... \oplus G_k$ such that G_is are isomorphic graphs.
- It is NP-hard to find graph transformations minimizing the editing distance between a graph and a k-isomorphic graph.
- One solution: select $1/k$ nodes randomly, create k clones, link the clones together easy to program with *AGREE* approach.

Using *AGREE* for k-anonymity

- Progaming with types !
- L is just a cloud of nodes, and K is made of k clones of L.
- Standard T_L is :



- Simplest T_K is :

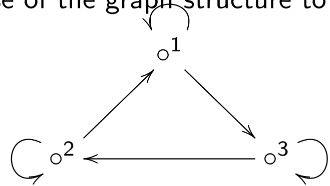


Types and structural graph properties

- The simplest k-clones are not connected to each others.

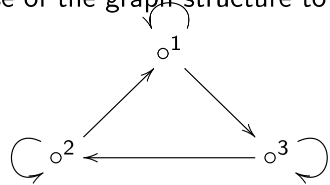
Types and structural graph properties

- The simplest k-clones are not connected to each others.
- AGREE allows the use of the graph structure to reconnect them:



Types and structural graph properties

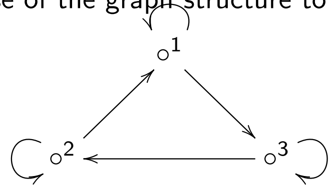
- The simplest k-clones are not connected to each others.
- AGREE allows the use of the graph structure to reconnect them:



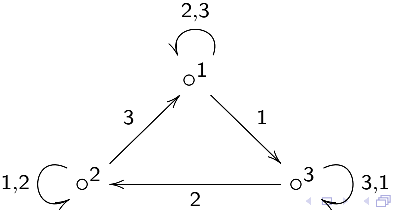
- Degree problems (nodes of degree 1).

Types and structural graph properties

- The simplest *k*-clones are not connected to each others.
- AGREE allows the use of the graph structure to reconnect them:

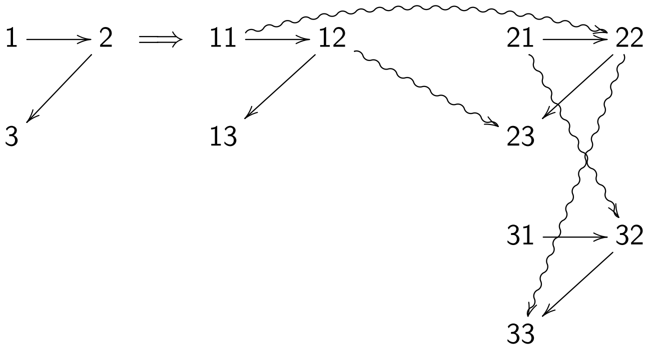


- Degree problems (nodes of degree 1).
  One possibility is to type differently the edges, eg:

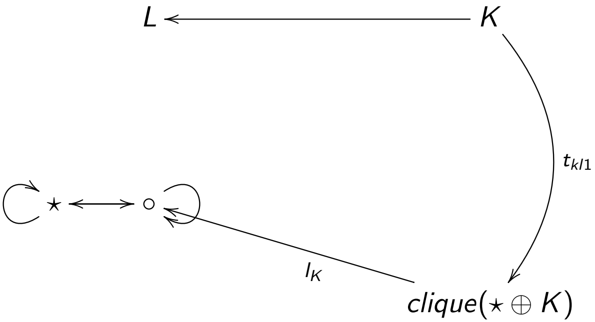


Self-similar graphs

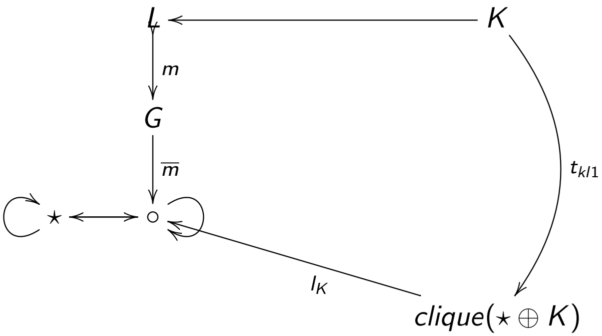
- Every vertex is replaced by a copy of the graph.
- Interconnections between copies of the original “mimic” the ones in the target graph.



Implementation in the AGREE Framework



Implementation in the AGREE Framework



$$
\begin{tikzcd}
 & L \arrow[d, "m"] & & K \arrow[ll] \arrow[ddd, bend left, "t_{kl1}"] \\
 & G \arrow[d, "\overline{m}"] & & \\
\star \arrow[loop left] \arrow[r, leftrightarrow] & \circ \arrow[loop right] & & \\
 & & & clique(\star \oplus K) \arrow[ull, "l_K"]
\end{tikzcd}
$$

Implementation in the AGREE Framework

$$
\begin{tikzcd}
 & L \arrow[d, "m"] & & K \arrow[ll] \arrow[ddd, bend left, "t_{kl1}"] \\
 & G \arrow[d, "\overline{m}"] & & \\
\star \arrow[loop left] \arrow[r, leftrightarrow] & \circ \arrow[loop right] & & TL_1 \arrow[llu, "l_1"'] \arrow[d, "kl_1"] \\
 & & & clique(\star \oplus K) \arrow[llu, "l_K"] \arrow[llu, phantom, "PB(1)"]
\end{tikzcd}
$$

Implementation in the AGREE Framework

$$\begin{tikzcd}
 & L \arrow[d, "m"] & K \arrow[l] \arrow[d, "t_k"] \arrow[dd, bend left, "t_{kl1}"] \\
 & G \arrow[d, "\overline{m}"] & \\
\star \arrow[loop left] \arrow[r, leftrightarrow] & \circ \arrow[loop right] & TL_1 \arrow[l, "\overline{m}\circ l_1"'] \arrow[ul, "l_1"'] \arrow[d, "kl_1"] \arrow[l, phantom, shift right=4, "PB(1)"] \\
 & & clique(\star \oplus K) \arrow[ul, "l_K"]
\end{tikzcd}$$

Implementation in the AGREE Framework

$$
\begin{tikzcd}
 & L \arrow[d, "m"] & & K \arrow[ll] \arrow[dd, "t_k"] \arrow[ddd, bend left] & \\
 & G \arrow[d, "\overline{m}"] & & & D \arrow[lll] \arrow[dl, "t_{kl1}"] \\
\star \arrow[loop left] \arrow[r, leftrightarrow] & \circ \arrow[loop right] & PB(2) & TL_1 \arrow[ull, "l_1"'] \arrow[ll, "\overline{m} \circ l_1"] \arrow[d, "kl_1"] & \\
 & & PB(1) & clique(\star \oplus K) \arrow[ull, "l_K"] &
\end{tikzcd}
$$

Plan

Conclusion

- Categorical frameworks allow simple and mathematically workable definition of complex transformations.
- Only basic constructs are needed: pushouts, pullbacks.
- An very generic implementation is scheduled.
- Open questions:
  - matching ? (random match does not lead to scale-free networks)
  - What statistics can be interesting (Ramsey-like theory) ?
  - What kind of certificate can be produced ?