An Introduction to Graph Rewriting

Rachid Echahed

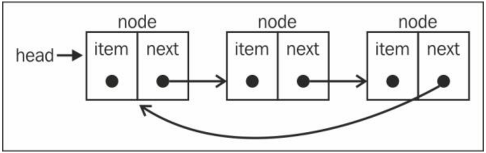
CNRS and Université Grenoble Alpes, Grenoble, France

July 1 and 2, 2019

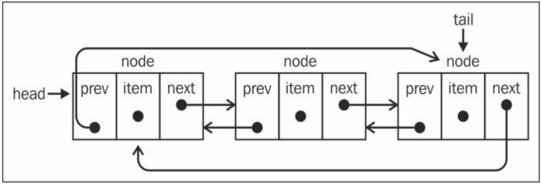
Graph Rewriting: Motivation

Handling real-world data structures

A Circular Linked List

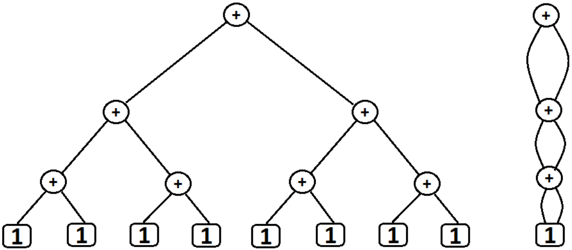


A Doubly-Linked Circular List



Graph Rewriting: Motivation

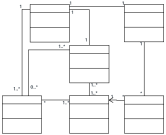
Efficient Implementations

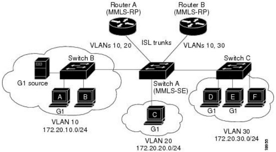


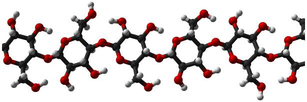
Graph Rewriting: Motivation

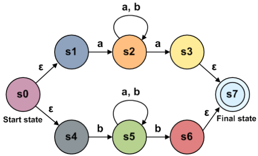
Various Application Domains

Programming, Graph Grammars, UML-like Modeling, Databases, etc.





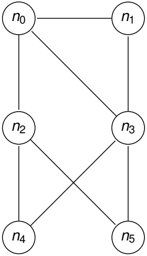


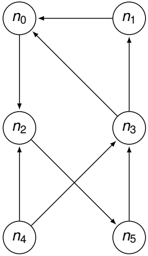


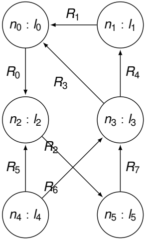
Graph Rewriting

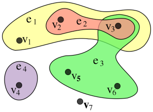
Various Definitions of Graphs

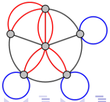
- Undirected graphs
- Directed graphs
- Labeled graphs
- Hypergraphs
- Multigraphs
- Rooted graphs
- Attributed graphs
- ...



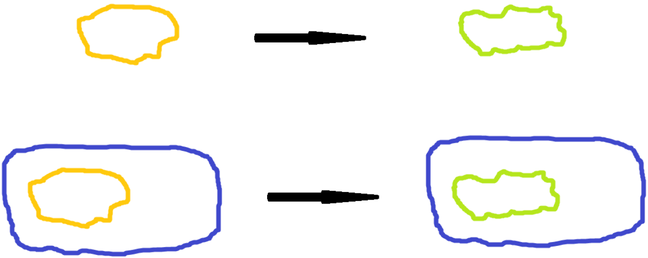




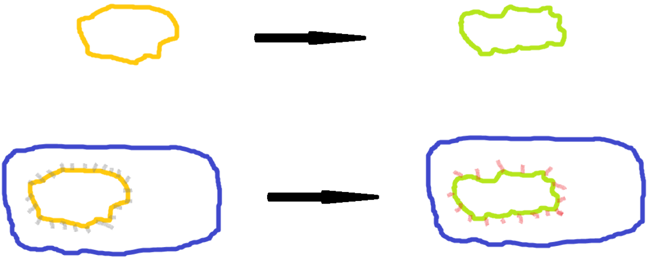




Graph Rewriting



Graph Rewriting

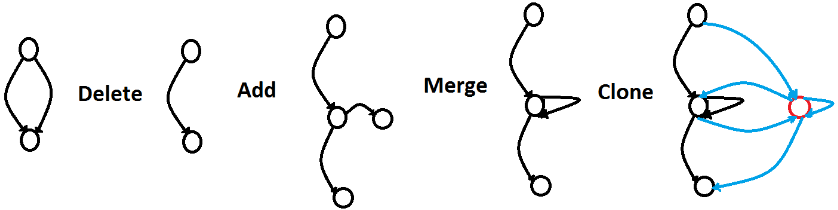


Graph Rewriting : Elementary Actions

There are different possible elementary actions on graphs.

- Delete an existing item (node or edge)
- Add a new item
- Merge two or more items
- Clone (copy) an item or a subgraph
- ...

Graph Rewriting :Elementary Actions



Graph Rewriting

Different frameworks

Since late 1960's!

- There are several approaches, in the literature, to rewrite graphs:
  - Imperative Programs
  - Rule-Based Programs
  - Graph Grammars
  - Knowledge-Base updates
  - Non-classical Logics
  - ...

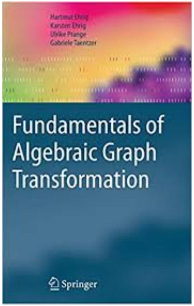
Graph Rewriting

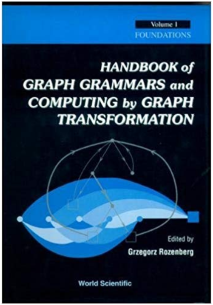
Different frameworks

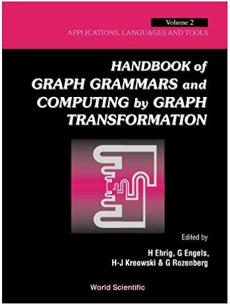
Since late 1960's!

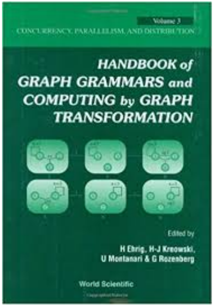
- There are several approaches, in the literature, to rewrite graphs:
  - Imperative Programs
  - Rule-Based Programs
    - Algebraic/Categorial approaches (DPO, SPO, SqPO, PBPO, . . .)
    - Algorithmic approaches
  - Graph Grammars
  - Knowledge-Base updates
  - Non-classical Logics
  - ...

Some References









Outline

Categories

A category C=(Obj_C, Hom_C, $\circ$, id) consists of

- A class Obj_C of objects
- A class Hom_c of morphisms. We write $Hom_c(A, B)$ for the morphisms from object A to B and $f : A \rightarrow B$ an element of $Hom_c(A, B)$
- A composition of morphisms $\circ$. For all objects, A,B and C, $\circ : Hom_c(A, B)xHom_c(B, C) \rightarrow Hom_c(A, C)$.

Such that:

- The composition $\circ$ is associative: For all morphisms $f : A \rightarrow B$, $g : B \rightarrow C$ and $h : C \rightarrow D$, $(h \circ g) \circ f = h \circ (g \circ f)$ and
- For every object A, there exists a morphism $id_A : A \rightarrow A$ called the identity such that: for all morphism $f : A \rightarrow B$, $f \circ id_A = f$ and $id_B \circ f = f$.

Examples of categories

Category of sets :

- objects are sets
- morphisms are functions

Category of graphs :

- objects are graphs
- morphisms are graph homomorphisms

Graphs

In this talk we consider the category of graphs where objects and morphisms are defined as follows:

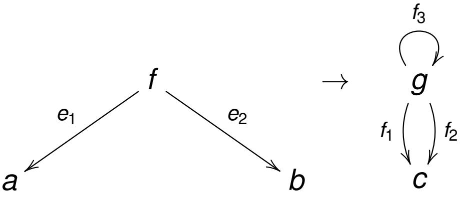
A graph (or multigraph) $G = (N_G, E_G, s_G, t_G)$ consists of

- a set of nodes N_G
- a set of edges E_G
- a source function $s_G : E_G \to N_G$
- a target function $t_G : E_G \to N_G$

A graph homomorphism between two graph G and T, $h : G \to T$, consists of two functions $h_N : N_G \to N_T$ and $h_E : E_G \to E_T$ such that :

- $h_N \circ s_G = s_T \circ h_E$
- $h_N \circ t_G = t_T \circ h_E$

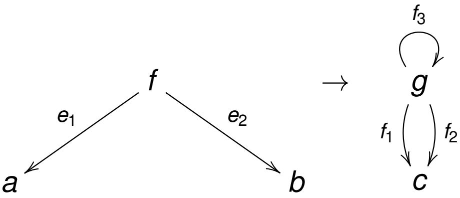
Graph Homomorphism: Example



Graph G　　　　Graph T

$N_G = \{f, a, b\}$ and $E_G = \{e_1, e_2\}$
$N_T = \{g, c\}$ and $E_T = \{f_1, f_2, f_3\}$
Notice that symbols f, a, b, c, g represent nodes and not function symbols!
A first homomorphism $h : G \to T$ can be defined as follows:
$h_N(f) = g$ and $h_N(a) = h_N(b) = c$
$h_E(e_1) = f_1$ and $h_E(e_2) = f_2$

Graph Homomorphism : Example



Graph G

Graph T

$N_G = \{f, a, b\}$ and $E_G = \{e_1, e_2\}$
$N_T = \{g, c\}$ and $E_T = \{f_1, f_2, f_3\}$
Notice that symbols f, a, b, c, g represent nodes and not function symbols!
A second homomorphism $k : G \rightarrow T$ can be defined as follows:
$k_N(f) = k_N(a) = k_N(b) = g$
$k_E(e_1) = k_E(e_2) = f_3$
Are there other homomorphisms between G and T?

Pushout

Definition

$$
\begin{tikzcd}
A \arrow[r, "f"] \arrow[d, "g"'] & B \arrow[d, "g'"] \arrow[ddr, bend left, "u"] & \\
C \arrow[r, "f'"'] \arrow[drr, bend right, "v"'] & D \arrow[dr, dotted, "h"] & \\
& & D'
\end{tikzcd}
$$

The Pushout of morphisms f and g consists of an object D and two morphisms f' and g' such that :

- Commutativity
  $g' \circ f = f' \circ g$, and
- Universal Property
  For all objects D' and morphisms u and v such that $u \circ f = v \circ g$, there exists a unique morphism $h : D \to D'$ such that $h \circ g' = u$ and $h \circ f' = v$.

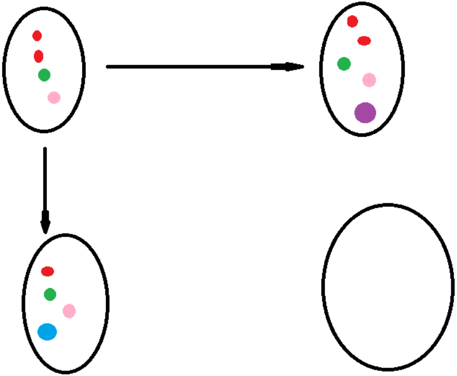
Pushout

$$
\begin{tikzcd}
A \arrow[r, "f"] \arrow[d, "g"'] & B \arrow[d, "g'"] \arrow[ddr, bend left, "u"] & \\
C \arrow[r, "f'"'] \arrow[drr, bend right, "v"'] & D \arrow[dr, dotted, "h"] & \\
 & & D'
\end{tikzcd}
$$

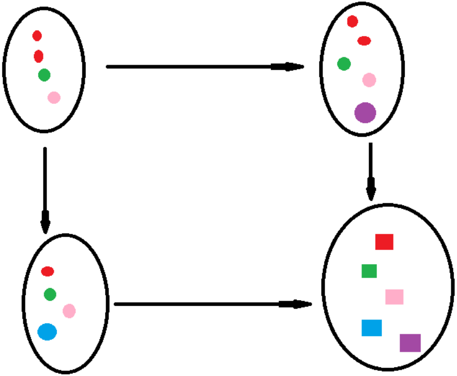
In Sets:

- $D = (B + C)/ \equiv$
  with $\equiv$ being the least equivalence generated by the pairs $\{(f(x), g(x)) \mid x \in A\}$ over $B + C$.
- For all $x \in B$, $g'(x) = \bar{x}$
- For all $x \in C$, $f'(x) = \bar{x}$

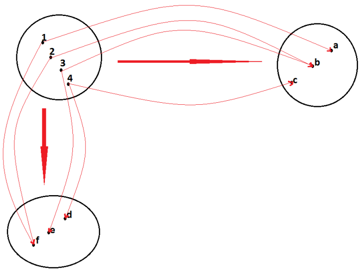
Pushout: Example 1



Pushout: Example 1

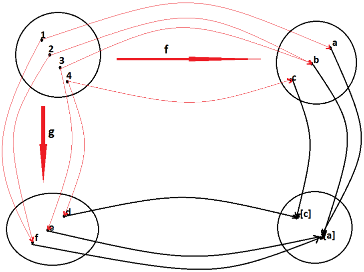


Pushout: Example 2



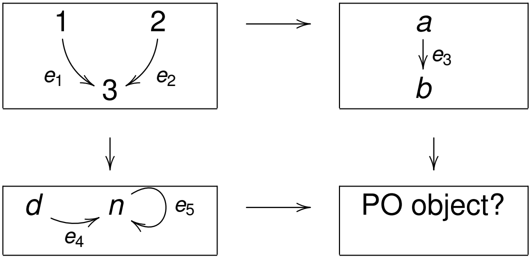
$f(1) = a, f(2) = b, f(3) = b, f(4) = c$
$g(1) = f, g(2) = f, g(3) = e, g(4) = d$

Pushout: Example 2



$f(1) = a, f(2) = b, f(3) = b, f(4) = c$
$g(1) = f, g(2) = f, g(3) = e, g(4) = d$

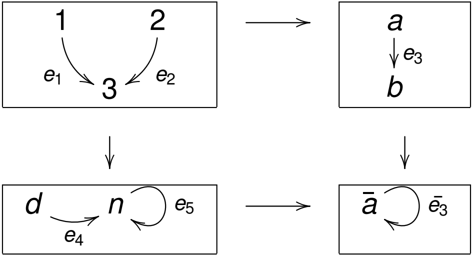
Pushout: Example3



$$f(1) = a, f(2) = a, f(3) = b, f(e_1) = f(e_2) = e_3$$
$$g(1) = n, g(2) = d, g(3) = n, g(e_1) = e_5, g(e_2) = e_4$$

In graphs: The sets of nodes and edges of the pushout object (D) can be constructed componentwise as pushouts in Sets (respecting the source and target functions)

Pushout: Example3



$f(1) = a, f(2) = a, f(3) = b, f(e_1) = f(e_2) = e_3$
$g(1) = n, g(2) = d, g(3) = n, g(e_1) = e_5, g(e_2) = e_4$

In graphs: The sets of nodes and edges of the pushout object (D) can be constructed componentwise as pushouts in Sets (respecting the source and target functions)

Pullback

$$\begin{tikzcd}
 & & D' \arrow[dll, bend right, "u"'] \arrow[dl, dotted, "h"] \arrow[ddl, bend left, "v"] \\
B \arrow[d, "f"'] & \color{blue}{D} \arrow[l, "g'"'] \arrow[d, "f'"] & \\
A & C \arrow[l, "g"] &
\end{tikzcd}$$

The Pullback of morphisms f and g consists of an object D and two morphisms f' and g' such that :

- Commutativity
  $f \circ g' = g \circ f'$, and
- Universal Property
  For all objects D' and morphisms u and v such that $f \circ u = g \circ v$, there exists a unique morphism $h : D' \to D$ such that $g' \circ h = u$ and $f' \circ h = v$.

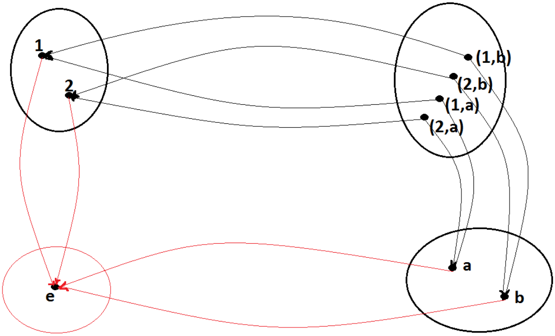
Pullback

$$
\begin{tikzcd}
 & & D' \arrow[dll, bend right, "u"'] \arrow[dl, dotted, "h"] \arrow[ddl, bend left, "v"] \\
B \arrow[d, "f"'] & D \arrow[l, "g'"'] \arrow[d, "f'"] & \\
A & C \arrow[l, "g"] &
\end{tikzcd}
$$

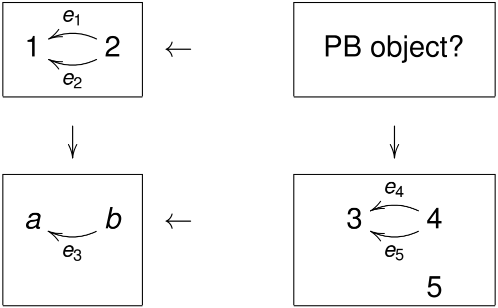
In Sets,

- $D = \{(x, y) \in BxC \mid f(x) = g(y)\}$
- For all $(b, c) \in D$, $g'(b, c) = b$
- For all $(b, c) \in D$, $f'(b, c) = c$

Pullback



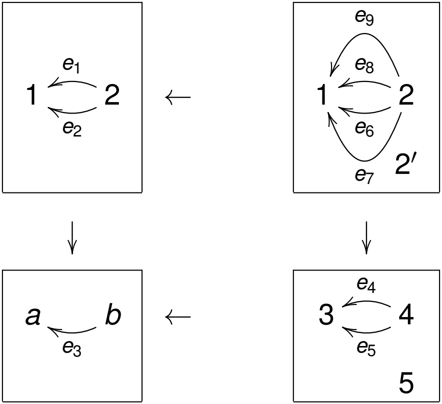
Pullback



$f(1) = a, f(2) = b, f(e_1) = f(e_2) = e_3$
$g(3) = a, g(4) = g(5) = b, g(e_4) = g(e_5) = e_3$

In graphs: The sets of nodes and edges of pullback object D can be constructed componentwise as pullbacks in Sets (respecting the source and target functions)

Pullback



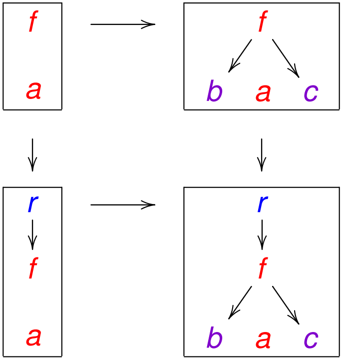
$f(1) = a, f(2) = b, f(e_1) = f(e_2) = e_3$
$g(3) = a, g(4) = g(5) = b, g(e_4) = g(e_5) = e_3$

In graphs: The sets of nodes and edges of pullback object D can be constructed componentwise as pullbacks in Sets (respecting the source and target functions)

Outline

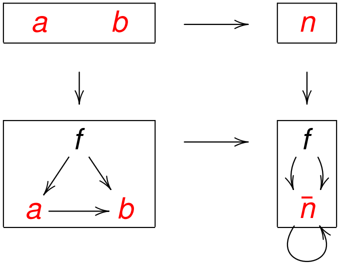
Adding New Items

Pushouts can be used to add new items to a graph.



Merging Existing Items

Pushouts can be used to merge existing items of a graph.



Deleting Existing Items

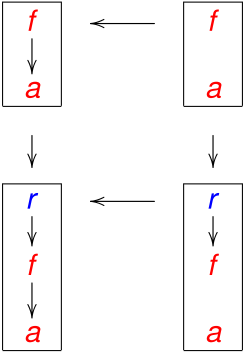
- Both Pushouts and Pullbacks can be used to delete items within a graph!
- Single pushout can be used to delete items in a graph but requires partial morphisms (out of this talk).

Use of pushout complement: A pushout complement (POC) of two morphisms $m : L \to G$ and $l : K \to L$ is an object D and two morphisms $l' : D \to G$ and $m' : K \to D$ such that the following diagram is a pushout :

$$\begin{tikzcd} L \arrow[d, "m"'] & K \arrow[l, "l"'] \arrow[d, "m'"] \\ G & D \arrow[l, "l'"] \end{tikzcd}$$

Deleting Existing Items

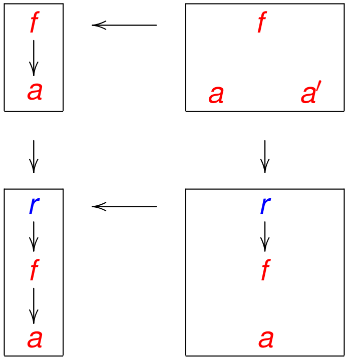
Example of the use of pushout complement



Remark: Pushout complements may not exist or not be unique!

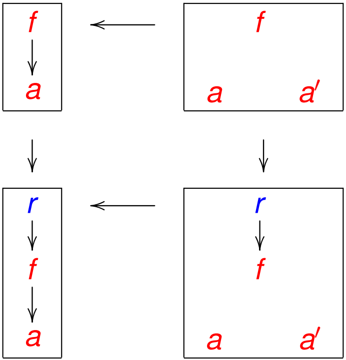
Pushout Complement

Pushout complements may not be unique!



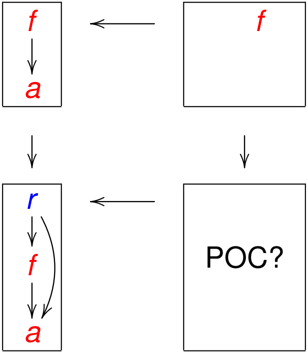
Pushout Complement

Pushout complements may not be unique!



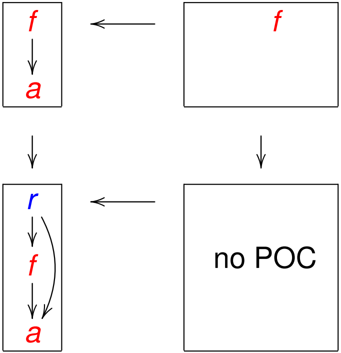
Pushout Complement

Exercise



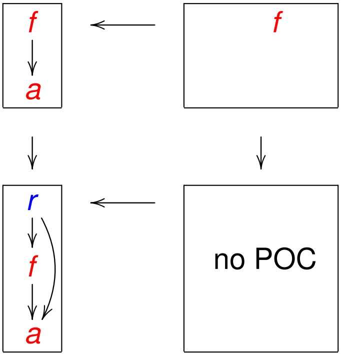
Pushout Complement

Pushout complement may not exist!

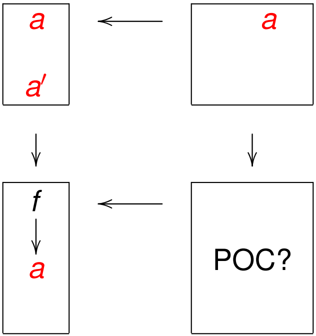


Pushout Complement

Pushout complement may not exist!

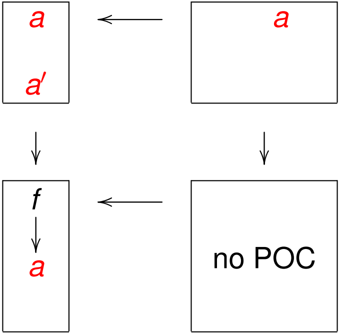


Pushout Complement



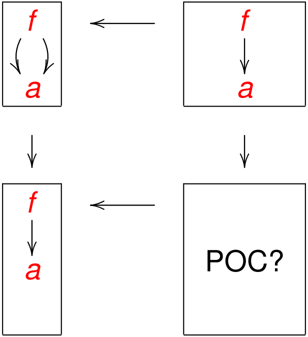
Pushout Complement

Pushout complement may not exist!



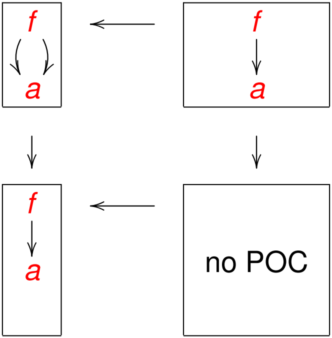
Pushout Complement

Exercise



Pushout Complement

Pushout complement may not exist!



Existence of Pushout Complements (in Graphs)

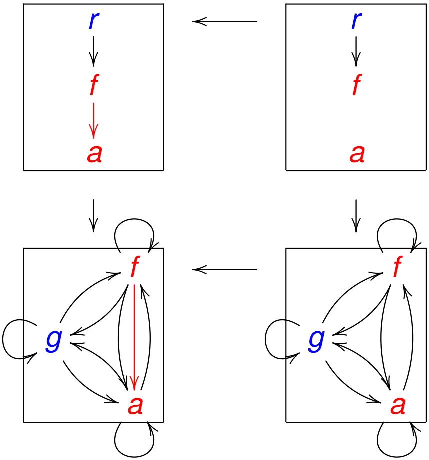
$$\begin{tikzcd} L & K \arrow[l, "l"'] \arrow[d, "m'"] \\ G \arrow[from=u, "m"'] & D \arrow[l, "l'"] \end{tikzcd}$$

Let $m : L \to G$ and $l : K \to L$ be two graph morphisms. There exits a pushout complement defined by a graph D and two morphisms $l' : D \to G$ and $m' : K \to D$ iff the following *gluing* conditions hold :

- Dangling Condition:
  $\{n \in N_L \mid \exists e \in E_G \setminus m(E_L), s_G(e) = m(n) \text{ or } t_G(e) = m(n)\} \subseteq l(N_K)$
- Identification Condition:
  - $\{n \in N_L \mid \exists n' \in N_L, n \neq n' \text{ and } m(n) = m(n')\} \subseteq l(N_k)$
  - $\{e \in E_L \mid \exists e' \in E_L, e \neq e' \text{ and } m(e) = m(e')\} \subseteq l(E_k)$

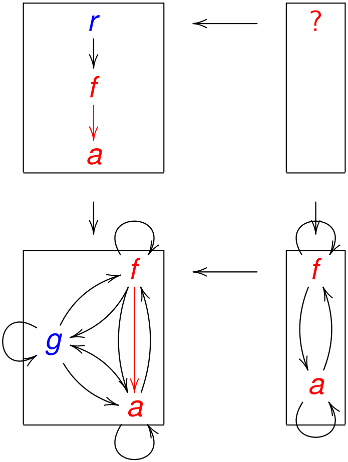
Deleting Existing Items

Use of pullbacks: Example



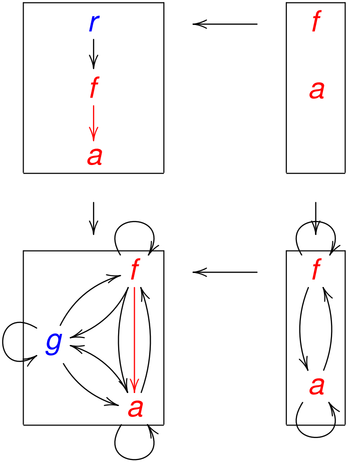
Deleting Existing Items

Use of pullbacks: Example



Deleting Existing Items

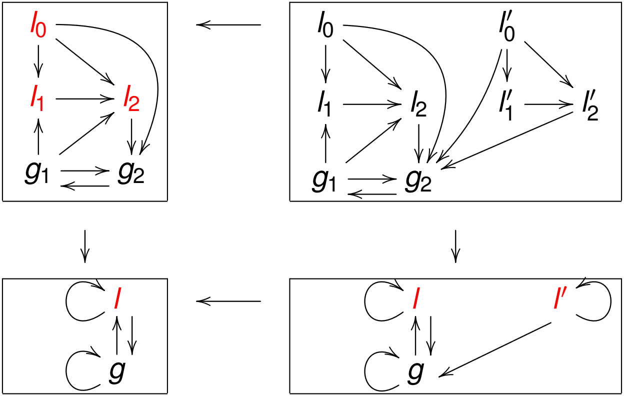
Use of pullbacks: Example



Cloning Items

Use of Pullbacks

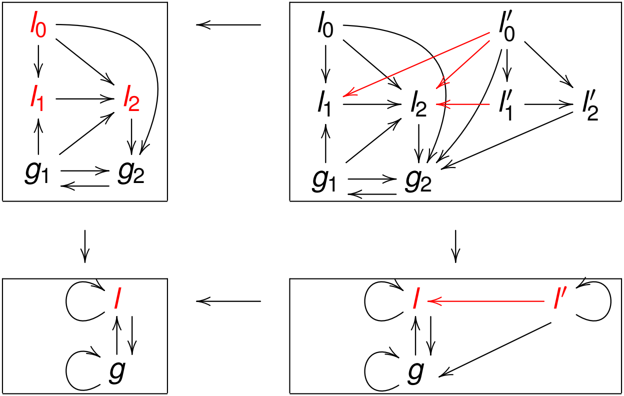
Cloning the subgraph containing nodes l_0, l_1, l_2



Cloning Items

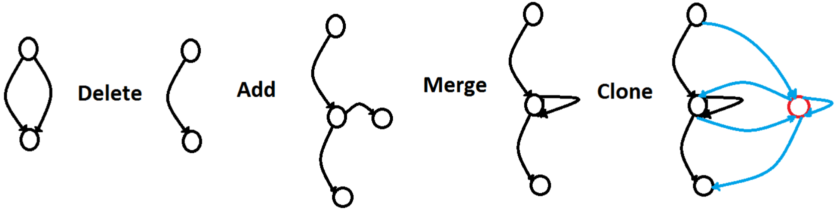
Use of Pullbacks

Cloning the subgraph containing nodes l_0, l_1, l_2



Graph Rewriting

Give three rules implementing the following evolution



Starting from a graph G modeling agents (A), files (F) and an arbitrary access relation (R) including possible prohibited accesses ($R \subseteq AxF$), give a rewrite rule which transforms G into a graph that satisfies the following policy.

There are two responsibility levels among agents : H and L. Files are classified according to 3 security levels : 1, 2 and 3. Agents of responsibility level H have the right to access files of security levels 1 and 2. Agents of responsibility level L have the right to access files of security levels 2 and 3.

Outline

DPO Rules

First things first!

[EPS'73] H. Ehrig, M. Pfender, H. J. Schneider: Graph-Grammars: An Algebraic Approach. SWAT (FOCS) 1973: 167-180

$$L \longleftarrow K \longrightarrow R$$

DPO Rules

First things first!

[EPS'73] H. Ehrig, M. Pfender, H. J. Schneider: Graph-Grammars: An Algebraic Approach. SWAT (FOCS) 1973: 167-180

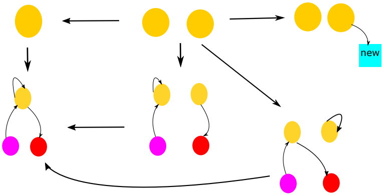
$$L \longleftarrow K \longrightarrow R$$

A DPO rewrite step :

$$\begin{tikzcd}
L \arrow[d, "m"] \arrow[dr, phantom, "POC"] & K \arrow[l, "l"'] \arrow[r, "r"] \arrow[d, "d"] \arrow[dr, phantom, "PO"] & R \arrow[d, "m'"] \\
G & \color{red}{D} \arrow[l, "l'"] \arrow[r, "r'"'] & H
\end{tikzcd}$$

DPO Rules

An example



POCs (Pushout complement) are not unique when cloning items!
Delete actions are restricted by the gluing conditions.

SQPO Rules

$$L \longleftarrow K \longrightarrow R$$

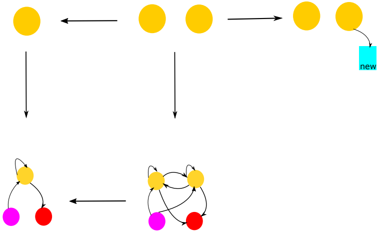
A SQPO rewrite step :

$$\begin{tikzcd} L \arrow[d, "m"] \arrow[dr, phantom, "FPBC"] & K \arrow[l, "l"'] \arrow[r, "r"] \arrow[d, "d"] \arrow[dr, phantom, "PO"] & R \arrow[d, "m'"] \\ G & \color{red}{D} \arrow[l, "l'"] \arrow[r, "r'"'] & H \end{tikzcd}$$

[ICGT 2006] Corradini et al. Sesqui-Pushout Rewriting. ICGT 2006: 30-45

SQPO Rules

An example



Clone action is still quite limited!

AGREE Rules

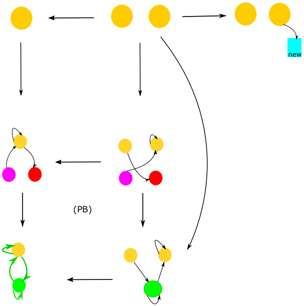
$$
\begin{tikzcd}
L & K \arrow[l, "l"'] \arrow[r, "r"] \arrow[d, "t"] & R \\
 & T_K &
\end{tikzcd}
$$

$$
\begin{tikzcd}
L \arrow[d, "m"] \arrow[dd, bend right=60, "\eta_L"'] & & K \arrow[ll, "l"'] \arrow[rr, "r"] \arrow[d, "n"'] \arrow[dd, bend left=60, "t"] \arrow[drr, phantom, "PO\ (b)"] & & R \arrow[d, "p"] \\
G \arrow[u, phantom, "="', shift right=6ex] \arrow[d, "\overline{m}"] \arrow[drr, phantom, "PB\ (a)"] & & D \arrow[ll, "g"'] \arrow[rr, "h"] \arrow[d, "n'"'] & & H \\
T(L) & & T_K \arrow[ll, "l'"] & &
\end{tikzcd}
$$

Caution : The definition of AGREE transformation requires the existence, in the underlying category, of a *partial map classifier* [ICGT 2015][TCS 2019,to appear]

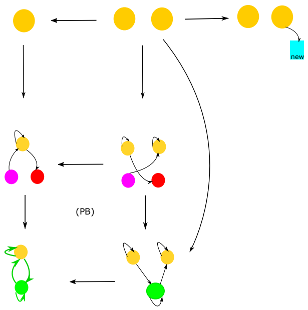
AGREE Rules

An example



AGREE Rules

An example



Clone action is more flexible than SQPO but can still be improved!

PBPO Rules

A PBPO rule consists of a (classical) first span of the form:

$$L \leftarrow K \rightarrow R$$

to which it is added a (typing) second span

$$L' \leftarrow K' \rightarrow R'$$

such that the two following squares commute :

$$\begin{tikzcd} L \arrow[d, "t_L"] & & K \arrow[ll, "l"'] \arrow[rr, "r"] \arrow[d, "t_K"] & & R \arrow[d, "t_R"] \\ L' & & K' \arrow[ll, "l'"] \arrow[rr, "r'"'] & & R' \end{tikzcd}$$

[JLAMP2019]The PBPO graph transformation approach. J. Log. Algebr. Meth. Program. 103: 213-231 (2019)

PBPO

A rewrite step

PBPO rule :

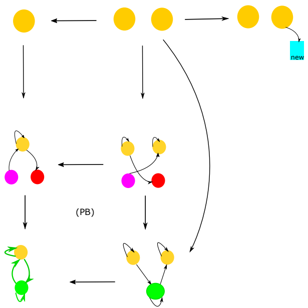
$$\begin{tikzcd}
L \arrow[d, "t_L"] \arrow[dr, phantom, "="] & K \arrow[l, "l"'] \arrow[r, "r"] \arrow[d, "t_K"] \arrow[dr, phantom, "="] & R \arrow[d, "t_R"] \\
L' & K' \arrow[l, "l'"] \arrow[r, "r'"'] & R'
\end{tikzcd}$$

PBPO rewrite step : The match is defined as a pair (m, m')!

$$\begin{tikzcd}
L \arrow[d, "m"] \arrow[dd, bend right=60, "t_L"'] \arrow[dr, phantom, "= (a')"] & K \arrow[l, "l"'] \arrow[r, "r"] \arrow[d, "n"] \arrow[dd, bend right=60, "t_K"'] \arrow[dr, phantom, "PO\ (b)"] & R \arrow[d, "p"] \arrow[dd, bend right=60, "t_R"'] \\
G \arrow[d, "m'"] \arrow[dr, phantom, "PB\ (a)"] & D \arrow[l, "g"'] \arrow[r, "h"] \arrow[d, "n'"] \arrow[dr, phantom, "= (b')"] & H \arrow[d, "p'"] \\
L' & K' \arrow[l, "l'"] \arrow[r, "r'"'] & R'
\end{tikzcd}$$

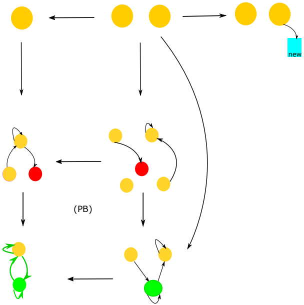
PBPO Rewrite Step

Example



PBPO Rewrite Step

Example

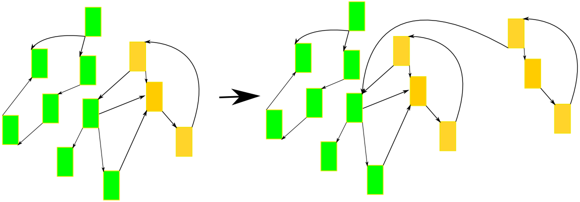


The PBPO Approach :

Exercise

Give a rewrite rule that makes a copy of the pages of a local web site or a copy of a whole directory

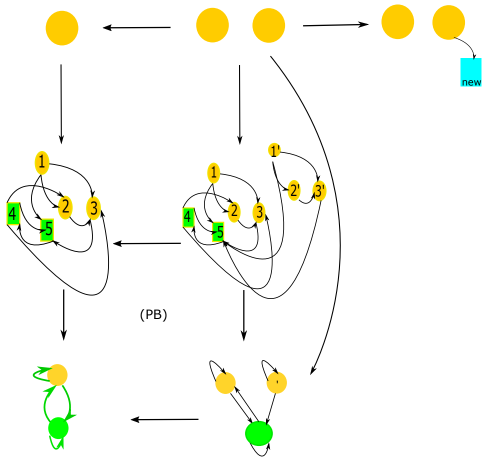
```
cp -r <directory> <new directory>
cp <a local web site>
```



AGREE needs a new rule for every specific shape of the web site
PBPO uses only one generic rule!

PBPO Rewrite Step

Example of the copy of local Web pages



PBPO vs AGREE, SQPO

Proposition

Let α be an AGREE rule in a category with a partial map classifier. Then there is a PBPO rule ρ_α such that for each mono $m : L \rightarrowtail G$ we have $G \Rightarrow_{\alpha}^{\text{AGREE}} H$ if and only if $G \Rightarrow_{\rho_\alpha} H$ using match $(m, \overline{m})$ with $\overline{m} : G \to T(L)$.

$$
\begin{tikzcd}
L & K \arrow[l, "l"'] \arrow[r, "r"] \arrow[d, tail, "t"] & R \\
 & T_K &
\end{tikzcd}
\qquad
\begin{tikzcd}
L \arrow[d, tail, "m"] \arrow[dd, bend right=60, "\eta_L"', "="] & & K \arrow[ll, "l"'] \arrow[rr, "r"] \arrow[d, tail, "n"'] \arrow[dd, bend left=60, "t", "="'] \arrow[drr, phantom, "PO\ (b)"] & & R \arrow[d, "p"] \\
G \arrow[d, "\overline{m}"] \arrow[drr, phantom, "PB\ (a)"] & & D \arrow[ll, "g"'] \arrow[rr, "h"] \arrow[d, "n'"'] & & H \\
T(L) & & T_K \arrow[ll, "l'"] & &
\end{tikzcd}
$$

Add R' as a Pushout of morphisms t and r to end the construction!

PBPO vs SQPO

Proposition

Let α be a SQPO rule in a category with a partial map classifier. Then there is a PBPO rule ρ_α such that for each mono $m : L \rightarrowtail G$ we have $G \Rightarrow_\alpha^{\text{SQPO}} H$ if and only if $G \Rightarrow_{\rho_\alpha} H$ using match $(m, \overline{m})$ with $\overline{m} : G \to T(L)$.

$$
\begin{tikzcd}
L \arrow[d, "m"'] \arrow[dr, phantom, "FPBC"] & K \arrow[l, "l"'] \arrow[r, "r"] \arrow[d, "t"] & R \arrow[d, "p_{\eta_L}"] \\
G & D \arrow[l, "g"] \arrow[r, "h"'] & H
\end{tikzcd}
\qquad
\begin{tikzcd}
L \arrow[d, "m"] \arrow[dd, bend right=60, "="'] & & K \arrow[ll, "l"'] \arrow[rr, "r"] \arrow[d, "n"'] \arrow[dd, bend left=60, "t"] \arrow[drr, phantom, "PO\ (b)"] & & R \arrow[d, "p"] \\
G \arrow[d, "\overline{m}"] \arrow[drr, phantom, "PB\ (a)"] & & D \arrow[ll, "g"'] \arrow[rr, "h"] \arrow[d, "n'"'] & & H \\
T(L) & & {\color{red}T(K)} \arrow[ll, "l'"] & &
\end{tikzcd}
$$

Add R' as a Pushout of morphisms t and r to end the construction!

Outline

Attributed Graphs

definition borrowed from [DEPR, FASE2014]

- Let **Graph** be a category of structures (e.g., graphs)
- Let **Att** be a category of attribute structures (e.g., Σ-algebras)
- Let S : **Graph** $\rightarrow$ **Set** be a functor
- Let T : **Att** $\rightarrow$ **Set** be a functor

Definition

The category **AttG** of attributed graphs is defined as the comma category $S \downarrow T$.

Attributed Graphs

- Let S : **Graph** $\to$ **Set** be a functor
- Let T : **Att** $\to$ **Set** be a functor
- Attributed Graph : $\widehat{G} = (G, A, \alpha)$
  - G in **Graph**,
  - A in **Att** and
  - $\alpha : S(G) \to T(A)$ (in **Set**) is a labelling function
- Morphisms : $\widehat{g} : \widehat{G} \to \widehat{G'}$, where $\widehat{G} = (G, A, \alpha)$ and $\widehat{G'} = (G', A', \alpha')$, is a pair $\widehat{g} = (g, a)$ with $g : G \to G'$ is a morphism in **Graph** and $a : A \to A'$ is a morphism in **Att** such that $\alpha' \circ Sg = Ta \circ \alpha$ (in **Set**).

$$
\begin{tikzcd}
\widehat{G} \arrow[d, "\widehat{g}"'] \\
\widehat{G'}
\end{tikzcd}
\quad = \quad
\begin{tikzcd}
G \arrow[d, "g"'] \\
G'
\end{tikzcd}
\qquad
\begin{tikzcd}
SG \arrow[r, "\alpha"] \arrow[d, "Sg"'] \arrow[dr, phantom, "="] & TA \arrow[d, "Ta"] \\
SG' \arrow[r, "\alpha'"] & TA'
\end{tikzcd}
\qquad
\begin{tikzcd}
A \arrow[d, "a"] \\
A'
\end{tikzcd}
$$

Partially Attributed Graphs

- Partially Attributed Graph : $\widehat{G} = (G, A, \alpha)$
  - G in **Graph**,
  - A in **Att** and
  - $\alpha : S_p(G) \to T_p(A)$ (in **Pfn**) is a partial labeling function
- Morphisms: $\widehat{g} : \widehat{G} \to \widehat{G'}$, where $\widehat{G} = (G, A, \alpha)$ and $\widehat{G'} = (G', A', \alpha')$, is a pair $\widehat{g} = (g, a)$ with $g : G \to G'$ is a morphism in **Graph** and $a : A \to A'$ is a morphism in **Att** such that $\alpha' \circ S_p g \geq T_p a \circ \alpha$ (in **Pfn**).

$$\begin{array}{c} \widehat{G} \\ \widehat{g}\downarrow \\ \widehat{G'} \end{array} \quad = \quad \begin{array}{c} G \\ g\downarrow \\ G' \end{array} \quad \begin{tikzcd} S_pG \arrow[r, "\alpha"] \arrow[d, "S_pg"'] \arrow[dr, phantom, "\geq"] & T_pA \arrow[d, "T_pa"] \\ S_pG' \arrow[r, "\alpha'"] & T_pA' \end{tikzcd} \quad \begin{array}{c} A \\ \downarrow a \\ A' \end{array}$$

Remark: $\geq$ states that morphisms preserve defined attributes
A morphism of partially attributed structures (g, a) is called strict when $\alpha' \circ S_p g = T_p a \circ \alpha$.

PBPO Rules for Attributed Graphs

$$\begin{tikzcd}
(L, A, \alpha_L) \arrow[d, "\widehat{t}_L"] \arrow[dr, phantom, "="] & (K, A, \alpha_K) \arrow[l, "{(l, id_A)}"'] \arrow[r, "{(r, id_A)}"] \arrow[d, "\widehat{t}_K"] \arrow[dr, phantom, "="] & (R, A, \alpha_R) \arrow[d, "\widehat{t}_R"] \\
(L', A', \alpha_{L'}) & (K', A', \alpha_{K'}) \arrow[l, "{(l', id_{A'})}"] \arrow[r, "{(r', id_{A'})}"'] & (R', A', \alpha_{R'})
\end{tikzcd}$$

with the additional conditions

- $\alpha_L, \alpha'_L, \alpha_R$ and α'_R are total labeling functions
- The morphism $\widehat{t}_K$ is strict
- The morphism $\widehat{t}_K$ is injective on non-attributed items

PBPO Rewrite Step

The Attributed case

$$\begin{tikzcd}[column sep=huge]
(L, A, \alpha_L) \arrow[d, "{(m,a)}"] \arrow[dd, bend right, "\widehat{t}_L"'] \arrow[dr, phantom, "="] & (K, A, \alpha_K) \arrow[l, "{(l,id_A)}"'] \arrow[r, "{(r,id_A)}"] \arrow[d, "{(n,a)}"] \arrow[dd, bend right, "\widehat{t}_K"'] \arrow[dr, phantom, "PO"] & (R, A, \alpha_R) \arrow[d, "{(p,a)}"] \arrow[dd, bend right, "\widehat{t}_R"'] \\
(G, A_0, \alpha_G) \arrow[d, "{(m',a')}"] \arrow[dr, phantom, "PB"] & (D, A_0, \alpha_D) \arrow[l, "{(g,id_{A_0})}"] \arrow[r, "h"] \arrow[d, "{(n',a')}"] \arrow[dr, phantom, "="] & (H, A_0, \alpha_H) \arrow[d, "{(p',a')}"] \\
(L', A', \alpha_{L'}) & (K', A', \alpha_{K'}) \arrow[l, "{(l',id_{A'})}"] \arrow[r, "{(r',id_{A'})}"'] & (R', A', \alpha_{R'})
\end{tikzcd}$$

Does H always exist?
Is H completely attributed?

PBPO Rewrite Step

Easy examples

$$\begin{tikzcd}
\boxed{n : x} \arrow[d] & \boxed{n : x} \arrow[l] \arrow[r] \arrow[d] & \boxed{n : x} \arrow[d] \\
\boxed{n : 6} \arrow[d] & \boxed{n : 6} \arrow[l] \arrow[r] \arrow[d] & \boxed{n : 6} \arrow[d] \\
\boxed{n : nat} & \boxed{n : nat} \arrow[l] \arrow[r] & \boxed{n : nat}
\end{tikzcd}$$

Node n is preserved together with its attribute

PBPO Rewrite Step

Easy examples

$$\begin{tikzcd}
\boxed{n : x} \arrow[d] & \boxed{n : \bot} \arrow[l] \arrow[r] \arrow[d] & \boxed{n : char_of_int(suc(x))} \arrow[d] \\
\boxed{n : 68} \arrow[d] & \boxed{n : \bot} \arrow[l] \arrow[r] \arrow[d] & \boxed{n : \text{“E”}} \arrow[d] \\
\boxed{n : nat} & \boxed{n : \bot} \arrow[l] \arrow[r] & \boxed{n : char}
\end{tikzcd}$$

Node *n* is preserved but re-attributed

PBPO Rewrite Step

Problematic Examples

$$
\begin{tikzcd}
\boxed{n : x} & \boxed{n : \perp} \arrow[l] \arrow[r, "\widehat{r}"'] \arrow[d, "\widehat{n}"] & \boxed{n : 2} \arrow[d] \\
\boxed{n : 6} \arrow[d] & \boxed{n : 6} \arrow[l] \arrow[r] \arrow[d] & \boxed{n : ?} \arrow[d] \\
\boxed{n : nat} & \boxed{n : nat} \arrow[l] \arrow[r] & \boxed{n : nat}
\end{tikzcd}
$$

Case of a non strict $\widehat{t_K}$

PBPO Rewrite Step

Problematic Examples

$$
\begin{array}{ccccc}
\boxed{\begin{array}{c} n = n' : x \\ \end{array}} & \leftarrow & \boxed{\begin{array}{c} n : \bot \\ n' : \bot \end{array}} & \rightarrow & \boxed{\begin{array}{c} n : 2 \\ n' : 3 \end{array}} \\
\downarrow & & \downarrow & & \downarrow \\
\boxed{n = n' : 6} & \leftarrow & \boxed{n = n' : \bot} & \rightarrow & \boxed{n = n' : ?} \\
\downarrow & & \downarrow & & \downarrow \\
\boxed{n = n' : nat} & \leftarrow & \boxed{n = n' : \bot} & \rightarrow & \boxed{n = n' : nat}
\end{array}
$$

Case where $\widehat{t_K}$ is not injective on non-attributed items

PBPO Rewrite Step

Problematic Examples

$$\begin{array}{ccccc}
\boxed{n : x} & \leftarrow & \boxed{n : \bot} & \rightarrow & \boxed{n : suc(x)} \\
\downarrow & & \downarrow & & \downarrow \\
\boxed{\begin{array}{c} n : 6 \\ n' : 8 \end{array}} & \leftarrow & \boxed{\begin{array}{c} n : \bot \\ n' : \bot \end{array}} & \rightarrow & \boxed{\begin{array}{c} n : 7 \\ n' : \bot \end{array}} \\
\downarrow & & \downarrow & & \downarrow \\
\boxed{\begin{array}{c} n : nat \\ n' : nat \end{array}} & \leftarrow & \boxed{\begin{array}{c} n : \bot \\ n' : \bot \end{array}} & \rightarrow & \boxed{\begin{array}{c} n : nat \\ n' : nat \end{array}}
\end{array}$$

Case where a non-attributed element in $\widehat{K'}$, n', has no antecedent in $\widehat{K}$.

Existence and Total Attribution of Transformed Graphs

$$
\begin{tikzcd}
(L, A, \alpha_L) \arrow[d, "{(m,a)}"] \arrow[dd, bend right, "\widehat{t}_L"'] \arrow[dr, phantom, "="] & (K, A, \alpha_K) \arrow[l, "{(l,id_A)}"'] \arrow[r, "{(r,id_A)}"] \arrow[d, "{(n,a)}"] \arrow[dd, bend right, "\widehat{t}_K"'] \arrow[dr, phantom, "PO"] & (R, A, \alpha_R) \arrow[d, "{(p,a)}"] \arrow[dd, bend right, "\widehat{t}_R"'] \\
(G, A_0, \alpha_G) \arrow[d, "{(m',a')}"] \arrow[dr, phantom, "PB"] & (D, A_0, \alpha_D) \arrow[l, "{(g,id_{A_0})}"'] \arrow[r, "h"] \arrow[d, "{(n',a')}"] \arrow[dr, phantom, "="] & (H, A_0, \alpha_H) \arrow[d, "{(p',a')}"] \\
(L', A', \alpha_{L'}) & (K', A', \alpha_{K'}) \arrow[l, "{(l',id_{A'})}"] \arrow[r, "{(r',id_{A'})}"'] & (R', A', \alpha_{R'})
\end{tikzcd}
$$

Proposition

If the following conditions hold

- $\alpha_L, \alpha'_L, \alpha_R$ and α'_R are total labeling functions
- The morphism $\widehat{t}_K$ is strict and injective on non-attributed items
- G is completely attributed
- $\forall n \in G$, if $\exists n_{K'} \in K'$ such that $n_{K'}$ is not attributed and $m'(n) = l'(n_{K'})$, then $\exists n_k \in K$ with $n = m(l(n_K))$ and $n_{K'} = t_K(n_K)$.

Then the graph H exists and is completely attributed

Outline

Termgraph Rewriting
Motivation

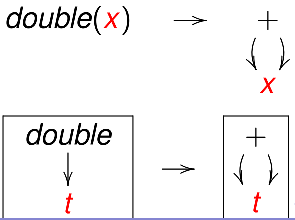
- Handling Data-structure rewriting
  including cyclic data-structures with pointers
  such as circular lists, doubly-linked lists, etc.
- Data-structures are more complex than terms (Cycles, Sharing)
- Difficult to encode efficiently using terms
- Usually described by pointers ($\Rightarrow$ pointer rewriting)
- Formally described as termgraphs
  Informally: termgraph = term with cycles and sharing

Termgraph Rewriting

Motivation

$$
\begin{array}{lcl}
0 + x & \rightarrow & x \\
s(x) + y & \rightarrow & s(x + y) \\
double(x) & \rightarrow & x + x
\end{array}
$$

Term rewrite systems constitute a very well established domain with several results : Confluence, Termination, Strategies, Proof methods (equational reasoning, induction) etc.
However, subterm sharing, as in termgraph, does not preserve classical properties of term rewriting such as, e.g., the confluence property.



Sharing Subterms (information) and Term Rewriting

Consider the following rules:

$$\begin{array}{lcl} f(a,b) & \rightarrow & c \\ a & \rightarrow & b \end{array}$$

Sharing does not preserve properties of tree (term) rewriting !

Term rewrite derivation: $f(a,a) \rightarrow f(a,b) \rightarrow c$

Termgraph rewrite derivation: $\begin{tikzcd} f \arrow[d, bend right] \arrow[d, bend left] \\ a \end{tikzcd} \rightarrow \begin{tikzcd} f \arrow[d, bend right] \arrow[d, bend left] \\ b \end{tikzcd} \not\rightarrow$

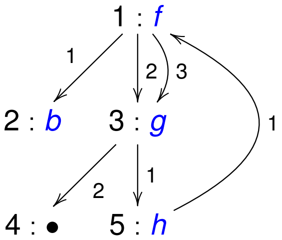
[Plump 99] survey on rewriting with “dags”.

Termgraphs

[Barendregt et al. 87]
[Plump 99, survey on *acyclic* term-graphs]

Let Ω be a set of operation symbols.
A *term-graph t* over Ω is defined by:

- a set of nodes $\mathcal{N}_t$,
- a subset of labeled nodes $\mathcal{N}_t^\Omega \subseteq \mathcal{N}_t$,
- a labeling function $\mathcal{L}_t : \mathcal{N}_t^\Omega \to \Omega$,
- a successor function $\mathcal{S}_t : \mathcal{N}_t^\Omega \to \mathcal{N}_t^*$,



Termgraphs

[Barendregt et al. 87]
[Plump 99, survey on *acyclic* term-graphs]

Let Ω be a set of operation symbols and $\mathcal{F}$ a set of feature symbols.
A *term-graph* t over Ω and $\mathcal{F}$ is defined by:

- a set of nodes $\mathcal{N}_t$,
- a set of edges E_t
- a subset of labeled nodes $\mathcal{N}_t^{\Omega} \subseteq \mathcal{N}_t$,
- a node labeling function $\mathcal{L}_t^n : \mathcal{N}_t^{\Omega} \to \Omega$,
- an edge labeling function $\mathcal{L}_t^e : E_t \to \mathcal{F}$
- a source function $\mathcal{S}_t : E_t \to \mathcal{N}_t$,
- a target function $\mathcal{T}_t : E_t \to \mathcal{N}_t$,

Algorithmic approach

[Barendregt et al. 87]
Shape of a rule:

$$L \to R$$

where L and R are rooted term-graphs.
A rule can be defined as one graph together with two roots

$$(L + R, r_1, r_2)$$

where r_1 and r_2 are the roots of L and R respectively
Let ρ be the rule $(L + R, r_1, r_2)$
We say that G rewrites to H using the rule ρ if

- L matches a subgraph of G ($h : L \to G\mid_n$)
- (build phase) Construct graph $G_1 = G + h(R)$
- (redirection phase) $G_2 = [h(r_1) \gg h(r_2)]G_1$
- (garbage collection phase) $H = G_2\mid_{\text{root}}$

A cumbersome definition, hard to deal with in practice!

Rewrite Rules with actions

Shape of a rewrite rule :

$$[L \mid C] \to R$$

- L is a term-graph pattern
- C is a node constraint, $\bigwedge_{i=1}^{n}(\alpha_i \not\approx \beta_i)$.
- R is a sequence of actions $a_1 ; a_2 ; \ldots ; a_n$

Actions

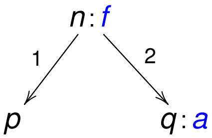
We consider three kinds of actions :

- Node definition $\alpha : f(\alpha_1, \dots, \alpha_n)$
- Edge redirection $\alpha \gg_i \beta$
- Global redirection $\alpha \gg \beta$

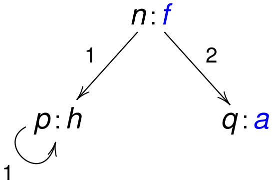
Application of actions

$a[t]$ denotes the application of action(s) a to the termgraph t

- Let $t = n : f(p, q : a)$



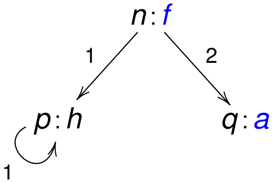
- Let $t_1 = p : h(p)[t] = n : f(p : h(p), q : a)$



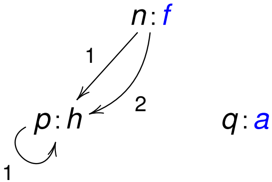
Application of actions

$a[t]$ denotes the application of action(s) a to the termgraph t

- Let $t_1 = p{:}h(p)[t] = n{:}f(p{:}h(p), q : a)$



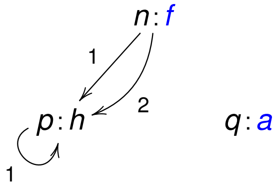
- Let $t_2 = n \gg_2 p[t_1] = n{:}f(p{:}h(p), p); q : a$



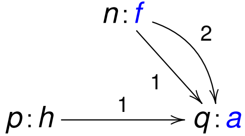
Application of actions

$a[t]$ denotes the application of action(s) a on the term-graph t

- Let $t_2 = n \gg_2 p[t_1] = n\!:\!f(p\!:\!h(p), p); q : a$



- Let $t_3 = p \gg q[t_2] = n\!:\!f(q, q); p\!:\!h(q)$



Rewrite Step

Let t be a termgraph

Let ρ be a rewrite rule $[L \mid C] \to R$

t rewrites to s at node α, $t \to_\alpha s$ iff:

- $\exists m : L \to t$ a homomorphism
- $m(root_L) = \alpha$
- α is reachable from $root_t$
- $m(C)$ holds
- $s = m(R)[t]$

Termgraph Rewrite Systems (tGRS)

Example

Length of a circular list :

$r : length(p) \to r : length'(p, p)$

$r : length'(p_1 : cons(n, p_2), p_2) \to r : s(0)$

$[r : length'(p_1 : cons(n, p_2), p_3) \mid p_2 \not\approx p_3] \to r : s(q); q : length'(p_2, p_3)$

Remark: term rewrite systems are tGRS's.

Termgraph Rewrite Systems
Example

In-situ list reversal :

$o : reverse(p) \to o : rev(p, nil)$

$o : rev(p_1 : cons(n, nil), p_2) \to p_1 \gg_2 p_2 ; o \gg p_1$

$o : rev(p_1 : cons(n, p_2 : cons(m, p_3), p_4) \to p_1 \gg_2 p_4 ; o \gg_1 p_2 ; o \gg_2 p_1$

Visual Programming would help!

DPO approach of rewrite rules with actions

A categorical approach can be found in [TERMGRAPH 06, ENTCS07, RTA07]

$$\begin{tikzcd} L \arrow[d, "m"'] & K \arrow[l, "l"'] \arrow[r, "r"] \arrow[d, "d"] & R \arrow[d, "m'"] \\ G & D \arrow[l, "l'"'] \arrow[r, "r'"] & H \end{tikzcd}$$

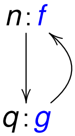
Figure: Double pushout: a rewrite step ($G \to H$)

Redirections of edges (pointers) are handled by
$K = disconnection(L, E, N)$ and the morphisms l and r.
Remark: Morphisms l and r are not injective! D is not unique!

Confluence

$f(x) \to x$
$g(x) \to x$

The following term-graph



rewrites to

Confluence

$$\alpha : f(\beta : c) \to \beta : a; \alpha \gg \beta$$

$$\alpha : g(\beta : c) \to \beta : b; \alpha \gg \beta$$

$$\begin{tikzcd}
p : f \arrow[dr] & & q : g \arrow[dl] \\
& q : c &
\end{tikzcd}$$

The label of node q may end as $q : a$ or $q : b$

Computing with non-confluent orthogonal Termgraph Rewrite Systems

How to evaluate the following termgraph ?

- $addlast(length(n : [1, 2]), n)$
- Two normal forms
  - $[1, 2, 2]$ (evaluate *addlast* after *length*)
  - $[1, 2, 3]$ (evaluate *length* after *addlast*)

Termgraphs with Priority

[PPDP06][RTA07][RTA08]

- Endow Termgraphs with priorities $(G, <_G)$ to express which node should be evaluated first
  - $m_1 : addlast(m_2 : length(n : [1, 2]), n); m_1 < m_2$
- Priorities should not be a total order (stay declarative)
- Which nodes should be ordered?
- Solution: Order only nodes producing a “side-effect”

Strategies

A strategy ϕ is a partial function which takes a rooted termgraph t and returns a node (position) n and a rule R,

$$\phi(t) = (n, R)$$

such that the termgraph t can be reduced at node n using the rule R,

$$t \rightarrow_n t'$$

Needed Nodes

Let ϕ be a rewrite strategy.
Let $\phi(t) = (p, R)$.
The node p is needed iff for all derivations

$$t \rightarrow_{\beta_1} t_1 \rightarrow_{\beta_2} \ldots t_{n-1} \rightarrow_{\beta_n} t_n$$

such that t_n is a value, there exists $i \in [1..n]$ s.t. $\beta_i = p$

Inductively sequential Term Rewrite Systems

- Constitute a subclass of TRSs for which efficient rewrite strategies are available [Antoy 92]
- Are as expressive as Strongly Sequential TRSs
- Are defined by means of data-structures called Definitional trees

Definitional Trees -case of terms-

Let $\mathcal{R}$ be the following TRS

```
f(k,nil) → R1
f(0,cons(x,l)) → R2
f(succ(n),cons(x,l)) → R3
```

A definitional tree of operator *f* is a hierarchical structure whose leaves are the rules defining *f*.

f(k, l)
- f(k, nil) $\rightarrow$ R1
- f(k, cons (x, u))
  - f(0, cons (x,u)) $\rightarrow$ R2
  - f(succ(y), cons (x,u)) $\rightarrow$ R3

Definitional trees
-case of termgraphs-

$r : length'(p_1 : nil, p_2 : \bullet) \to rhs_1$
$r : length'(p_1 : cons(n : \bullet, p_2 : \bullet), p_2) \to rhs_2$
$[r : length'(p_1 : cons(n : \bullet, p_2 : \bullet), p_3 : \bullet) \mid p_2 \dot{\neq} p_3] \to rhs_3$

A definitional tree T of the operation *length'* is given bellow:

$r : length'(p_1 : \bullet, p_2 : \bullet)$
$\quad r : length'(p_1 : nil, p_2 : \bullet) \to rhs_1$
$\quad r : length'(p_1 : cons(n : \bullet, p_3 : \bullet), p_2 : \bullet)$
$\quad\quad r : length'(p_1 : cons(n : \bullet, p_2 : \bullet), p_2) \to rhs_2$
$\quad\quad [r : length'(p_1 : cons(n : \bullet, p_2 : \bullet), p_3 : \bullet) \mid p_2 \dot{\neq} p_3] \to rhs_3$

A Rewrite strategy ϕ

Consider the following definitional tree T of the operation g :

$$
\begin{array}{l}
r : g(p_1 : \bullet, p_2 : \bullet) \\
\quad r : g(p_1 : nil, p_2 : \bullet) \to rhs_1 \\
\quad r : g(p_1 : cons(n : \bullet, p_3 : \bullet), p_2 : \bullet) \\
\quad\quad r : g(p_1 : cons(n : \bullet, p_2 : \bullet), p_2) \to rhs_2 \\
\quad\quad [r : g(p_1 : cons(n : \bullet, p_2 : \bullet), p_3 : \bullet) \mid p_2 \dot{\neq} p_3] \to rhs_3
\end{array}
$$

$$
\begin{array}{ll}
\multicolumn{2}{l}{\phi(1 : g\ (2 : g(3 : g(nil, p), q), 4 : g(nil, o)))} \\
= & \phi(2 : g(3 : g(nil, p), q)) \\
= & \phi(3 : g(nil, p)) \\
= & (3, Rule1)
\end{array}
$$

Naive extension of TRS's

Contrary to term rewriting, Definitional trees are not enough to ensure the neededness of positions computed by the strategy ϕ, in the context of term-graph rewriting.

Proposition: Let $SP = \langle \Omega, \mathcal{R} \rangle$ be tGRS such that Ω is constructor-based and the rules of every defined operation are stored in a definitional tree. Let t be a rooted termgraph. Then,

1. if $\phi(t) = (p, R)$, the node p is not needed in general.
2. if $\phi(t)$ is not defined, g can still have a constructor normal form.

Counter-examples

$r : f(p : 0) \to r \gg p$ $\qquad$ $r : h(p : 0, q : succ(n : \bullet)) \to q \gg p$

$r : f(p : succ(p' : \bullet)) \to r \gg p$

$$\text{Let } t = \quad \begin{tikzcd} n : succ \arrow[d] \\ r : succ \arrow[d] \\ p : f \arrow[d] \\ q : succ \arrow[d] \\ s : h \arrow[d] \arrow[uuu, bend right] \\ u : 0 \end{tikzcd}$$

$\phi(t) = (p, r : f(p : succ(p' : \bullet)) \to r \gg p)$.
However, the node *p* is not needed in *t*.

Counter-examples

$r : g(p : 0) \to r \gg p$ $\qquad$ $r : h(p : 0, q : succ(n : \bullet)) \to q \gg p$

Let $t =$

$$\begin{tikzcd} n : succ \arrow[d] \\ r : succ \arrow[d] \\ p : g \arrow[d] \\ q : succ \arrow[d] \\ s : h \arrow[d] \arrow[uuu, bend right] \\ u : 0 \end{tikzcd}$$

$\phi(t)$ is not defined!.
However, the termgraph t rewrites to $n : succ(u : 0)$.

Inductively Sequential Termgraph Rewrite Systems

Let $SP = \langle \Omega, \mathcal{R} \rangle$ be a tGRS.
SP is called inductively sequential iff

- The rules of every defined operation can be stored in a definitional tree and
- for all rules $[L \mid C] \to r$ in $\mathcal{R}$, for all global (respectively, local) redirections of the form $p \gg q$ (respectively, $p \gg_i q$ for some i), occurring in the right-hand side r, $p = Root_L$.

Main Properties of Strategy Φ

In presence of Inductively Sequential Termgraph Rewrite Systems

- The positions computed by Φ are needed
- Φ is c-normalizing
- Φ is c-hyper-normalizing
- Derivations computed by Φ have minimal length

Confluence

Inductively sequential tGRS are not confluent!

$$f(p : \bullet, p) \to 0$$
$$[f(p : \bullet, q : \bullet) \mid p \neq q] \to 1$$
$$r : g(q : \bullet) \to r \gg q$$

Let $t =$
$$\begin{tikzcd} & n : f \arrow[dl] \arrow[dr] & \\ p : g \arrow[rr] & & q : 0 \end{tikzcd}$$

There are two different derivations starting from t :

$$t \to_n 1$$
$$t \to_p f(q : 0, q) \to_n 0$$

Admissible termgraphs

[JICSLP98]
Ω is contructor-based, i.e. $\Omega = D \cup C$ and $D \cap C = \emptyset$
D is a set of defined operations
C is a set of constructors

A termgraph is admissible if none of its cycles includes a defined operation.

$n : succ(n)$ is an admissible termgraph
$n : +(n, n)$ and $n : tail(n)$ are not admissible

Admissible termgraphs

The set of admissible termgraphs is not closed under rewriting

$n : f(m) \to q : g(n); n \gg m$

Let $\Omega = D \cup C$ with $C = \{0, succ\}$ and $D = \{f, g\}$

$$n_1 : f(m_1 : 0) \to q_1 : g(q_1)$$

Admissible Inductively sequential Termgraph Rewrite Systems

Let $SP = \langle \Omega, \mathcal{R} \rangle$ be an inductively sequential tGRS. SP is called admissible iff for all rules $[\pi \mid C] \to r$ in $\mathcal{R}$ the following conditions are satisfied

- for all global (respectively, local) redirections of the form $p \gg q$ (respectively, $p \gg_i q$ for some i), occurring in the right-hand side r, we have $p = Root_\pi$ and $q \neq Root_\pi$.
- for all actions of the form $\alpha : f(\beta_1, \ldots, \beta_n)$, for all $i \in 1..n$, $\beta_i \neq Root_\pi$
- the set of actions of the form $\alpha : f(\beta_1, \ldots, \beta_n)$, appearing in r, do not construct a cycle including a defined operation.
- Constraint C includes disequations of the form $p \not\doteq q$ where p and q are labeled by constructor symbols.

Admissible Inductively sequential Termgraph Rewrite Systems

[ICGT08][JICSLP98]
In presence of Admissible Inductively sequential Termgraph Rewrite Systems

- The set of admissible termgraphs is closed under the rewrite relation defined by admissible rules.
- Φ computes needed positions
- Admissible termgraphs admit unique normal forms

Narrowing

Lifting optimal rewrite strategies to narrowing in the case of Admissible termgraph rewrite systems

Let $\mathcal{R}$ be the following TRS

$\leq$ `(0,y) ` $\to$ `true`

$\leq$ `(s(x),0) ` $\to$ `false`

$\leq$ `(succ(x),succ(y)) ` $\to$ $\leq$`(x,y)`

A definitional tree of operator $\leq$ is as follows:

$\leq$(i, j)

- $\leq (0, j) \to \mathit{true}$
- $\leq (s(i_1), j)$
  - $\leq (s(i_1), 0) \to \mathit{false}$
  - $\leq (s(i_1), s(j_1)) \to \leq (i_1, j_1)$

Definitional Trees -case of terms-

$\leq$(i, j)

$\leq (0, j) \rightarrow true$

$\leq (s(i_1), j)$

$\leq (s(i_1), 0) \rightarrow false$

$\leq (s(i_1), s(j_1)) \rightarrow \leq (i_1, j_1)$

How to narrow the expression $\leq (i, j + k)$?
$\leq (i, j + k) \rightsquigarrow_{j \mapsto 0} \leq (i, k) \rightsquigarrow_{i \mapsto 0} true$

Remark: The assignement $j \mapsto 0$ is useless!

The use of definitional trees prevents non necessary assignments and develops $\leq (i, j + k) \rightsquigarrow_{i \mapsto 0} true$

Key idea: Get rid of most general unifiers. Use of definitional trees to make a traversal of term (graphs) and compute only necessary

Some Results

Needed Term narrowing [POPL94][JACM2000]
Needed Graph Narrowing [JICSLP98]
Needed Collapsing Narrowing [Gratra 2000]
Narrowing-based algorithm for data-structure rewriting [ICGT06]

- Goal
  $o : equal(p : length(q), s(s(0))) = true$

- Solution : a circular list of length two
  $[q : cons(n_1, r : cons(n_2, q)) \mid q \not\approx r]$

Outline

Partial Correctness à la Hoare of Graph Rewrite Systems

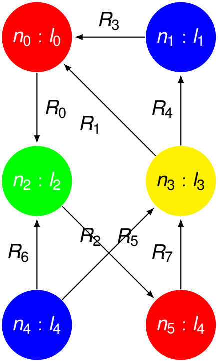
To be proven: $\{Pre(input)\}$ *Program* $\{Post(output)\}$

- Program is a graph or model transformation system
- input and output are graphs or models
- *Pre* and *Post* are formulas, of a given logic $\mathcal{L}$, over the inputs and the outputs

Logically Decorated Graphs

Let $\mathcal{L}$ be a set of formulas, a logically decorated graph G is a tuple $(N, E, \lambda_N, \lambda_E, s, t)$ where:

- N is a set of *nodes*,
- E is a set of *edges*,
- $\lambda_N : N \to 2^{\mathcal{L}}$ is a *node labeling function*,
- $\lambda_E : E \to \mathcal{L}$ is an *edge labeling function*
- source and target functions: $s : E \to N$ and $t : E \to N$



In this talk, the set $\mathcal{L}$ consists of description logic (DL) formulas.

Why considering Description Logics (DLs)?

- DLs constitute a formal basis of knowledge representation languages.
- DLs provide logical basis for ontologies.
  (E.g., the web ontology language OWL is based on DLs)
- Reasoning problems for DLs are decidable (in general)

DL Syntax

a DL syntax allows one to define:

- Concept names, which are equivalent to classical first-order logic unary predicates,
- Role names, which are equivalent to binary predicates and
- Individuals, which are equivalent to classical constants.

There are various DLs in the literature, they mainly differ by the logical operators they offer to construct concept and role expressions or axioms.

DL syntax: Concepts and roles

Let $\mathcal{C}_0$ (resp. $\mathcal{R}_0$ and $\mathcal{O}$) be a set of atomic concepts (resp. atomic roles and nominals).
Let $c_0 \in \mathcal{C}_0$, $r_0 \in \mathcal{R}_0$, $o \in \mathcal{O}$, and n an integer.

The set of concepts C and roles R are defined by:

$C := \top \mid c_0 \mid \exists R.C \mid \neg C \mid C \vee C$
$\quad \mid o$ (nominals, $\mathcal{O}$)
$\quad \mid \exists R.Self$ (self loops, $\mathcal{S}elf$)
$\quad \mid (< \ n \ R \ C)$ (counting quantifiers, $\mathcal{Q}$)

$R := r_0$
$\quad \mid U$ (universal role, $\mathcal{U}$)
$\quad \mid R^-$ (inverse role, $\mathcal{I}$)

Examples of DL logics: $\mathcal{ALC}$, $\mathcal{ALCUO}$, $\mathcal{ALCUI}$, $\ldots$

Examples of properties

Examples of some requirements about the organization of a hospital:

- All patients of a pediatrician are children:
  First-order formula:
  $\forall x, y. Pediatrician(x) \wedge Has\_patient(x, y) \Rightarrow Child(y)$
  DL formula ($\mathcal{ALCU}$): $\forall U. Pediatrician \Rightarrow \forall Has\_patient.Child$

Examples of properties

Examples of some requirements about the organization of a hospital:

- All patients of a pediatrician are children:
  First-order formula:
  $\forall x, y. Pediatrician(x) \wedge Has\_patient(x, y) \Rightarrow Child(y)$
  DL formula ($\mathcal{ALCU}$): $\forall U. Pediatrician \Rightarrow \forall Has\_patient. Child$
- Dr. Smith is a pediatrician:
  First-order formula: $\exists x. Dr.Smith = x \wedge Pediatrician(x)$
  DL formula ($\mathcal{ALCUO}$): $\exists U. Dr.Smith \wedge Pediatrician$

Examples of properties

Examples of some requirements about the organization of a hospital:

- All patients of a pediatrician are children:
  First-order formula:
  $\forall x, y. Pediatrician(x) \wedge Has\_patient(x, y) \Rightarrow Child(y)$
  DL formula ($\mathcal{ALCU}$): $\forall U. Pediatrician \Rightarrow \forall Has\_patient. Child$
- Dr. Smith is a pediatrician:
  First-order formula: $\exists x. Dr.Smith = x \wedge Pediatrician(x)$
  DL formula ($\mathcal{ALCUO}$): $\exists U. Dr.Smith \wedge Pediatrician$
- All patients are a doctor's patients:
  First-order formula:
  $\forall x, y. Patient(x) \Rightarrow Has\_patient(y, x) \wedge Doctor(y)$
  DL formula ($\mathcal{ALCUI}$):$\forall U. Patient \Rightarrow \exists Has\_patient^{-}. Doctor$

Examples of properties (Continued)

Examples of some requirements about the organization of a hospital:

1. An operation can only be associated with one operating room:
   First-order formula:
   $\forall x, y, z. Operation(x) \wedge Scheduled\_in(x, y) \wedge Scheduled\_in(x, z) \wedge Operation\_room(y) \wedge Operation\_room(z) \Rightarrow y = z$
   DL formula ($\mathcal{ALCUQ}$):
   $\forall U. Operation \Rightarrow (< 2 Scheduled\_in.Operation\_room)$

Examples of properties (Continued)

Examples of some requirements about the organization of a hospital:

1. An operation can only be associated with one operating room:
   First-order formula:
   $\forall x, y, z. Operation(x) \wedge Scheduled\_in(x, y) \wedge Scheduled\_in(x, z) \wedge Operation\_room(y) \wedge Operation\_room(z) \Rightarrow y = z$
   DL formula ($\mathcal{ALCUQ}$):
   $\forall U.Operation \Rightarrow (< 2 Scheduled\_in.Operation\_room)$
2. A doctor can not be his/her own patient:
   First-order formula: $\forall x.Doctor(x) \Rightarrow \neg Has\_patient(x, x)$
   DL formula ($\mathcal{ALCUQ}$): $\forall U.Doctor \Rightarrow \neg \exists Has\_patient.SELF$

Examples of properties (Continued)

Examples of some requirements about the organization of a hospital:

1. An operation can only be associated with one operating room:
   First-order formula:
   $\forall x, y, z. Operation(x) \wedge Scheduled\_in(x, y) \wedge Scheduled\_in(x, z) \wedge Operation\_room(y) \wedge Operation\_room(z) \Rightarrow y = z$
   DL formula ($\mathcal{ALCUQ}$):
   $\forall U. Operation \Rightarrow (< 2 Scheduled\_in. Operation\_room)$
2. A doctor can not be his/her own patient:
   First-order formula: $\forall x. Doctor(x) \Rightarrow \neg Has\_patient(x, x)$
   DL formula ($\mathcal{ALCUQ}$): $\forall U. Doctor \Rightarrow \neg \exists Has\_patient. SELF$
3. Only "private" nodes can have access to "private" nodes. Public "nodes" cannot have access to "private" nodes:
   First-order formula:
   $\forall x. y. (Has\_access(x, y)$ and $Private(y)) \Rightarrow Private(x)$
   DL formula ($\mathcal{ALCUI}$): $\forall U. Private \Rightarrow \forall Has\_access^{-}. Private$

Graph Transformation: Considered Rules

The considered Graph Rewriting rules are of the form $L \rightarrow R$ where:

- L is a graph
- R is a sequence of elementary actions

Some Elementary Actions

Let $\mathcal{C}_0$ (resp. $\mathcal{R}_0$) be a set of node (resp. edge) labels. An *elementary action*, say *a*, may be of the following forms:

- a *node addition* $add_N(i)$ (resp. *node deletion* $del_N(i)$)

Some Elementary Actions

Let $\mathcal{C}_0$ (resp. $\mathcal{R}_0$) be a set of node (resp. edge) labels. An *elementary action*, say a, may be of the following forms:

- a *node addition* $add_N(i)$ (resp. *node deletion* $del_N(i)$)
- a *node label addition* $add_C(i, c)$ (resp. *node label deletion* $del_C(i, c)$) where i is a node and c is a label in $\mathcal{C}_0$.

Some Elementary Actions

Let $\mathcal{C}_0$ (resp. $\mathcal{R}_0$) be a set of node (resp. edge) labels. An *elementary action*, say a, may be of the following forms:

- a *node addition* $add_N(i)$ (resp. *node deletion* $del_N(i)$)
- a *node label addition* $add_C(i, c)$ (resp. *node label deletion* $del_C(i, c)$) where i is a node and c is a label in $\mathcal{C}_0$.
- an *edge addition* $add_E(e, i, j, r)$ (resp. *edge deletion* $del_E(e, i, j, r)$) where e is an edge, i and j are nodes and r is an edge label in $\mathcal{R}_0$.

Some Elementary Actions

Let $\mathcal{C}_0$ (resp. $\mathcal{R}_0$) be a set of node (resp. edge) labels. An *elementary action*, say a, may be of the following forms:

- a *node addition* $add_N(i)$ (resp. *node deletion* $del_N(i)$)
- a *node label addition* $add_C(i, c)$ (resp. *node label deletion* $del_C(i, c)$) where i is a node and c is a label in $\mathcal{C}_0$.
- an *edge addition* $add_E(e, i, j, r)$ (resp. *edge deletion* $del_E(e, i, j, r)$) where e is an edge, i and j are nodes and r is an edge label in $\mathcal{R}_0$.
- a *global edge redirection* $i \gg j$ where i and j are nodes. It redirects all incoming edges of i towards j.

Some Elementary Actions

Let $\mathcal{C}_0$ (resp. $\mathcal{R}_0$) be a set of node (resp. edge) labels. An *elementary action*, say *a*, may be of the following forms:

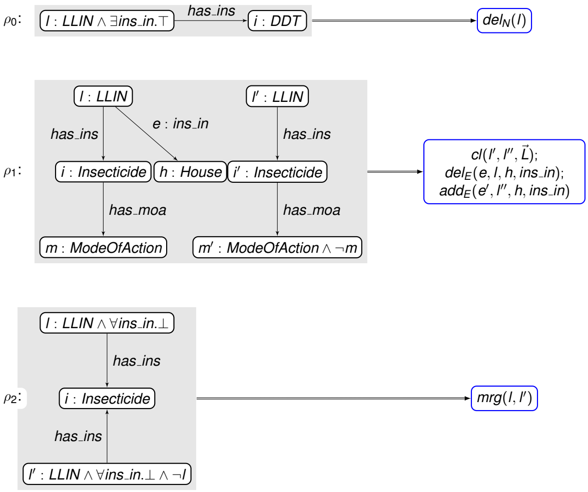
- a *node addition* $add_N(i)$ (resp. *node deletion* $del_N(i)$)
- a *node label addition* $add_C(i, c)$ (resp. *node label deletion* $del_C(i, c)$) where i is a node and c is a label in $\mathcal{C}_0$.
- an *edge addition* $add_E(e, i, j, r)$ (resp. *edge deletion* $del_E(e, i, j, r)$) where e is an edge, i and j are nodes and r is an edge label in $\mathcal{R}_0$.
- a *global edge redirection* $i \gg j$ where i and j are nodes. It redirects all incoming edges of i towards j.
- a *merge action* $mrg(i, j)$ where i and j are nodes.

Some Elementary Actions

Let $\mathcal{C}_0$ (resp. $\mathcal{R}_0$) be a set of node (resp. edge) labels. An *elementary action*, say a, may be of the following forms:

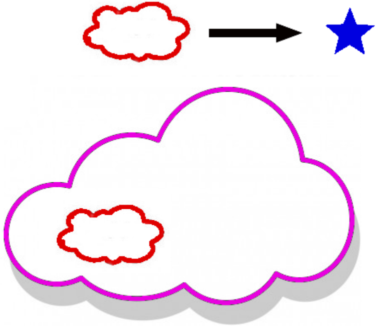
- a *node addition* $add_N(i)$ (resp. *node deletion* $del_N(i)$)
- a *node label addition* $add_C(i, c)$ (resp. *node label deletion* $del_C(i, c)$) where i is a node and c is a label in $\mathcal{C}_0$.
- an *edge addition* $add_E(e, i, j, r)$ (resp. *edge deletion* $del_E(e, i, j, r)$) where e is an edge, i and j are nodes and r is an edge label in $\mathcal{R}_0$.
- a *global edge redirection* $i \gg j$ where i and j are nodes. It redirects all incoming edges of i towards j.
- a *merge action* $mrg(i, j)$ where i and j are nodes.
- a *clone action* $cl(i, j, L_{in}, L_{out}, L_{l\_in}, L_{l\_out}, L_{l\_loop})$ where i and j are nodes and L_{in}, L_{out}, L_{l_in}, L_{l_out} and L_{l_loop} are subsets of $\mathcal{R}_0$. It clones a node i by creating a new node j and connects j to the rest of a host graph according to different information given in the parameters $L_{in}, L_{out}, L_{l\_in}, L_{l\_out}, L_{l\_loop}$.

Graph Rewrite Systems: Example



Match

- To be able to apply rules, we need to define when they can be applied.



Match

Definition: Match

A *match* h between a lhs L and a graph G is a pair of functions $h = (h^N, h^E)$, with $h^N : N^L \to N^G$ and $h^E : E^L \to E^G$ such that:

1. $\forall e \in E^L, s^G(h^E(e)) = h^N(s^L(e))$
2. $\forall e \in E^L, t^G(h^E(e)) = h^N(t^L(e))$
3. $\forall n \in N^L, \forall c \in \lambda_N^L(n), h^N(n) \models c$
4. $\forall e \in E^L, \lambda_E^G(h^E(e)) = \lambda_E^L(e)$

Remark: The third condition says that for every node, n, of the lhs, the node to which it is associated, $h(n)$, in G has to satisfy every concept in $\lambda_N^L(n)$. This condition clearly expresses additional negative and positive conditions which are added to the "structural" pattern matching.

Rewrite Step and Rewrie Derivation

Rewrite step

Let $\rho = L \to R$ be a rule and G and G' be two graphs.
G rewrites into G' using rule ρ, noted $G \to_\rho G'$ iff:

- There exists a match h from the left-hand side L to G, and
- $G \rightsquigarrow_{h(R)} G'$. I.e., G' is the result of performing $h(R)$ on G

Rewrite derivation

Let $\mathcal{R}$ be graph transformation system and G and G' be two graphs.
A *rewrite derivation* from G to G', noted $G \to_\mathcal{R} G'$, is a sequence $G \to_{\rho_0} G_1 \to_{\rho_1} \ldots \to_{\rho_n} G'$ such that $\forall i. \rho_i \in \mathcal{R}$.

Strategies

- A strategy is a word of the following language defined by $s ::=$
  - ρ (application of a rule)
  - $s; s$ (sequential composition of strategies)
  - $s \oplus s$ (non-deterministic choice between two strategies)
  - s^* (iteration as long as possible of a strategy)
  - $\ldots$

Strategies

- A strategy is a word of the following language defined by $s ::=$
  - ρ (application of a rule)
  - $s; s$ (sequential composition of strategies)
  - $s \oplus s$ (non-deterministic choice between two strategies)
  - s^* (iteration as long as possible of a strategy)
  - $\ldots$
- Example: Strategy $strat = s_0; s_1^*; s_2$ performs once the sub-strategy s_0, iterates as much as possible sub-strategy s_1, before performing once sub-strategy s_2.

Strategies

- A strategy is a word of the following language defined by $s ::=$
  - ρ (application of a rule)
  - $s; s$ (sequential composition of strategies)
  - $s \oplus s$ (non-deterministic choice between two strategies)
  - s^* (iteration as long as possible of a strategy)
  - $\ldots$
- Example: Strategy $strat = s_0; s_1^*; s_2$ performs once the sub-strategy s_0, iterates as much as possible sub-strategy s_1, before performing once sub-strategy s_2.
- A derivation $G \rightarrow_{\rho_0} G_1 \rightarrow_{\rho_1} \ldots \rightarrow_{\rho_n} G'$ is controlled by a strategy *strat* iff the word $\rho_0 \rho_1 \ldots \rho_n$ belongs to the language defined by strategy *strat*.

Specification and Correctness

A specification *spec* is a triple $(Pre, strat, Post)$ where:

- *Pre* is a DL formula called the *precondition*
- *strat* is a strategy with respect to a graph transformation system $\mathcal{R}$
- *Post* is a DL formula called the *postcondition*.

A specification $spec = (Pre, strat, Post)$ is said to be correct iff:

- for all graphs G,
- for all graphs G' such that $G \rightarrow_{strat} G'$
- if $G \models Pre$ then $G' \models Post$

Floyd-Hoare Logics

- Let $\mathcal{R}$ be a graph transformation system
- Let *strat* be a strategy and $\rho_0 \ldots \rho_{n-1}\rho_n$ an element of *strat*
- Let *Pre* and *Post* be two DL formulas
- Aim: Prove that specification $spec = (Pre, strat, Post)$ is correct

Pre

ρ_0;

...

ρ_{n-1};

ρ_n;

Post

Floyd-Hoare Logics

- Let $\mathcal{R}$ be a graph transformation system
- Let *strat* be a strategy and $\rho_0 \ldots \rho_{n-1}\rho_n$ an element of *strat*
- Let *Pre* and *Post* be two DL formulas
- Aim: Prove that specification $spec = (Pre, strat, Post)$ is correct

Pre
a_0;
...

a_{m-1};

a_m;
Post

Floyd-Hoare Logics

- Let $\mathcal{R}$ be a graph transformation system
- Let *strat* be a strategy and $\rho_0 \ldots \rho_{n-1}\rho_n$ an element of *strat*
- Let *Pre* and *Post* be two DL formulas
- Aim: Prove that specification $spec = (Pre, strat, Post)$ is correct

Pre

a_0;

...

a_{m-1};

$Post[a_m]$

a_m;

Post

Floyd-Hoare Logics

- Let $\mathcal{R}$ be a graph transformation system
- Let *strat* be a strategy and $\rho_0 \ldots \rho_{n-1}\rho_n$ an element of *strat*
- Let *Pre* and *Post* be two DL formulas
- Aim: Prove that specification $spec = (Pre, strat, Post)$ is correct

Pre

a_0;

...

$Post[a_m][a_{m-1}]$

a_{m-1};

$Post[a_m]$

a_m;

Post

Floyd-Hoare Logics

- Let $\mathcal{R}$ be a graph transformation system
- Let *strat* be a strategy and $\rho_0 \ldots \rho_{n-1}\rho_n$ an element of *strat*
- Let *Pre* and *Post* be two DL formulas
- Aim: Prove that specification $spec = (Pre, strat, Post)$ is correct

$Pre \Rightarrow Post[a_m][a_{m-1}]...[a_0]$
a_0;
...
$Post[a_m][a_{m-1}]$
a_{m-1};
$Post[a_m]$
a_m;
Post

Substitutions

Definition: Substitution

A *substitution*, written $[a]$, is associated to each elementary action a, such that for all graphs G and DL formulas ϕ, $(G \models \phi[a]) \Leftrightarrow (G' \models \phi)$ where G' is obtained from G after application of action a,i.e., $G \rightsquigarrow_a G'$.

G	$\rightsquigarrow_a$	G'
$\phi[a]$		ϕ

Generating Weakest Preconditions

We define $wp(a, Q)$ the weakest precondition for an elementary action a and a formula Q.

¡$wp(a,\ Q) = Q[a]$

Generating Weakest Preconditions

We define $wp(a, Q)$ the weakest precondition for an elementary action a and a formula Q.

- $wp(a,\ Q) = Q[a]$ How to handle substitutions?

Floyd-Hoare Logics: a classical example
The assignment instruction (action)

Weakest precondition: $wp(x := X + 1, Post) \equiv x > 5[x := X + 1]$

Action: $x := x + 1$;

Post: $Post \equiv x > 5$

Floyd-Hoare Logics: a classical example
The assignment instruction (action)

$wp(x := X + 1, Post) \equiv x > 5[x := X + 1] \equiv x > 4$

Action: $x := x + 1$;

Post: $Post \equiv x > 5$

Floyd-Hoare Logics: a basic case

$wp(Add_E(e, a, b, R), Post) \equiv \exists U.(a \wedge (> 5R.\top))[Add_E(e, a, b, R)]$

Action: $Add_E(e, a, b, R)$;

Post: $\exists U.(a \wedge (> 5R.\top))$

Floyd-Hoare Logics: a basic case

$wp(Add_E(e, a, b, R), Post) \equiv \exists U.(a \wedge (> 5R.\top))[Add_E(e, a, b, R)] \equiv$
$(\exists U.(a \wedge \exists R.b) => \exists U.(a \wedge (> 5R.\top))) \wedge$
$(\exists U.(a \wedge \forall R.\neg b) => \exists U.(a \wedge (> 4R.\top)))$

Action: $Add_E(e, a, b, R)$;

Post: $\exists U.(a \wedge (> 5R.\top))$

Closure Under Substitutions

A logic $\mathcal{L}$ is said to be closed under substitution iff for every formula $\phi \in \mathcal{L}$, every substitution $[a]$, $\phi[a] \in \mathcal{L}$.

DLs and Closure Under Substitutions

Theorem: The description logics $\mathcal{ALCUO}$, $\mathcal{ALCUOI}$, $\mathcal{ALCQUOI}$, $\mathcal{ALCUOS}elf$, $\mathcal{ALCUOIS}elf$, and $\mathcal{ALCQUOIS}elf$ are closed under substitutions.

Theorem: The description logics $\mathcal{ALCQUO}$ and $\mathcal{ALCQUOS}elf$ are not closed under substitutions.

Generating Weakest Preconditions (continued)

We define $wp(strat, Q)$ the weakest precondition for a strategy *strat* and a formula Q.

- $wp(s_0; s_1,\ Q) = wp(s_0, wp(s_1,\ Q))$
- $wp(s_0 \oplus s_1,\ Q) = wp(s_0, Q) \wedge wp(s_1, Q)$
- $wp(\rho,\ Q) = App(\rho) \Rightarrow Q[a_n]...[a_0]$ where ρ's right-hand side is $a_0; ...; a_n$

Generating Weakest Preconditions

We define $wp(strat, Q)$ the weakest precondition for a strategy *strat* and a formula Q.

- $wp(\rho,\ Q) = App(\rho) \Rightarrow Q[a_n]...[a_0]$

Definition: Application Condition

Given a rule ρ, the *application condition* $App(\rho)$ is a formula such that a graph $G \models App(\rho)$ iff there exists a match between the left-hand side of ρ and G

Generating Weakest Preconditions

We define $wp(strat, Q)$ the weakest precondition for a strategy *strat* and a formula Q.

- $wp(a,\ Q) = Q[a]$
- $wp(\epsilon,\ Q) = Q$
- $wp(a; \alpha,\ Q) = wp(a, wp(\alpha, Q))$
- $wp(s_0; s_1,\ Q) = wp(s_0, wp(s_1,\ Q))$
- $wp(s_0 \oplus s_1,\ Q) = wp(s_0, Q) \wedge wp(s_1, Q)$
- $wp(\rho,\ Q) = App(\rho) \Rightarrow Q[a_n]...[a_0]$

Generating Weakest Preconditions

$wp(strat, Q)$ computes the weakest precondition for a strategy *strat* and a formula Q.

- $wp(a,\ Q) = Q[a]$
- $wp(\epsilon,\ Q) = Q$
- $wp(a; \alpha,\ Q) = wp(a, wp(\alpha, Q))$
- $wp(s_0; s_1,\ Q) = wp(s_0, wp(s_1,\ Q))$
- $wp(s_0 \oplus s_1,\ Q) = wp(s_0, Q) \wedge wp(s_1, Q)$
- $wp(\rho,\ Q) = App(\rho) \Rightarrow Q[a_n]...[a_0]$
- $wp(s^*,\ Q) = inv_s$

Verification Conditions

- $vc(\rho,\ Q) = \top$
- $vc(s_0; s_1,\ Q) = vc(s_0, wp(s_1,\ Q)) \wedge vc(s_1, Q)$
- $vc(s_0 \oplus s_1,\ Q) = vc(s_0, Q) \wedge vc(s_1, Q)$
- $vc(s^*,\ Q) =$
  $(inv_s \wedge \neg App(s) \Rightarrow Q) \wedge (inv_s \wedge App(s) \Rightarrow wp(s, inv_s)) \wedge vc(s, inv_s)$

Soundness of the verification

Let $spec = (Pre, strat, Post)$ be a specification. We call correctness formula the formula
$correct(spec) = (Pre \Rightarrow wp(strat, Post)) \wedge vc(strat, Post)$.

Theorem:
If $correct(spec)$ is valid, then for all graphs G, G' such that $G \rightarrow_{strat} G'$, $G \models Pre$ implies $G' \models Post$.

Decidability of the verification

Theorem:
Let $spec = (Pre, strat, Post)$ be a specification using one of the following DL logics $\mathcal{ALCUO}$, $\mathcal{ALCUOI}$, $\mathcal{ALCQUOI}$, $\mathcal{ALCUOS}elf$, $\mathcal{ALCUOIS}elf$, and $\mathcal{ALCQUOIS}elf$. Then, the correctness of $spec$ is decidable.

Other considered decidable logics

- Extension of the dynamic logic PDL: C2PDL
- First-order Logic : fragments $\exists^*\forall^*$ and $\mathcal{C}^2$

Conclusion

- Conferences and workshops: ICGT, ICMT, GCM, etc.
- Various Algebraic Approaches: DPO, SPO, SqPO,AGREE, PBPO, etc.
- Various Implementations: AGG, GROOVE, GP, PORGY, PROGRES, etc.
- General Framework: (Weak) Adhesive (HLR) Categories
- Other issues: Parallelism, Verification Techniques, Termination...