# Tutorial on
# Fundamentals of Algebraic Graph Transformation

Hartmut Ehrig, Ulrike Prange : TU Berlin

Karsten Ehrig : U. Leicester

July 15, 2006

Based on
FAGT-Book, EATCS Monographs in TCS, Springer 2006

## Overview





Part I

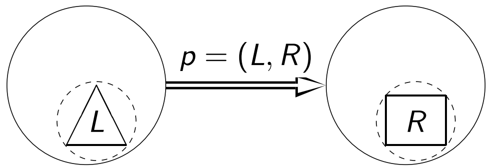
## General Introduction and Motivation





## What is Graph Transformation?

Rule-based modification of graphs




- from Chomsky grammars on strings to graph grammars;
- from term rewriting to graph rewriting;
- from textual description to visual modeling.

## Aims and Paradigms of Graph Transformation

Computing by graph transformation is a fundamental concept for

- visual modeling and specification,
- model transformation,
- concurrency and distribution,
- software development.

**Projects**

Supported by EU projects since 1990:
COMPUGRAPH 1 + 2, GETGRATS, APPLIGRAPH, SEGRAVIS.



## Graph Transformation Approaches

1. Node label replacement approach
2. Hyperedge replacement approach
3. Algebraic approach
   - Double Pushout (DPO) (since 1973)
   - Single Pushout (SPO) (since 1984/90)
   - High Level Replacement (HLR) (since 1991/2004)
4. Logical approach
5. Theory of 2-structures
6. Programmed graph replacement approach

**Literature**

Handbook of Graph Grammars and Computing by Graph Transformation, Vol. 1 (1997)



## Application Areas of Graph Transformation Systems

Application areas are

- model and program transformation;
- syntax and semantics of visual languages;
- visual modeling of behavior and programming;
- modeling, metamodeling, and model-driven architecture;
- software architectures and evolution;
- refactoring of programs and software systems;
- security policies.

**Literature**

Handbook of Graph Grammars and Computing by Graph Transformation, Vol. 2 (1999)
Proc. ICGT '02-06, LNCS



# Part II

## Category of Graphs and Gluing Construction

# Graphs and Graph Morphisms

## Definition

$$\begin{tikzcd} E_1 \arrow[r, shift left, "s_1"] \arrow[r, shift right, "t_1"'] \arrow[d, "f_E"'] \arrow[dr, phantom, "="] & V_1 \arrow[d, "f_V"] \\ E_2 \arrow[r, shift left, "s_2"] \arrow[r, shift right, "t_2"'] & V_2 \end{tikzcd}$$

graph $G = (V, E, s, t)$ consists of

- sets $V$ of *nodes* and $E$ of *edges*, and
- *source* and *target* functions $s, t : E \to V$

graph morphism $f : G_1 \to G_2$, $f = (f_V, f_E)$ consists of two functions $f_V : V_1 \to V_2$ and $f_E : E_1 \to E_2$ with

$$f_V \circ s_1 = s_2 \circ f_E \text{ and } f_V \circ t_1 = t_2 \circ f_E$$

composition $g \circ f = (g_V \circ f_V, g_E \circ f_E) : G_1 \to G_3$ of $f : G_1 \to G_2$ and $g : G_2 \to G_3$ is graph morphism



# Example: Graphs and Graph Morphisms

## Example

**Top:** graph $G_S = (V_S, E_S, s_S, t_S)$, with

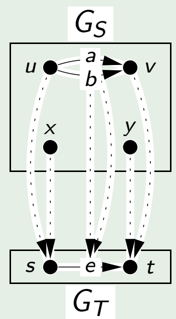
- $V_S = \{u, v, x, y\}$, $E_S = \{a, b\}$,
- $s_S : E_S \to V_S : a, b \mapsto u$,
  $t_S : E_S \to V_S : a, b \mapsto v$

**Bottom:** graph $G_T = (V_T, E_T, s_T, t_T)$, with

- $V_T = \{s, t\}$, $E_S = \{e\}$,
- $s_T : E_T \to V_T : e \mapsto s$,
  $t_T : E_T \to V_T : e \mapsto t$

**Inbetween:** graph morphism $f : G_S \to G_T$ with

- $f_V : V_S \to V_T : u, x \mapsto s; v, y \mapsto t$
- $f_E : E_S \to E_T : a, b \mapsto e$






# Typed Graphs and Typed Graph Morphisms

## Definition

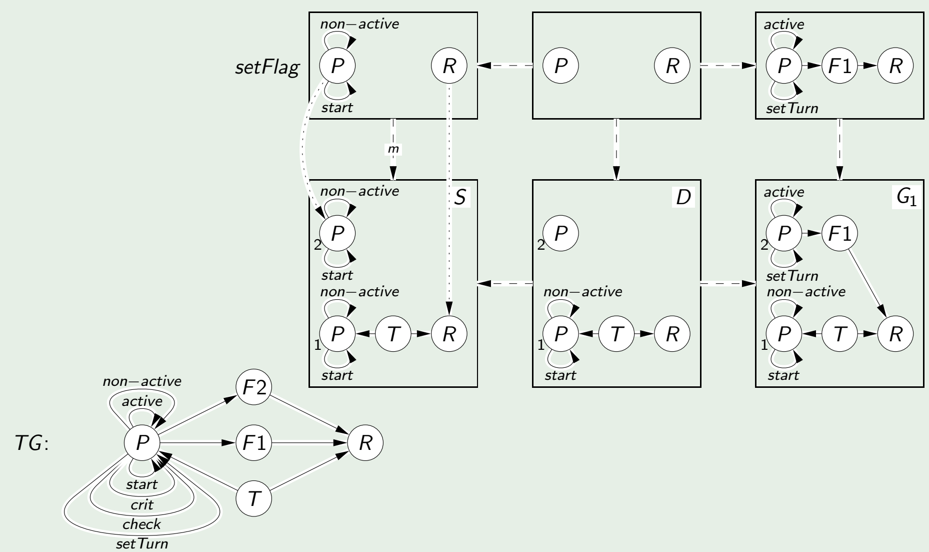
- type graph is a distinguished graph $TG = (V_{TG}, E_{TG}, s_{TG}, t_{TG})$
- typed graph $G^T = (G, type)$ over $TG$ is graph $G$ and graph morphism $type : G \to TG$
- typed graph morphism $f : G_1^T \to G_2^T$ is graph morphism $f : G_1 \to G_2$ with $type_2 \circ f = type_1$

$$\begin{tikzcd} G_1 \arrow[rr, "f"] \arrow[dr, "type_1"'] & \arrow[d, phantom, "="] & G_2 \arrow[dl, "type_2"] \\ & TG & \end{tikzcd}$$



# Example: Mutual Exclusion and Gluing Construction

## Example



## Notion of Category

### Definition

category $\mathbf{C} = (Ob_C, Mor_C, \circ, id)$ defined by

- class $Ob_C$ of objects;
- set $Mor_C(A, B)$ of morphisms $\forall A, B \in Ob_C$;
- composition $\circ : Mor_C(B, C) \times Mor_C(A, B) \to Mor_C(A, C)$ $\forall A, B, C \in Ob_C$,
- identity morphism $id_A \in Mor_C(A, A)$ $\forall A \in Ob_C$;

such that:

1. Associativity. $(h \circ g) \circ f = h \circ (g \circ f) \quad \forall A \xrightarrow{f} B \xrightarrow{g} C \xrightarrow{h} D$
2. Identity. $f \circ id_A = f$ and $id_B \circ f = f \quad \forall f : A \to B$

for $f \in Mor_C(A, B)$, we write $f : A \to B$



## Examples of Categories

1. category **Sets**
   - objects: sets, morphisms: functions $f : A \to B$,
   - composition for $f : A \to B$ and $g : B \to C$ defined by $(g \circ f)(x) = g(f(x))$,
   - identity: identical mapping $id_A : A \to A : x \mapsto x$.
2. category **Graphs**
   - objects: graphs, morphisms: graph morphisms,
   - composition and identity: componentwise on nodes and edges.
3. category $\mathbf{Graphs_{TG}}$
   - objects: typed graphs, morphisms: typed graph morphisms,
   - composition and identity: as in **Graphs**.
4. category $\mathbf{Alg}(\boldsymbol{\Sigma})$
   - objects: algebras over a given signature $\Sigma$, morphisms: homomorphisms
   - composition and identity: componentwise on the carrier sets.



## Mono-, Epi- and Isomorphisms

### Definition

$m : B \to C \in Mor_C$ is monomorphism if

$$m \circ f = m \circ g \text{ implies } f = g \quad \forall f, g : A \to B \in Mor_C$$

$$A \underset{g}{\overset{f}{\rightrightarrows}} B \xrightarrow{m} C \qquad A \xrightarrow{e} B \underset{g}{\overset{f}{\rightrightarrows}} C$$

$e : A \to B \in Mor_C$ is epimorphism if

$$f \circ e = g \circ e \text{ implies } f = g \quad \forall f, g : B \to C \in Mor_C$$

$i : A \to B \in Mor_C$ is isomorphism if
there is $i^{-1} : B \to A \in Mor_C$ with $i \circ i^{-1} = id_B$ and $i^{-1} \circ i = id_A$

### Fact

*In* **Sets**, **Graphs**, *and* $\mathbf{Graphs_{TG}}$ *the monomorphisms (or epimorphisms or isomorphisms) are exactly those morphisms which are injective (or surjective or bijective, respectively).*



## Pushouts as Gluing Construction

### Definition

pushout (PO) over morphisms
$f : A \to B$ and $g : A \to C$ defined by

- a pushout object $D$ and
- morphisms $f' : C \to D$ and $g' : B \to D$ with $f' \circ g = g' \circ f$

$$\begin{tikzcd}
A \arrow[r, "f"] \arrow[d, "g"'] \arrow[dr, phantom, "="] & B \arrow[d, "g'"] \arrow[ddr, bend left, "h"] & \\
C \arrow[r, "f'"'] \arrow[drr, bend right, "k"'] & D \arrow[dr, "x"] & \\
 & & X
\end{tikzcd}$$

with *universal property*:
For all objects $X$, morphisms $h : B \to X$, $k : C \to X$ with $k \circ g = h \circ f$:
there is a unique morphism $x : D \to X$ with $x \circ g' = h$ and $x \circ f' = k$

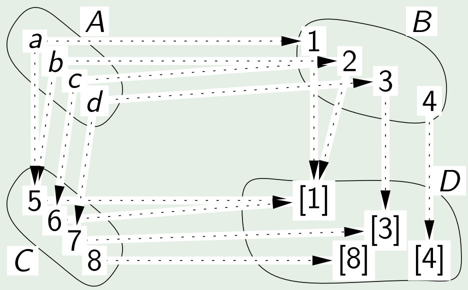
The pushout construction is unique up to isomorphism.

# Pushouts in **Sets**

## Fact

*In* **Sets**, *pushout* $C \xrightarrow{f'} D \xleftarrow{g'} B$ *over* $f : A \to B$ *and* $g : A \to C$ *is*
$D = B \dot{\cup} C|_{\equiv}$, *with* $\equiv$ *generated by* $(f(a), g(a)) \in \equiv \quad \forall a \in A$ *and*
$f'(c) = [c] \quad \forall c \in C$ *and* $g'(b) = [b] \quad \forall b \in B$.

## Example




$(1,5), (2,5), (2,6), (3,7) \in \equiv$

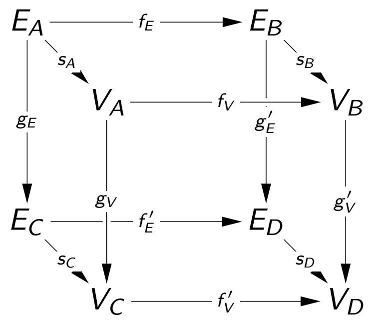
$[1] = [2] = [5] = [6] = \{1,2,5,6\}$,
$[3] = [7] = \{3,7\}$,
$[4] = \{4\}$, $[8] = \{8\}$



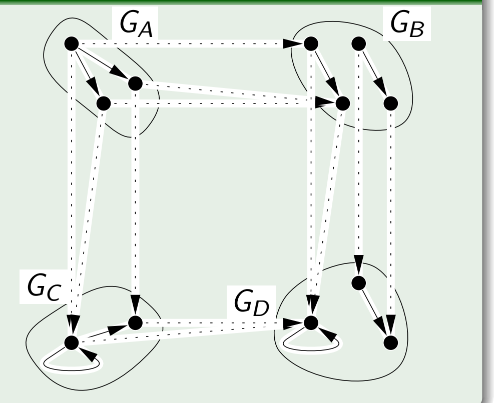
# Pushouts in **Graphs** and **Graphs$_{TG}$**

## Fact

*In* **Graphs** *and* **Graphs$_{TG}$**, *pushouts can be constructed componentwise for nodes and edges in* **Sets**.




## Example





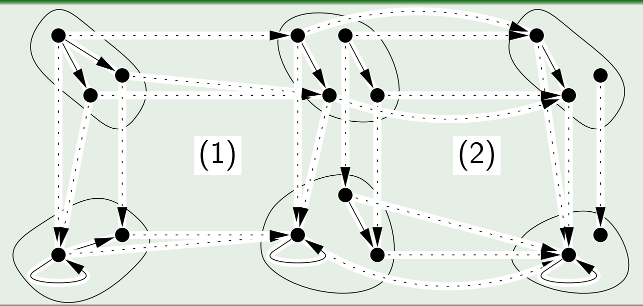

# Composition and Decomposition of Pushouts

## Fact

- *PO composition:*
  *(1) and (2) POs* $\Rightarrow$ *(1) + (2) PO*
- *PO decomposition:*
  *(1) and (1) + (2) POs* $\Rightarrow$ *(2) PO*

$$\begin{tikzcd} A \arrow[r, "f"] \arrow[d, "g"'] \arrow[dr, phantom, "(1)"] & B \arrow[r, "e"] \arrow[d, "g'"] \arrow[dr, phantom, "(2)"] & E \arrow[d, "e''"] \\ C \arrow[r, "f'"'] & D \arrow[r, "e'"'] & F \end{tikzcd}$$

## Example






# Pullbacks as Dual Construction of Pushouts

## Definition

pullback (PB) over morphisms $f : C \to D$ and $g : B \to D$ defined by

- a pullback object $A$ and
- morphisms $f' : A \to B$ and $g' : A \to C$ with $g \circ f' = f \circ g'$

$$\begin{tikzcd} X \arrow[dr, "x"] \arrow[drr, bend left, "h"] \arrow[ddr, bend right, "k"'] & & \\ & A \arrow[r, "f'"] \arrow[d, "g'"'] \arrow[dr, phantom, "="] & B \arrow[d, "g"] \\ & C \arrow[r, "f"'] & D \end{tikzcd}$$

with *universal property*:
For all objects $X$, morphisms $h : X \to B$, $k : X \to C$ with $f \circ k = g \circ h$:
there is a unique morphism $x : X \to A$ with $f' \circ x = h$ and $g' \circ x = k$.

The composition and decomposition of pullbacks is dual to pushouts.

## Pullbacks in **Sets**, **Graphs** and **Graphs**$_{\mathbf{TG}}$

### Fact

*In* **Sets**, *pullback* $C \xleftarrow{g'} A \xrightarrow{f'} B$ *over* $f : C \to D$ *and* $g : B \to D$:
$A = \bigcup_{d \in D} f^{-1}(d) \times g^{-1}(d) = \{(c, b) \mid f(c) = g(b)\} \subseteq C \times B$
*with* $f' : A \to B : (c, b) \mapsto b$ *and* $g' : A \to C : (c, b) \mapsto c$.

*Special case:* $A \overset{\sim}{=} B \cap C$ *for inclusions* $f$ *and* $g$.

*In* **Graphs** *and* **Graphs**$_{\mathbf{TG}}$*, pullbacks can be constructed componentwise for nodes and edges in* **Sets**.



Part III

## Graph Transformation Systems





## Graph Productions and Transformations

### Definition

(typed) graph production
$p = (L \xleftarrow{l} K \xrightarrow{r} R)$ consists of

- (typed) graphs $L$, $K$, and $R$,
- injective (typed) graph morphisms $l$ and $r$

$$\begin{tikzcd} L \arrow[d, "m"'] \arrow[dr, phantom, "(1)"] & K \arrow[l, "l"'] \arrow[r, "r"] \arrow[d, "k"] \arrow[dr, phantom, "(2)"] & R \arrow[d, "n"] \\ G & D \arrow[l, "f"] \arrow[r, "g"'] & H \end{tikzcd}$$

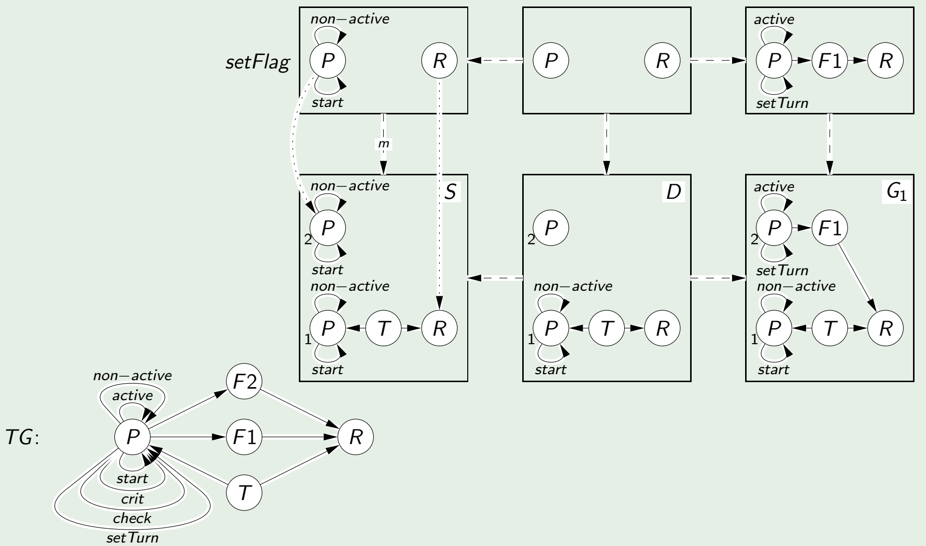
direct (typed) graph transformation $G \overset{p,m}{\Longrightarrow} H$ given by POs (1) and (2), production $p$ and match $m : L \to G$

(typed) graph transformation, $G_0 \overset{*}{\Rightarrow} G_n$, is sequence $G_0 \Rightarrow G_1 \Rightarrow \ldots \Rightarrow G_n$ of direct (typed) graph transformations



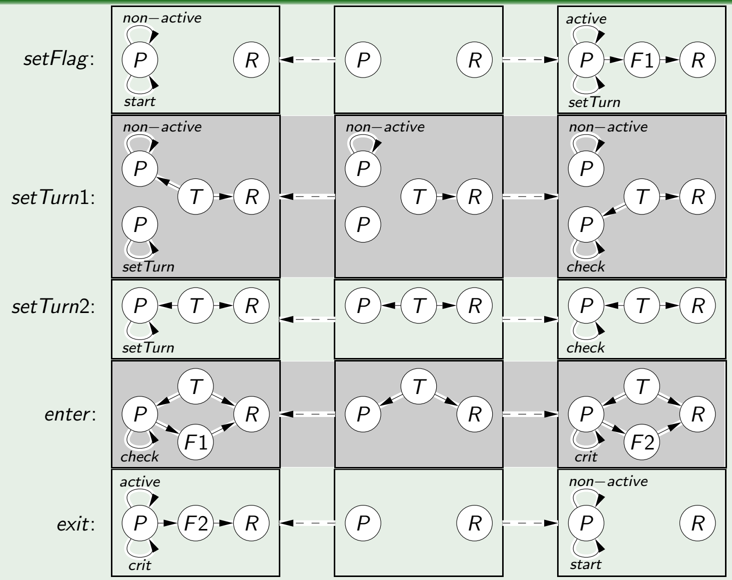
## Recap: Example Mutual Exclusion

### Example



# Example: Mutual Exclusion 2

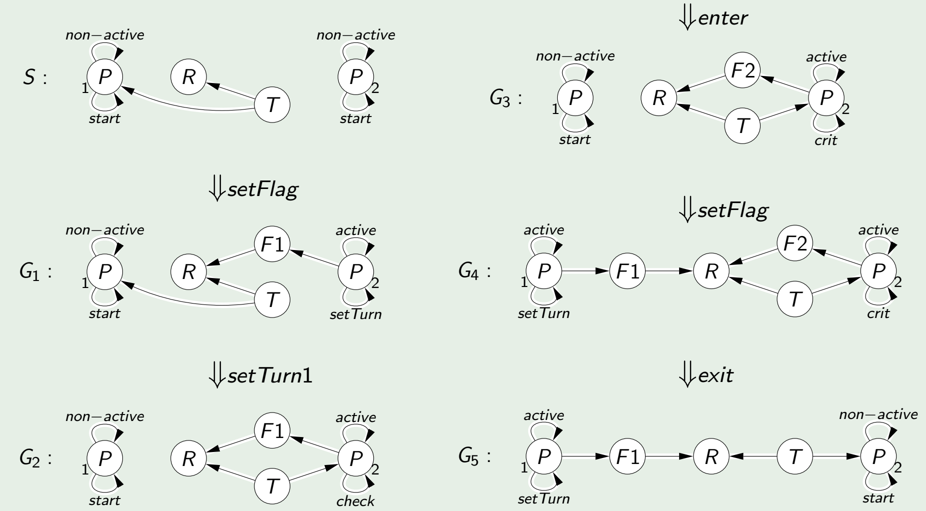
## Example






# Example: Mutual Exclusion 3

## Example






# Graph Transformation Systems, Grammars and Languages

## Definition

- (typed) graph transformation system $GTS = (TG, P)$ consists of (type graph $TG$ and) set of (typed) graph productions $P$
- (typed) graph grammar $GG = (GTS, S)$ consists of $GTS$ and (typed) start graph $S$
- (typed) graph language $L$ of $GG$ is defined by

$$L = \{G \mid \exists \text{ (typed) graph transformation } S \overset{*}{\Rightarrow} G\}.$$

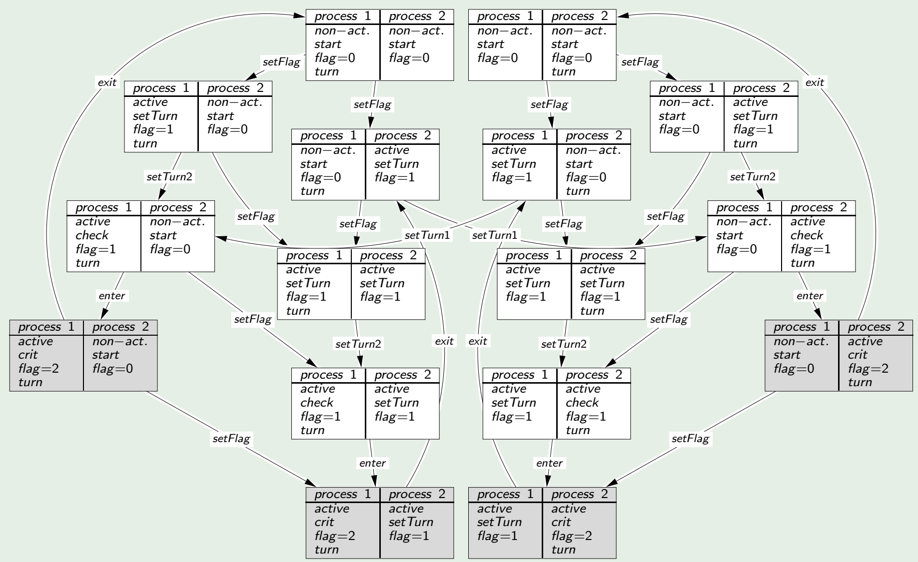
## Example

- typed graph transformation system $ME = (TG, P)$
- typed graph grammar $MutualExclusion = (TG, P, S)$
- $P = \{setFlag, setTurn1, setTurn2, enter, exit\}$.



# Example: Language of Mutual Exclusion Grammar

## Example



# Applicability and Gluing Condition

Problem: When is production $p$ applicable via match $m$?

## Definition

production $p$ is applicable to $G$ via match $m$ if context graph $D$ exists such that (1) is PO

$$\begin{tikzcd} L \arrow[d, "m"'] & K \arrow[l, "l"'] \arrow[r, "r"] \arrow[d, "k"] \arrow[dl, phantom, "(1)"] & R \\ G & D \arrow[l, "f"] \end{tikzcd}$$

Gluing condition: $IP \cup DP \subseteq GP$

- gluing points $GP = l_V(V_K) \cup l_E(E_K) = l(K)$
- identification points $IP =$
  $\{v \in V_L \mid \exists w \in V_L, w \neq v : m_V(v) = m_V(w)\} \cup \{e \in E_L \mid \exists f \in E_L, f \neq e : m_E(e) = m_E(f)\}$
- dangling points $DP =$
  $\{v \in V_L \mid \exists e \in E_G \backslash m_E(E_L) : s_G(e) = m_V(v) \text{ or } t_G(e) = m_V(v)\}$

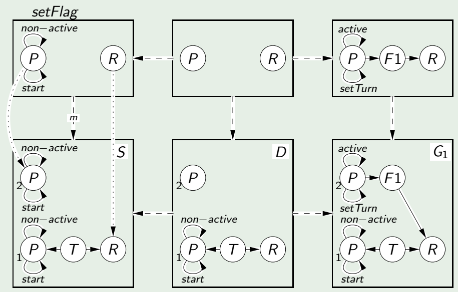
## Fact

*Context graph $D$ with the PO (1) exists iff the gluing condition is satisfied. If $D$ exists, it is unique up to isomorphism.*



# Example and Counterexample: Gluing Condition

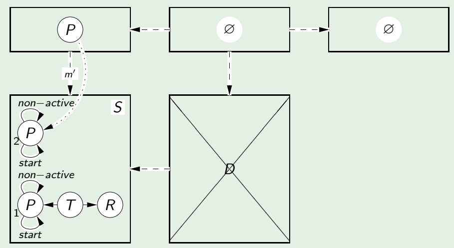
## Example




- $GP = l(K)$: both nodes in $L$,
- $IP = \emptyset$,
- $DP =$ node labelled $R$.

gluing condition is satisfied with $IP \cup DP \subseteq GP$

## Counterexample




- $GP = l(K) = \emptyset$,
- $IP = \emptyset$,
- $DP =$ node labeled $P$.

gluing condition is not satisfied, because $DP \not\subseteq GP$



# Construction of Direct Graph Transformations

## Fact

*If gluing condition is satisfied, $G \overset{p,m}{\Rightarrow} H$ can be constructed in two steps:*

$$\begin{tikzcd} L \arrow[d, "m"'] & K \arrow[l, "l"'] \arrow[r, "r"] \arrow[d, "k"] \arrow[dl, phantom, "(1)"] \arrow[dr, phantom, "(2)"] & R \arrow[d, "n"] \\ G & D \arrow[l, "f"] \arrow[r, "g"'] & H \end{tikzcd}$$

1. *Delete those nodes and edges in $G$ that are reached by the match $m$, but keep the image of $K$, i.e. $D = (G \backslash m(L)) \cup m(l(K))$, such that $G = L +_K D$ in PO (1).*
2. *Add those nodes and edges that are newly created in $R$, i.e. $H = D \,\dot\cup\, (R \backslash r(K))$, such that $H = R +_K D$ in PO (2).*

*This construction is unique up to isomorphism.*



# Embedding of Graph Transformations

Problem: When can graph transformation $t : G_0 \overset{*}{\Rightarrow} G_n$ be extended to $t' : G'_0 \overset{*}{\Rightarrow} G'_n$ for $k_0 : G_0 \to G'_0$?

## Theorem

*Given transformation $t : G_0 \overset{*}{\Rightarrow} G_n$ and consistent morphism $k_0 : G_0 \to G'_0$ w.r.t. $t$, then there is an extension diagram (1) over $t$ and $k_0$.*

$$\begin{tikzcd} G_0 \arrow[r, Rightarrow, "t", "*"'] \arrow[d, "k_0"'] \arrow[dr, phantom, "(1)"] & G_n \arrow[d, "k_n"] \\ G'_0 \arrow[r, Rightarrow, "t'", "*"'] & G'_n \end{tikzcd}$$

Problem: What is consistency of $k_0$ with respect to $t$?

## Boundary, Context and Consistency

### Definition

boundary $B$ is smallest subgraph of $G_0$ containing $IP$ and $DP$ of $k_0 : G_0 \to G_0'$

context $C$ is smallest subgraph of $G_0'$ such that $G_0' = G_0 +_B C$

consistency: boundary $B$ is preserved by $G_0 \overset{t}{\Rightarrow} G_n$ leading to $b_n : B \to G_n$

### Fact

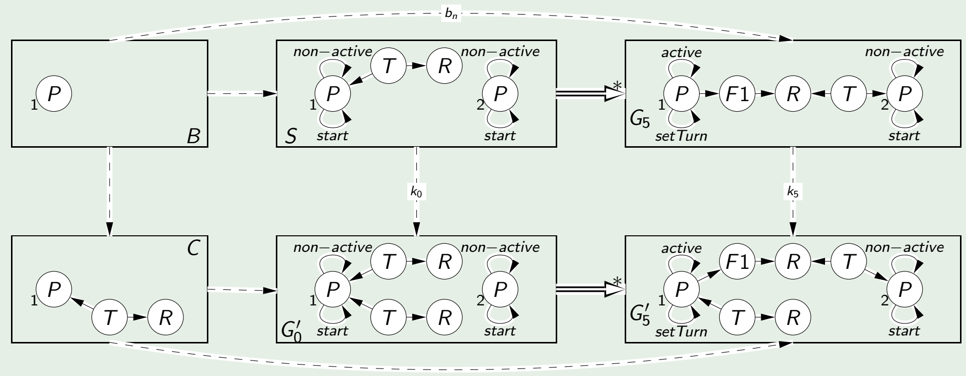
*In case of consistency we also have $G_n' = G_n +_B C$*

$$\begin{tikzcd}
B \arrow[r, "b_0"] \arrow[rr, bend left, "b_n"] \arrow[d] & G_0 \arrow[r, Rightarrow, "t", "*"'] \arrow[d, "k_0"] \arrow[dl, phantom, "(2)"] & G_n \arrow[d, "k_n"] \arrow[dl, phantom, "(1)"] \\
C \arrow[r] \arrow[rr, bend right] & G_0' \arrow[r, Rightarrow, "t'", "*"'] & G_n'
\end{tikzcd}$$



## Example: Embedding of Graph Transformation

### Example






# Part IV

## Independence of Graph Transformations and Critical Pairs





## Parallel and Sequential Independence

Problem: When are direct transformations commutable?

### Definition

$G \overset{p_1,m_1}{\Longrightarrow} H_1$ and $G \overset{p_2,m_2}{\Longrightarrow} H_2$ are parallel independent if
$$m_1(L_1) \cap m_2(L_2) \subseteq m_1(l_1(K_1)) \cap m_2(l_2(K_2)).$$
$G \overset{p_1,m_1}{\Longrightarrow} H_1 \overset{p_2,m_2}{\Longrightarrow} H_2$ are sequentially independent for comatch $n_1 : R_1 \to H_1$ if
$$n_1(R_1) \cap m_2(L_2) \subseteq n_1(r_1(K_1)) \cap m_2(l_2(K_2)).$$

### Fact

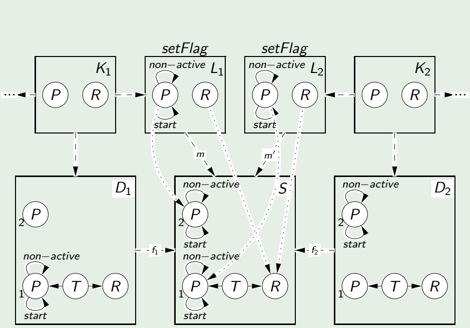
*$G \overset{p_1,m_1}{\Longrightarrow} H_1$ and $G \overset{p_2,m_2}{\Longrightarrow} H_2$ are parallel independent iff there exist morphisms $i : L_1 \to D_2$ and $j : L_2 \to D_1$ such that $f_2 \circ i = m_1$ and $f_1 \circ j = m_2$.*

$$\begin{tikzcd}
R_1 \arrow[d, "n_1"] & K_1 \arrow[l, "r_1"'] \arrow[r, "l_1"] \arrow[d, "k_1"] & L_1 \arrow[dr, "m_1"] \arrow[drrr, dashed, "i"] & L_2 \arrow[d, "m_2"] \arrow[dll, dashed, "j"'] & K_2 \arrow[l, "l_2"'] \arrow[r, "r_2"] \arrow[d, "k_2"] & R_2 \arrow[d, "n_2"] \\
H_1 & D_1 \arrow[l, "g_1"'] \arrow[rr, "f_1"] & & G & D_2 \arrow[l, "f_2"'] \arrow[r, "g_2"] & H_2
\end{tikzcd}$$

*Similar for sequential independence.*

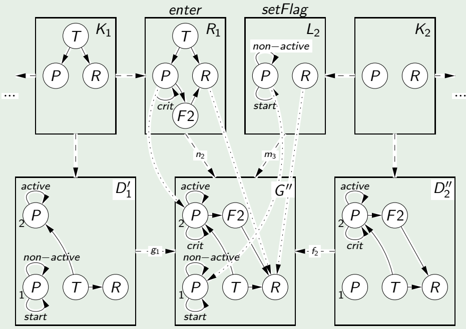
# Examples: Parallel and Sequential Independence

## Example




parallel independent:
$m(L_1) \cap m'(L_2)$
$= \{\text{node labelled } R\}$
$= m(l_1(K_1)) \cap m'(l_2(K_2))$

## Example



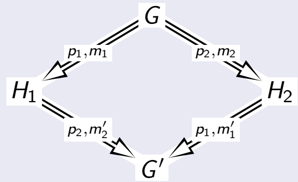

sequentially independent:
$n_2(R_1) \cap m_3(L_2)$
$= \{\text{node labelled } R\}$
$= n_2(r_1(K_1)) \cap m_3(l_2(K_2))$



# Local Church-Rosser Theorem

## Theorem

1. If $G \overset{p_1,m_1}{\Longrightarrow} H_1$ and $G \overset{p_2,m_2}{\Longrightarrow} H_2$ are parallel independent, there are $G'$ and $H_1 \overset{p_2,m_2'}{\Longrightarrow} G'$ and $H_2 \overset{p_1,m_1'}{\Longrightarrow} G'$ such that $G \overset{p_1,m_1}{\Longrightarrow} H_1 \overset{p_2,m_2'}{\Longrightarrow} G'$ and $G \overset{p_2,m_2}{\Longrightarrow} H_2 \overset{p_1,m_1'}{\Longrightarrow} G'$ are sequentially independent.

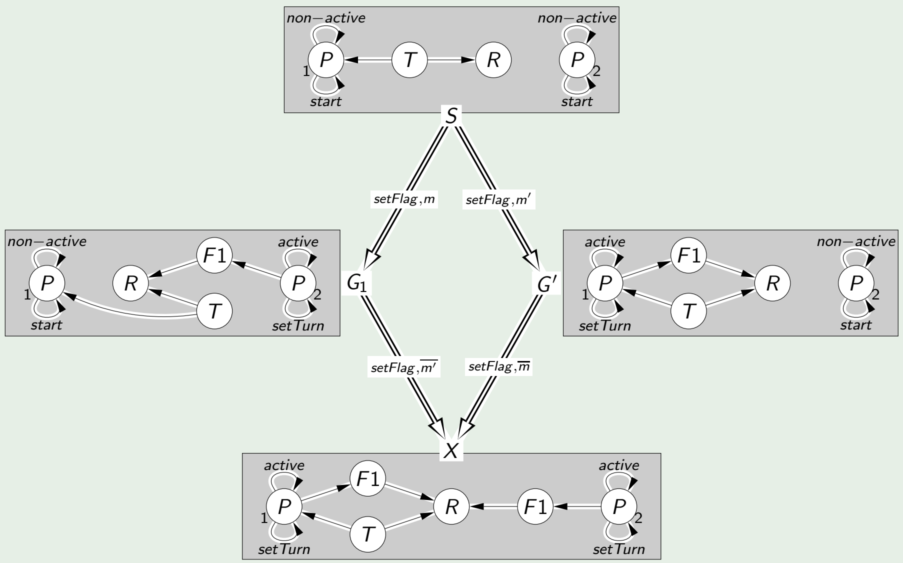



2. If $G \overset{p_1,m_1}{\Longrightarrow} H_1 \overset{p_2,m_2'}{\Longrightarrow} G'$ are sequentially independent, there are $H_2$ and $G \overset{p_2,m_2}{\Longrightarrow} H_2 \overset{p_1,m_1'}{\Longrightarrow} G'$ such that $G \overset{p_1,m_1}{\Longrightarrow} H_1$ and $G \overset{p_2,m_2}{\Longrightarrow} H_2$ are parallel independent.



# Example: Local Church-Rosser Theorem

## Example






# Overview of Parallelism and Concurrency

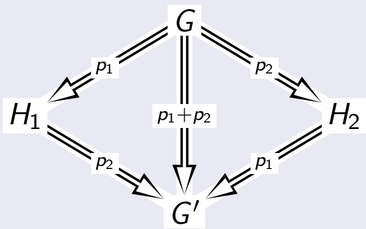
SKIP

Problem: How to compose productions $p_1$ and $p_2$?

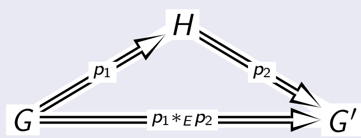
Solution:
1. Parallel composition $p_1 + p_2$ (independency)
2. E-concurrent production $p_1 *_E p_2$ (dependency)

## Theorem

*Parallelism*




*Concurrency*



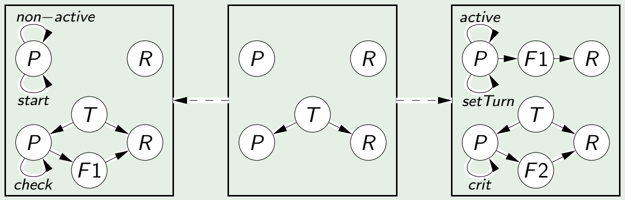
# Parallel Productions and Transformations

## Definition

- Given productions $p_1 = (L_1 \overset{l_1}{\leftarrow} K_1 \overset{r_1}{\rightarrow} R_1)$ and $p_2 = (L_2 \overset{l_2}{\leftarrow} K_2 \overset{r_2}{\rightarrow} R_2)$, the parallel production $p_1 + p_2$ is defined by the disjoint union
$$p_1 + p_2 = (L_1 \dot\cup L_2 \overset{l_1 \dot\cup l_2}{\longleftarrow} K_1 \dot\cup K_2 \overset{r_1 \dot\cup r_2}{\longrightarrow} R_1 \dot\cup R_2)$$
- The application of a parallel (typed) graph production is called a parallel direct (typed) graph transformation.

## Example

*enter + setFlag*:



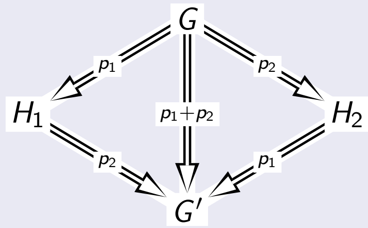



# Parallelism Theorem

## Theorem

*Given a transformation system GTS, we have:*

1. *Synthesis. If $G \Rightarrow H_1 \Rightarrow G'$ via productions $(p_1, p_2)$ are sequentially independent, then there is a parallel transformation $G \Rightarrow G'$ via the parallel production $p_1 + p_2$.*
2. *Analysis. For $G \Rightarrow G'$ via $p_1 + p_2$ there are sequentially independent $G \Rightarrow H_1 \Rightarrow G'$ via $(p_1, p_2)$ and $G \Rightarrow H_2 \Rightarrow G'$ via $(p_2, p_1)$.*
3. *Bijective correspondence. The synthesis and analysis constructions are inverse to each other up to isomorphism.*





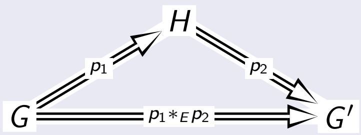

# Concurrency Theorem

Problem: How to compare productions and transformations in general?

## Theorem

*Given productions $p_1$ and $p_2$ with E-concurrent production $p_1 *_E p_2$, then:*

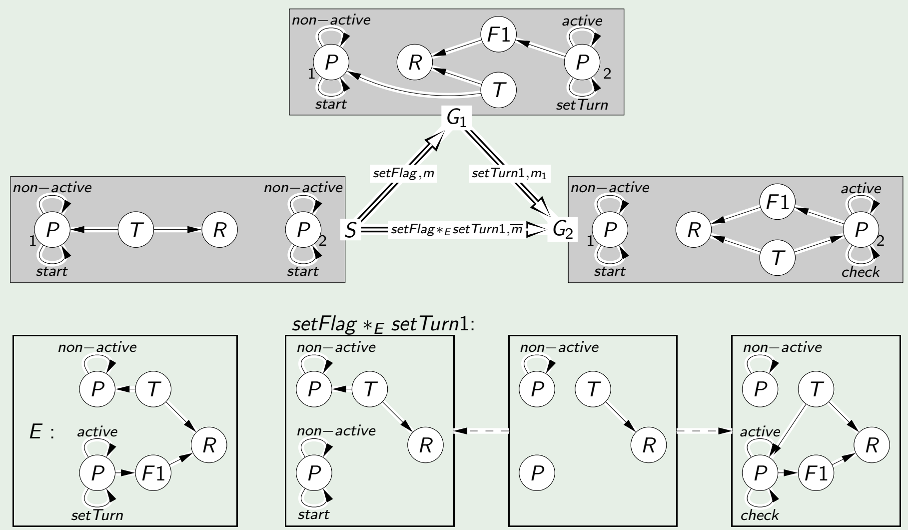
1. *Synthesis. Given an E-related transformation sequence $G \Rightarrow H \Rightarrow G'$ via $(p_1, p_2)$, then there is a synthesis construction leading to a direct transformation $G \Rightarrow G'$ via $p_1 *_E p_2$.*
2. *Analysis. Given a direct transformation $G \Rightarrow G'$ via $p_1 *_E p_2$, then there is an analysis construction leading to an E-related transformation sequence $G \Rightarrow H \Rightarrow G'$ via $(p_1, p_2)$.*
3. *Bijective correspondence. The synthesis and analysis constructions are inverse to each other up to isomorphism.*






# Example: Concurrency Theorem

## Example

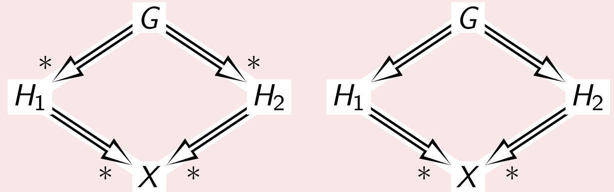


# Global Determinism and Confluence

Problem: How to achieve global determinism of a GTS in spite of local nondeterminism?

**Definition**

A GTS is (locally) confluent if, for all (direct) transformations $G \overset{*}{\Rightarrow} H_1$ and $G \overset{*}{\Rightarrow} H_2$, there is $X$ and transformations $H_1 \overset{*}{\Rightarrow} X$ and $H_2 \overset{*}{\Rightarrow} X$.




**Lemma**

*Every confluent GTS is globally deterministic.*



# Termination and Confluence

**Definition**

A GTS is terminating if there is no infinite sequence of graph transformations $(t_n : G \overset{*}{\Rightarrow} G_n)_{n\in\mathbb{N}}$ with $t_{n+1} = G \overset{t_n}{\Rightarrow} G_n \Rightarrow G_{n+1}$.

**Lemma**

*Every terminating and locally confluent GTS is also confluent.*

Problem: How to obtain local confluence in spite of parallel dependent graph transformations?

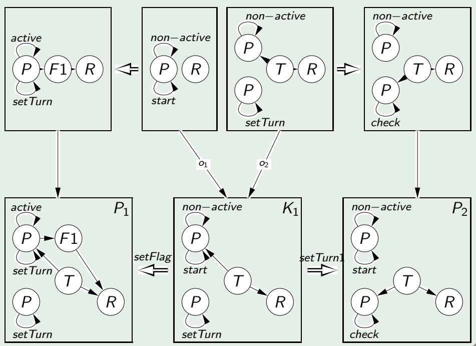


# Critical Pairs

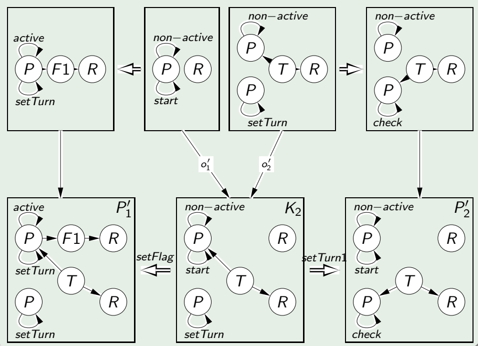
**Definition**

Pair $P_1 \overset{p_1,o_1}{\Longleftarrow} K \overset{p_2,o_2}{\Longrightarrow} P_2$ of direct transformations is called a critical pair if it is parallel dependent and minimal, i.e. the matches $o_1 : L_1 \rightarrow K$ and $o_2 : L_2 \rightarrow K$ are jointly surjective.

**Critical Pair 1**




**Critical Pair 2**



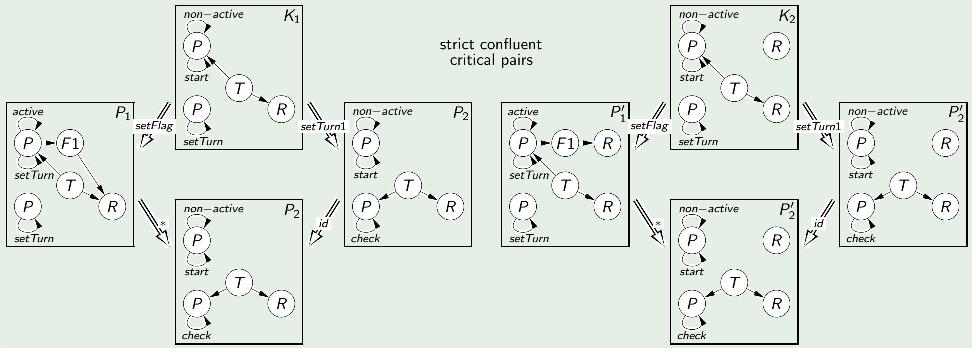



# Local Confluence Theorem - Critical Pair Lemma

**Theorem**

*A (typed) graph transformation system GTS is locally confluent if all its critical pairs are strictly confluent.*

**Example**



## Functional Behavior of Graph Transformation Systems

**Theorem**

*If a GTS is terminating and locally confluent, it has functional behavior:*

1. *For each $G$, there is a terminating transformation $G \overset{*}{\Rightarrow} H$ in GTS, and $H$ is unique up to isomorphism.*
2. *Each pair $G \overset{*}{\Rightarrow} H_1$ and $G \overset{*}{\Rightarrow} H_2$ can be extended to terminating transformations $G \overset{*}{\Rightarrow} H_1 \overset{*}{\Rightarrow} H$ and $G \overset{*}{\Rightarrow} H_2 \overset{*}{\Rightarrow} H$ with the same $H$.*

**Theorem**

*Every layered typed graph grammar $GG = (TG, P, G_0)$ with injective matches terminates if it satisfies suitable layer conditions for deleting and nondeleting productions.*



# Part V

## Adhesive HLR Categories and Typed AGT Systems





## Motivation for a Categorical Framework

- Problem:
  - Up to now we have graph transformation theory for **Graphs**, $\mathbf{Graphs_{TG}}$ only
  - But the theory is required also for
    - hypergraph transformation systems,
    - Petri net transformation systems,
    - transformation systems of algebraic specifications and
    - typed attributed graph transformation systems.
- Solution:
  - Replace **Graphs** by suitable category.
  - Suitable is Adhesive HLR category based on van Kampen property.



## Overview of Adhesive HLR Categories

SKIP

Axioms: Adhesive HLR categories $(\mathbf{C}, \mathcal{M})$ satisfy:

1. $\mathcal{M} \subseteq Monos$ closed under composition and decomposition
2. pushouts (POs) and pullbacks (PBs) along $\mathcal{M}$-morphisms
3. compatibility of POs and PBs by van Kampen squares

Examples: **Sets**, **Graphs**, $\mathbf{Graphs_{TG}}$, **PTNets**, **AHLNets**, $(\mathbf{AGraphs_{ATG}}, \mathcal{M})$

**Theorem**

1. *construction of adhesive HLR categories by product, slice, coslice, comma and functor categories*
2. *properties of adhesive HLR categories are sufficient for all constructions and results of Parts II-IV*

## Van Kampen Squares

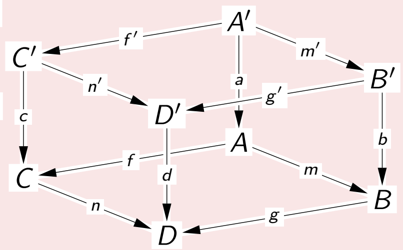
Idea: compatibility of pushouts and pullbacks

### Definition

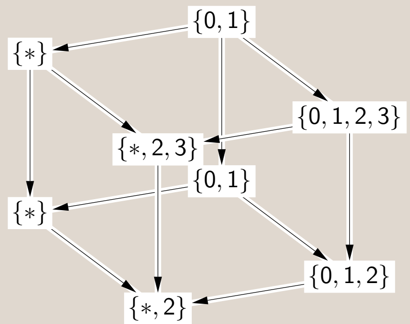
Pushout (1) is a van Kampen square if, for any commutative cube with (1) in the bottom and where the back faces are pullbacks, we have:

The top face is pushout iff the front faces are pullbacks.

$$\begin{tikzcd} A \arrow[r, "m"] \arrow[d, "f"'] \arrow[dr, phantom, "(1)"] & B \arrow[d, "g"] \\ C \arrow[r, "n"] & D \end{tikzcd}$$




### Example






## Adhesive Categories

### Definition

category **C** is an adhesive category if:

1. **C** has pushouts along monomorphisms (i.e. pushouts where at least one of the given morphisms is a monomorphism)
2. **C** has pullbacks
3. Pushouts along monomorphisms are VK squares

Examples: **Sets**, **Graphs**, $\mathbf{Graphs_{TG}}$, **Hypergraphs**, **ElemNets**

Counterexamples: **PTNets**, $\mathbf{AGraphs_{ATG}}$

Idea for generalization: Replace monomorphisms by subclass $\mathcal{M}$ and pullbacks by those along monomorphisms



## (Weak) Adhesive HLR Categories

### Definition

category **C** with a morphism class $\mathcal{M}$ is called a (weak) adhesive HLR category if:

1. $\mathcal{M}$ is a class of monomorphisms closed under isomorphisms, composition ($f : A \to B \in \mathcal{M}, g : B \to C \in \mathcal{M} \Rightarrow g \circ f \in \mathcal{M}$), and decomposition ($g \circ f \in \mathcal{M}, g \in \mathcal{M} \Rightarrow f \in \mathcal{M}$)
2. **C** has pushouts and pullbacks along $\mathcal{M}$-morphisms, and $\mathcal{M}$-morphisms are closed under pushouts and pullbacks
3. Pushouts in **C** along $\mathcal{M}$-morphisms are (weak) VK squares

Examples: $(\mathbf{PTNets}, \mathcal{M}_{mono})$, $(\mathbf{Spec}, \mathcal{M}_{strict})$, $(\mathbf{AHLNets}, \mathcal{M}_{mono})$, $(\mathbf{AGraphs_{ATG}}, \mathcal{M})$ (see below)



## Construction and Properties of AHLR categories

### Theorem

*If $(\mathbf{C}, \mathcal{M}_1)$ and $(\mathbf{D}, \mathcal{M}_2)$ are (weak) adhesive HLR categories, then the following categories are (weak) adhesive HLR categories:*

1. *the product category $(\mathbf{C} \times \mathbf{D}, \mathcal{M}_1 \times \mathcal{M}_2)$, the slice category $(\mathbf{C}\backslash X, \mathcal{M}_1 \cap \mathbf{C}\backslash X)$ and the coslice category $(X\backslash\mathbf{C}, \mathcal{M}_1 \cap X\backslash\mathbf{C})$,*
2. *the functor category $([\mathbf{X}, \mathbf{C}], \mathcal{M} - \textit{functor transformations})$,*
3. *the comma category $(ComCat(F, G; \mathcal{I}), (\mathcal{M}_1 \times \mathcal{M}_2) \cap Mor_{ComCat})$, where $F : \mathbf{C} \to \mathbf{X}$ preserves pushouts along $\mathcal{M}_1$-morphisms and $G : \mathbf{D} \to \mathbf{X}$ preserves pullbacks (along $\mathcal{M}_2$-morphisms).*

*In (weak) adhesive categories, the following properties hold:*

1. *Pushouts along $\mathcal{M}$-morphisms are pullbacks*
2. *$\mathcal{M}$ pushout–pullback decomposition lemma*
3. *Cube pushout–pullback lemma*
4. *Uniqueness of pushout complements*

## Adhesive HLR Systems

### Definition

- adhesive HLR system $AHS = (\mathbf{C}, \mathcal{M}, P)$ consists of (weak) adhesive HLR category $(\mathbf{C}, \mathcal{M})$ and set of productions $P$ with morphisms in $\mathcal{M}$
- adhesive HLR grammar $AHG = (AHS, S)$ is an adhesive HLR system together with a distinguished start object $S$
- language $L$ of an adhesive HLR grammar is defined by

$$L = \{G \mid \exists \text{ transformation } S \overset{*}{\Rightarrow} G\}.$$

The categorical framework allows to prove the results from Parts III and IVj for all the instantiations (like **Graphs**, **HyperGraphs**, **PTNets** etc.).



## Motivation for Typed AGT Systems

Typed attributed graph transformation (AGT) is

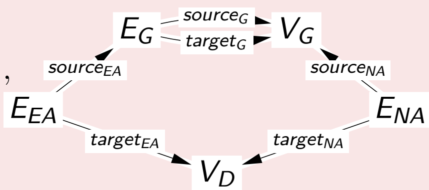
- integration of typed graph transformation with
- data types as attributes for
  - modeling of visual languages (VL) with
    - alphabet of a VL given by attributed type graph,
    - syntax given by generating grammar,
    - operational semantics given by simulating grammar and animation.
  - model transformations between VLs by
    - typed AGT systems.



## Typed Attributed Graphs

### Definition

E-graph $G$ is defined by $G = (V_G, V_D, E_G, E_{NA}, E_{EA}, (source_j, target_j)_{j \in \{G,NA,EA\}})$.




Let $DSIG = (S_D, OP_D)$ be a data signature with $S'_D \subseteq S_D$. Attributed graph $AG = (G, D)$ is E-graph $G$ and $DSIG$-algebra $D$ with $\dot{\bigcup}_{s \in S'_D} D_s = V_D$.

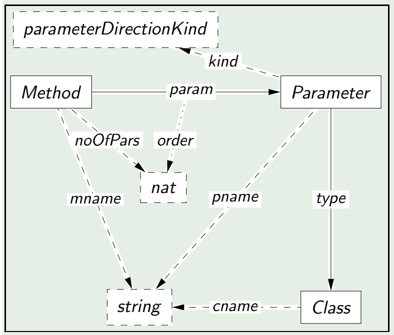
$\mathbf{AGraphs_{ATG}}$ = category of typed attributed graphs $(G, D)$ and typed attributed graph morphisms $(f_G, f_D)$ consistent w.r.t. $S'_D$ over an attributed type graph $ATG$



## Example: Typed Attributed Graphs

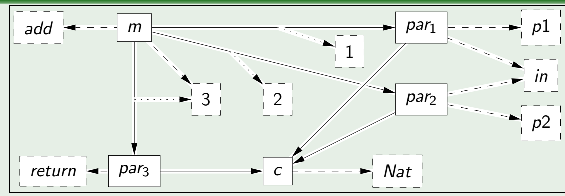
### Example

Type graph $ATG$:

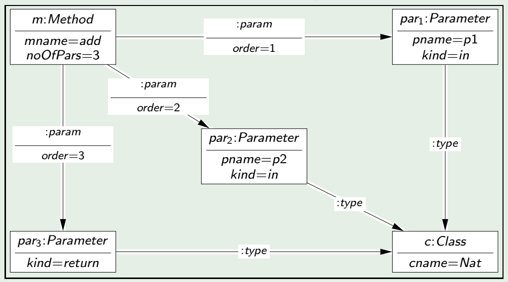



$DSIG = CHAR + STRING + NAT +$

sorts : $parameterDirectionKind$
opns : $in, out, inout, return : \to parameterDirectionKind$
$S'_D = \{string, nat, parameterDirectionKind\}$




Notations for typed attributed graphs:



## Typed Attributed Graph Transformation

### Definition

- typed attributed graph production $p = (L \overset{l}{\leftarrow} K \overset{r}{\rightarrow} R)$ consists of typed attributed graphs $L$, $K$, $R$ with common $DSIG$-termalgebra $T_{DSIG}(X)$ with variables $X$, and $l, r \in \mathcal{M}$, with $\mathcal{M} = \{f = (f_G, f_D) \mid f_g \text{ injective, } f_D \text{ isomorphism of } DSIG\text{-algebras}\}$
- direct typed attributed graph transformation $G \overset{p,m}{\Longrightarrow} H$ from $G$ to $H$ via the match $m$ is given by the following double pushout diagram in $\mathbf{AGraphs_{ATG}}$

$$\begin{tikzcd} L \arrow[d, "m"'] \arrow[dr, phantom, "(1)"] & K \arrow[l, "l"'] \arrow[r, "r"] \arrow[d, "k"] \arrow[dr, phantom, "(2)"] & R \arrow[d, "n"] \\ G & D \arrow[l, "f"] \arrow[r, "g"'] & H \end{tikzcd}$$



## Theory for Typed AGT Systems

Idea: Instantiation of theory for adhesive HLR systems

### Theorem

*$(\mathbf{AGraphs_{ATG}}, \mathcal{M})$ is an adhesive HLR category.*

### Corollary

*In $\mathbf{AGraphs_{ATG}}$, the following results hold:*

- *Construction of typed attributed graph transformations,*
- *Embedding Theorem,*
- *Local Church-Rosser Theorem,*
- *Parallelism Theorem,*
- *Concurrency Theorem,*
- *Local Confluence Theorem.*



# Part VI

## Advanced Features and Conclusion





## Overview of Constraints and Application Conditions

SKIP

Idea of graph constraints:
Restriction of graphs / graph languages by existence or nonexistence of suitable graph patterns, called constraints $c$

Idea of application conditions:
Restriction of direct graphs transformations $G \overset{p,m}{\Rightarrow} H$ by suitable conditions for the match $m$ (e.g. gluing condition)

### Theorem

*Construction of application condition $acc$ for match $m$ from graph constraint $c$ such that $H \models c$ if $m \models acc$.*

# Graph Constraints

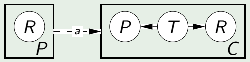
Problem: How to define graph constraints?

## Definition

- atomic (typed) graph constraint is of form $PC(a)$, where $a : P \to C$ is (typed) graph morphism
- (typed) graph constraint is Boolean formula over atomic (typed) graph constraints
- $G \models PC(a)$, if, for every injective morphism $p : P \to G$, there exists injective morphism $q : C \to G$ such that $q \circ a = p$.
  This can be extended to boolean formulas.

$$\begin{tikzcd} P \arrow[rr, "a"] \arrow[dr, "p"'] & \arrow[d, phantom, "="] & C \arrow[dl, "q"] \\ & G & \end{tikzcd}$$

## Example




$G$ satisfies this constraint if, for each resource node $R$, there is a turn variable that connects it to a process.



# Application Conditions

Problem: How to define application conditions?

## Definition

- atomic application condition over $L$ is given by $P(x, \vee_{i \in I} x_i)$
- $m \models P(x, \vee_{i \in I} x_i)$, if, for all injective $p : X \to G$ with $p \circ x = m$, there exist $i \in I$ and injective $q_i : C_i \to G$ with $q_i \circ x_i = p$.
  This can be extended to boolean formulas.
- negative application condition $NAC(x)$ is a morphism $x : L \to X$. A morphism $m : L \to G$ satisfies $NAC(x)$ if there does not exist an injective $p : X \to G$ with $p \circ x = m$.

$$\begin{tikzcd} L \arrow[r, "x"] \arrow[dr, "m"'] & X \arrow[r, "x_i"] \arrow[d, "p"] & C_i \arrow[dl, "q_i"] \\ & G & \end{tikzcd}$$

$$\begin{tikzcd} L \arrow[rr, "x"] \arrow[dr, "m"'] & \arrow[d, phantom, "="] & X \arrow[dl, "p"] \\ & G & \end{tikzcd}$$



# Productions with Application Conditions

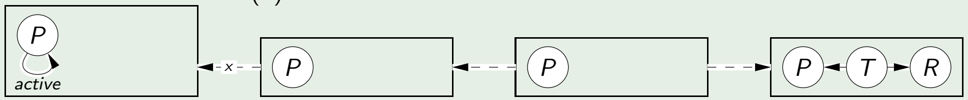
## Definition

application condition $A(p) = (A_L, A_R)$ for production $p$ consists of left application condition $A_L$ over $L$ and right application condition $A_R$ over $R$

$G \overset{p,m}{\Longrightarrow} H$ with comatch $n : R \to H$ satisfies $A(p)$ if $m \models A_L$ and $n \models A_R$.

## Example

*addResource* with $NAC(x)$:






# Construction of Application Conditions

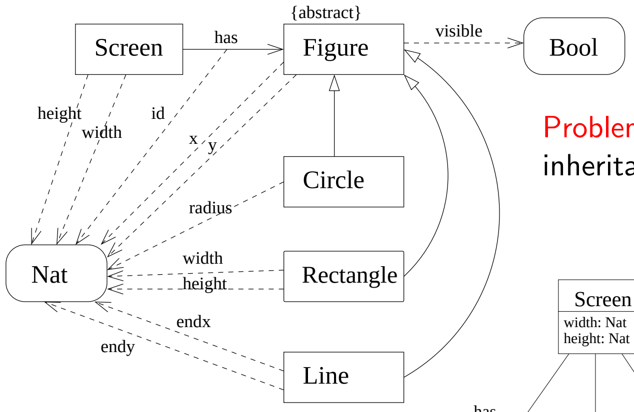
## Theorem

- *For each graph constraint c for H, there is an equivalent right application condition racc for the comatch n.*
- *For each right application condition racc for comatch n, there is an equivalent left application condition lacc for match m.*

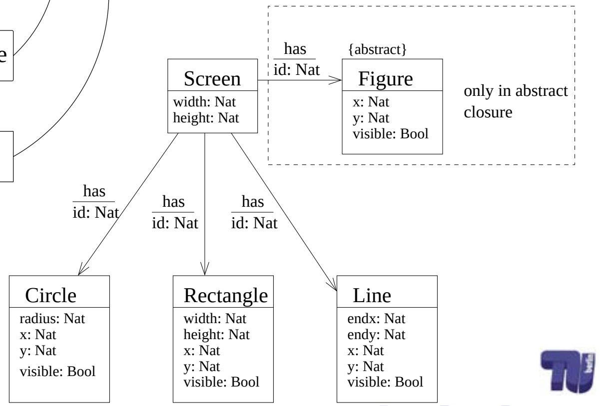
$$\begin{tikzcd} L \arrow[d, "m"'] \arrow[dr, phantom, "(1)"] & K \arrow[l, "l"'] \arrow[r, "r"] \arrow[d, "k"] \arrow[dr, phantom, "(2)"] & R \arrow[d, "n"] \\ G & D \arrow[l, "f"] \arrow[r, "g"'] & H \end{tikzcd}$$

- $H \models c$, *if* $m \models lacc$.

# Typed AGT Systems with Inheritance




Problem: How to handle inheritance?

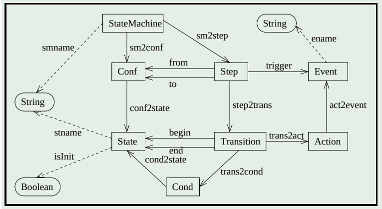




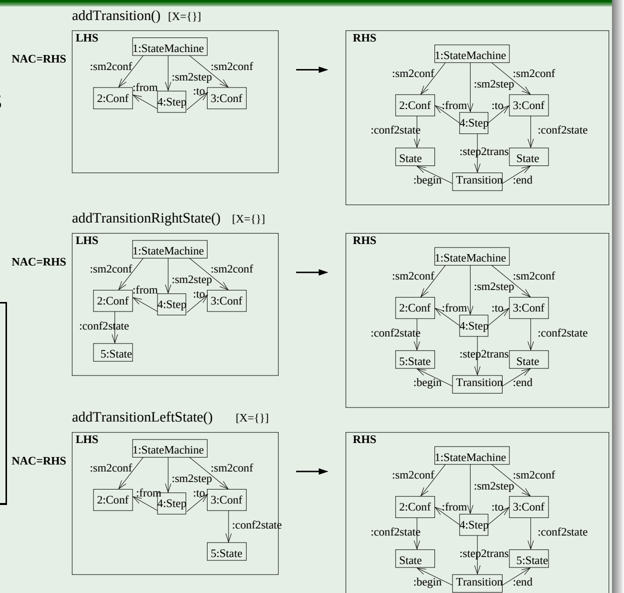

# Modeling of Visual Languages

## Example

Generation of Statecharts by typed attributed graph grammar

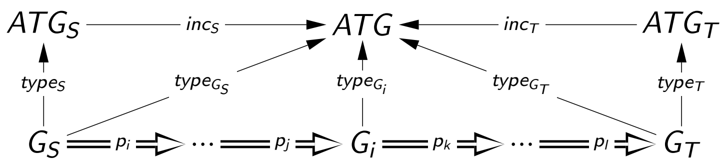


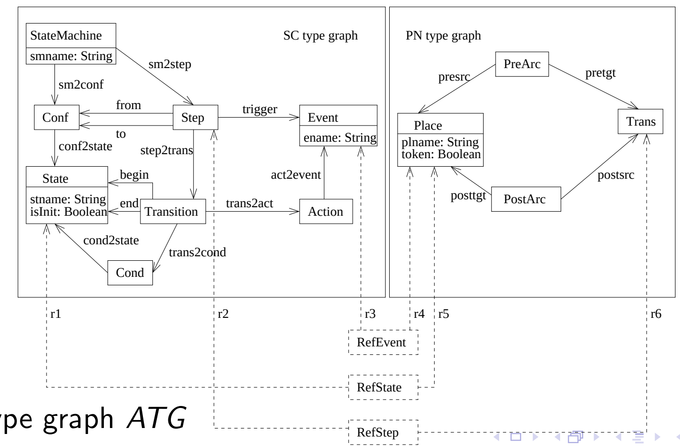

SC type graph






# Model Transformation from Statecharts to Petri Nets



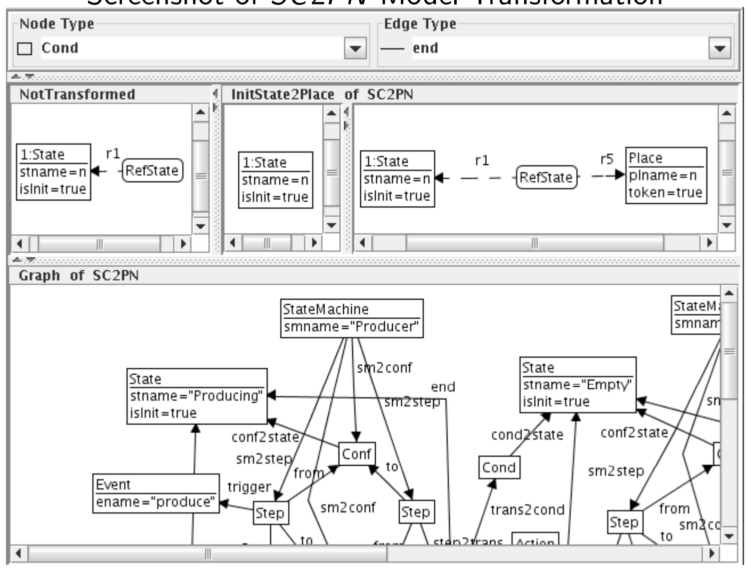




SC2PN type graph ATG



# Implementation of Typed AGT Systems in AGG

Screenshot of *SC2PN* Model Transformation



# Conclusion

**Fundamental aspects of algebraic graph transformation:**

1. Algebraic concept of graph transformation based on
   - gluing construction and double pushouts.
2. Basic results concerning
   - independence, parallelism, critical pairs.
3. Categorical framework of AHLR systems for
   - unified constructions and proofs.
4. Typed attributed graph transformation for
   - modeling of software systems,
   - model transformations of visual languages.