Graph Grammars & Petri Net Transformations

Hartmut Ehrig
Technical University Berlin

Overview of the Talk

- Part 1: Graph Grammars and Graph Transformation
- Part 2: High-Level Replacement and Petri Net Transformations
- Conclusion
Part 1:
Graph Grammars & Graph Transformation

- General Overview
- Introduction to Double Pushout (DPO) Approach
- Concepts for Parallelism
- Graph Grammars and Petri Nets
What is Graph Transformation

Rule-Based Modification of Graphs

- from Chomsky grammars to graph grammars
- from term rewriting to graph transformation
- from textual description to visual specification
Aims and Paradigms

- computing based on term rewriting
  (functional and logic programming)

- replace trees by graphs
  (sharing of common substructures)

- computing by graph transformation
  as fundamental concept for
  - programming
  - specification
  - concurrency
  - distribution
Overview of Graph Grammar Approaches

- Node Replacement (Rozenberg, Engelfried)
- Hyperedge Replacement (Habel, Kreowski)
- Algebraic Approaches
  - Double Pushout (Ehrig, Schneider, Corradini et al)
  - Single Pushout (Raoult, Löwe et al)
  - Pullback (Bouderon)
  - Double Pullback (Heckel et al)
- Logical approach (Courcelle, Bouderon)
- 2-structures (Rozenberg)
- Programmed Graph Replacement (Schuerr)

COMPUGRAPH — GETGRATS — APPLIGRAPH — SEGRAVIS
History of Algebraic Approaches

1972/73 * DPO
   Berlin–Erlangen–SWAT’73

73–80 Church-Rosser, Parallelism
   Berlin–Yorktown Heights

80–90 * SPO, * Hyperedge
   Concurrency, Logical
   Germany–Europe–USA

90–98 * HLR, * PB-DPB
   Semantics, Distribution, Modularization
   ESPRIT & TMR
Basic Notions of the DPO Approach

- Graphs
- Graph Morphisms
- Graph Grammars
- Direct Derivations via Double Pushout
- Derivations
Graphs

- Directed, labeled graphs over fixed sets of colors $\Omega_E, \Omega_V$

$$G = \Omega_E \xleftarrow{le} E \xrightarrow{s} V \xrightarrow{lv} \Omega_V$$

- $\Omega_V = \{●, ●, ●, ●, ●\}$
- $\Omega_E = \{\ast\}$
- Identities of nodes and edges are ignored (⇒ abstract graphs)

Pacman Graph $PG$
Graph Morphisms

Category **Graph** has graphs as objects and graph morphisms as arrows.
Graph Grammars

- Production: \[ p = L \xleftarrow{l} K \xrightarrow{r} R \]

  - moveP =

  - eat =

  - kill =

- Graph Grammar: \{ PG, \{moveP, moveA, kill, eat\} \}
Given a production $p = (L \xleftarrow{l} K \xrightarrow{r} R)$ and a match $m : L \rightarrow G$, there is a direct derivation

$$G \xrightarrow{p,m} H$$

if the following Double-Pushout diagram can be constructed:

$$\begin{array}{ccc}
L & \xleftarrow{l} & K & \xrightarrow{r} & R \\
\downarrow m & & \downarrow & & \downarrow \\
G & \xleftarrow{} & D & \rightarrow & H
\end{array}$$

- Derivation: $G_0 \xrightarrow{p_1,m_1} G_1 \xrightarrow{p_2,m_2} \ldots \xrightarrow{p_n,m_n} G_n$
A Sample Derivation

\[ \text{moveP} = \]

\[ G = G_0 \xrightarrow{\text{moveP}} G_1 \xrightarrow{\text{moveP}} G_2 \xrightarrow{\text{eat}} G_3 \xrightarrow{\text{kill}} \]

Graph Transformations & Petri Nets

H. Ehrig - TU Berlin
• “Union” of structures along common sub-structure.

• In the DPO diagram it models “addition” of (part of) $R$.

Categorical characterization

$$
\begin{align*}
A & \xrightarrow{f} B \\
C & \xrightarrow{g} D \\
\end{align*}
\quad \xrightarrow{D'}
$$

• In $\textbf{Set}$, $D = B \uplus C/\equiv$, where $\equiv$ is generated by $f(a) \sim g(a)$ for all $a \in A$. 
Pushout Complement

- In the DPO diagram it models “deletion” of (part of) $L$.
- Does not always exist: gluing conditions.
- Not unique in general.
Concepts for Parallelism

- Parallel and Sequential Independence
- Parallel Productions and Derivations
- Local Church-Rosser Theorem
- Parallelism Theorem
- Shift Equivalence and Canonical Derivations
Parallel and Sequential Independence

- Two direct derivations are *parallel independent* if their matches only overlap on preserved items.

- Two consecutive direct derivations are *sequential independent* if they overlap in the intermediate graphs only on items preserved by both.
Parallel Productions and Derivations

- **Parallel Production**: an example

```
  moveP + moveG
```

- A **Parallel Derivation** is a derivation which uses parallel productions
Local Church-Rosser Theorem

\[ G \xrightarrow{p_1,m_1} H_1 \text{ and } G \xrightarrow{p_2,m_2} H_2 \text{ are parallel independent} \]

iff

\[ G \xrightarrow{p_1,m_1} H_1 \xrightarrow{p_2,m_2'} X \text{ is sequential independent} \]

iff

\[ G \xrightarrow{p_2,m_2''} H_2 \xrightarrow{p_1,m_1'} X \text{ is sequential independent} \]
Parallelism Theorem

There is a parallel direct derivation \( G^{p_1+p_2,m} \xrightarrow{m} X \)
iff
\( G^{p_1,m_1} \xrightarrow{m_1} H_1 \) and \( G^{p_2,m_2} \xrightarrow{m_2} H_2 \) are parallel independent
iff
\( G^{p_1,m_1} \xrightarrow{m_1} H_1 \xrightarrow{m_2} X \) is sequential independent
iff
\( G^{p_2,m_2'} \xrightarrow{m_2'} H_2 \xrightarrow{p_1,m_1'} X \) is sequential independent

\[ \begin{array}{c}
\text{parallelism diamond} \\
G \xrightarrow{p_1+p_2} X \\
\end{array} \]
Shift Equivalence and Canonical Derivations

- **Shift equivalence** on derivations is the closure of the relation generated by the *parallelism diamond* under parallel and sequential composition.

- **Shift relation**:

\[
G_1 \xrightarrow{p_1} G_2 \xrightarrow{p_2 + p_3} G_3 \quad \sqsubseteq_{\text{shift}} \quad G_1 \xrightarrow{p_1 + p_2} G'_2 \xrightarrow{p_3} G_3
\]

if \( p_1 \) and \( p_2 \) are sequentially independent.

- The shift relation is well-founded. *Canonical derivations* are minimal with respect to shift relation.
Graph Grammars and Petri Nets

- Graph grammars and Petri nets are well known as specification formalisms for concurrent and distributed systems.

- Token game of Petri nets can be modelled by graph transformation (transfer of semantical concepts).

- Graph Grammars can be seen to generalize classical nets by allowing *more structured states* and *contextual* rewriting.

- Net structure of Petri nets can be modified by graph transformation (see Net Transformation Systems).
**Transition Firing as Double Pushout**

A sample transition firing

- **Markings** are discrete graphs labeled on *places*
- **Transitions** are productions with empty interface
## Correspondence of Notions

<table>
<thead>
<tr>
<th>Petri Nets</th>
<th>Graph Grammars</th>
</tr>
</thead>
<tbody>
<tr>
<td>tokens</td>
<td>nodes</td>
</tr>
<tr>
<td>places</td>
<td>node labels</td>
</tr>
<tr>
<td>marking</td>
<td>discrete, labeled graph</td>
</tr>
<tr>
<td>transition enabled at a marking</td>
<td>production applicable to a graph</td>
</tr>
<tr>
<td>firing</td>
<td>direct derivation</td>
</tr>
<tr>
<td>firing sequence</td>
<td>derivation</td>
</tr>
<tr>
<td>concurrent transitions</td>
<td>parallel independent productions</td>
</tr>
<tr>
<td>conflict</td>
<td>parallel dependence</td>
</tr>
<tr>
<td>step</td>
<td>parallel direct derivation</td>
</tr>
<tr>
<td>step sequence</td>
<td>parallel derivation</td>
</tr>
</tbody>
</table>
- Graph grammars can be considered as a non-trivial extension of Petri nets (Diss. Baldan)

\[
\text{Petri nets} \quad \longrightarrow \quad \text{Contextual nets} \quad \longrightarrow \quad \text{Inhibitor nets} \quad \longrightarrow \quad \text{Graph grammars}
\]

- Semantics of safe P/T nets as chain of adjunctions ([Win87])

- Semantics of graph transformations depicted as a chain of functors (Diss. Baldan)

\[
\text{Systems} \quad \longleftarrow \quad \text{Acyclic systems} \quad \longrightarrow \quad \text{Event structures} \quad \longleftarrow \quad \text{Domains} \quad \longleftarrow \quad \text{Prime event structures}
\]
Part 2: High-Level Replacement & Petri Net Transformations

- General Idea of High Level Replacement (HLR)-Systems
- Basic HLR-Results
- Petri Net Transformations
- Property Preserving Net Transformations
Motivation for HLR Systems

Generalization of concept of transformations to other high-level structures.

Abstract Transformation Example

- Rule: the upper line
- Transformation: the bottom line
High-Level Replacement Systems

Generalized results from graph grammars. Based on double pushout approach. [Ehrig, Habel, Kreowski, Parisi-Presicce 91]

HLR system: category $\text{CAT}$ with a class of injective morphisms.

Transformation defined via double pushout diagram

\[
\begin{array}{ccc}
L & \leftarrow & K \\
& l & \uparrow m & \rightarrow & r \\
& (1) & k & (2) & n \\
G & \leftarrow & C & \rightarrow & H \\
& g & \downarrow & h
\end{array}
\]

- applicability of rules (gluing conditions)
- HLR-System is given for a category $\text{CAT}$ with class of morphisms $\mathcal{M}$ by an object in $\text{CAT}$ and a set of rules.
Rule-based transformations of structures.

Petri net example
**Horizontal Structuring**

*Union* is a combination of two structures with shared substructure.

*Fusion* is a combination of two copies of a substructure within one structure.

Union:

Fusion:
Application: Stepwise Development of Systems

Stepwise development using Model Transformations

\[ N = N_1 \Longrightarrow N_2 \Longrightarrow \ldots \Longrightarrow N_k = N' \]

Instantiations: Graph Transformations, Petri Nets, Alg. Specifications
Application: Decomposition of Complex Systems

Describing a complex system as union and fusion of its component.

Union and fusion are compatible with each other.
Generalized results from graph transformations:

- Independence Conditions
- Local Church-Rosser Theorem(s)
- Parallelism Theorem
- Compatibility of Union with Transformation
- Compatibility of Fusion with Transformation
Compatibility of independent sequential and parallel transformations of high-level structures.
Compatibility of Union and Transformation

Union of transformed structures (nets) is equivalent (up to isomorphisms) to the result of transformation of unions of structures (nets).
Compatibility of Fusion and Transformation

Fusion of transformed structures (nets) is equivalent (up to isomorphisms) to the result of transformation of fused structures (nets).

\[ F \]
\[ N_1 \xrightarrow{p} N'_1 \]
\[ N_2 \xrightarrow{p} N'_2 \]
Net transformation systems are instantiations of HLR-systems to different classes (categories) of Petri nets as P/T nets, coloured nets, AHL nets.
**Place/Transition Petri Nets**

**P/T net**

\[ N = (P, T, \text{pre}, \text{post}) \]

is defined by

- \( P \) .. set of places
- \( T \) .. set of transitions
- \( \text{pre} : T \rightarrow P^\oplus \) .. predomain function
- \( \text{post} : T \rightarrow P^\oplus \) .. postdomain function

\( P^\oplus \) ... free commutative monoid

**P/T net morphism**

\[ f = (f_P : P_1 \rightarrow P_2, f_T : T_1 \rightarrow T_2) \]

with

\[ T_1 \xrightarrow{\text{pre}_1} P_1^\oplus \]
\[ f_T \downarrow \]
\[ T_2 \quad \xrightarrow{\text{post}_2} \]
\[ P_2^\oplus \]
\[ \text{post}_1 \]
\[ \text{pre}_2 \]
\[ f_P^\oplus \]

Graph Transformations & Petri Nets

H. Ehrig - TU Berlin
**Example: Dining Philosophers**

- **P/T net**
  - States: `think1`, `eat1`, `put1`, `take1`, `side3`, `side1`, `put3`, `eat3`, `think3`, `take3`, `put2`, `eat2`, `think2`, `side2`
  - Transitions: `think1`, `eat1`, `put1`, `take1`, `put3`, `eat3`, `think3`, `take3`, `put2`, `eat2`, `think2`, `side1`, `side2`, `side3`

- **AHL net**
  - States: `think`, `eat`, `side`, `put```

**SIG=**
- sorts: philo, stick
- opns: `p1, p2, p3: philo`
- s1, s2, s3: stick
- ls, rs: philo → stick

**E:**
- `ls(pi) = s_i`
- `rs(p1) = s3, rs(p2) = s1, rs(p3) = s3`
**Algebraic High-Level Nets**

**Algebraic High-Level Net** (AHL net) is given by

\[
N = (SPEC, P, T, pre, post, cond, A)
\]

- \(SPEC\) : \((SIG, E)\) algebraic specification
- \(P\) : set of places
- \(T\) : set of transitions
- \(pre, post\) : \(T \rightarrow (TOP(X) \times P)^\oplus\) arcs
- \(cond\) : \(T \rightarrow \mathcal{P}_{fin}(EQNS(SIG))\) transition guards
- \(A\) : \(SPEC\) algebra

Notation:

\[
\mathcal{P}_{fin}(EQNS(SIG)) \xrightarrow{\text{cond}} T \xrightarrow{T_{OP}(X) \times P} (TOP(X) \times P)^\oplus
\]
Morphisms of AHL nets

A Morphism \( f : N_1 \to N_2 \) is given by \( f = (f_{SPEC}, f_P, f_T, f_A) \)

\[
\begin{align*}
\mathcal{P}_{fin}(EQNS(SIG_1)) & \xleftarrow{\text{cond}_1} T_1 \xrightarrow{\text{pre}_1} (TOP_1(X_1) \times P_1)^\oplus \\
\mathcal{P}_{fin}(f^*_{SPEC}) & \xrightarrow{\text{fT}} (f_{SPEC}; f_A) \\
\mathcal{P}_{fin}(EQNS(SIG_2)) & \xleftarrow{\text{cond}_2} T_2 \xrightarrow{\text{pre}_2} (TOP_2(X_2) \times P_2)^\oplus \\
\end{align*}
\]

\[ (SPEC_1, A_1) \xrightarrow{(f_{SPEC}, f_A)} GALG \xrightarrow{iso} \text{category of generalized homomorphisms} \]

\[ \text{with } f_A : A_1 \xrightarrow{\sim} V_{f_{SPEC}}(A_2) \]
**Property Preserving Net Transformations**

*Q*-transformation [Padberg 96]: A theory of more general refinement/abstraction morphisms introduced for HLR-systems.

**Q-Transformation** $N_1 \Rightarrow N_2$ via $(p, m)$

\[
p : \quad L \leftarrow K \rightarrow R
\]

Double Pushout with

- $Q$-morphisms
- preserving/respecting properties
Q-Morphism and Q-Transformation

Q-morphism

Q-transformation

Graph Transformations & Petri Nets

H. Ehrig - TU Berlin
Property Preserving Net Transformations

\[ Q - \text{Net Transformation} \ N_1 \Rightarrow N_2 \text{ via } (p, m) \text{ preserves } \]

- **safety properties** [Padberg, Gajewsky, Hoffmann, Ermel 98-00]
- **liveness** [Padberg, Gajewsky, Urbasuk, TUB-Brno 01]
- **union & fusion** [Padberg 96, 99]

**Safety properties** – propositional logic formulas upon the actual marking of Petri nets.

\( Q: \) transition gluing/place preserving/collapsing morphisms

**Liveness** – no deadlock or livelock can occur, i.e.

\( Q: \) collapsing morphisms

For arbitrary reachable marking \( m_1 \in [\widehat{m}_1] \) and arbitrary \( t_1 \in T_1 \) there exists some \( m'_1 \in [m_1] \) such that \( m'_1[t_1] \).
Example - Liveness: Producers-Consumers System

\[
\text{PCS}_1 \xrightarrow{Q-\text{Trafo}^*} \text{PCS}_2
\]

\[\text{Result: PCS}_1 \text{ live and } Q \text{ collapsing } \implies \text{PCS}_2 \text{ live.}\]
Example - Liveness: Producers-consumers system, cont.

Transition refinement based on collapsing morphisms [GPU 01].

- *Delivering* refined by *Packing & Delivering*: parallel subprocess
- collapsing morphism: liveness respecting
- collapsing morphisms are related to vicinity respecting morphisms
Example - Liveness: Producers-consumers system, cont.

Refined structure

$PCS_2$

Rule is applied three times

Result:
refined net $PCS_2$ is live by construction
Safety properties are translated by transformation based on collapsing morphisms.

\[ PCS_1 : \quad \Box (P \lor W) \quad \Box (R \lor C) \]

\[ PCS_2 : \quad \Box (P \lor W) \quad \Box ((R \lor \neg A) \lor C) \]

Refinement of a transition in postdomain of place \( R \) in \( PCS_1 \) adds additional term \( \neg A \) into the safety property formula of \( PCS_2 \).

Other kinds of safety property preserving transformations are studied in [Gajewsky, Hoffmann, Padberg 1999, 2000] and [Urbášek 2002].
Related Work - Petri Nets

- Refinement of Petri nets (Desel, Merceron, Glabbeck, Golz)
- Reductions of Petri nets (David, Alla)
- Verification of Petri nets with temporal logic (Damm, Gerstner, Bradfield, Stirling, etc.)
- Verification of workflow systems (Jensen)
- Verification of distributed algorithms with Petri nets (Peuker)
- Categorical transformations between net classes (Parisi-Presicce, Gajewsky, Tavakoli, Urbasek)
- Petri net technology (Weber, Ehrig, Reisig)
- Case study of Information system of Berlin Cardiac Center (Padberg, Ermel)
- Petri nets are monoids (Meseguer, Montanari)
Conclusion

Theory, Applications & Tools

Handbook of Graph Grammars and Computing by Graph Transformation


Translation of Concepts

- Semantics and Concurrency from Petri Nets to Graph Transformation
- Transformation of Graphs/Nets from Graph Transformation to Petri Nets