Modeling with Graph Transformations

Hartmut Ehrig, Ulrike Prange
Technical University of Berlin, Germany
{ehrig,uprange}@cs.tu-berlin.de

Abstract. In this paper we give a general overview of graph grammars and graph transformation as important modeling techniques for several areas in computer science. In particular we present the main ideas of the algebraic approach, which is the basis for a categorical theory of rule-based transformations of high-level structures.

1 General Overview of Graph Transformation

The research area of graph grammars or graph transformation is a discipline of computer science which dates back to the early seventies. Methods, techniques, and results from the area of graph transformations have already been studied and applied in many fields of computer science such as formal language theory, pattern recognition and generation, compiler construction, software engineering, concurrent and distributed systems modeling, database design and theory, logical and functional programming, artificial intelligence, visual modeling, etc.

The wide applicability is due to the fact that graphs are a very natural way of explaining complex situations on an intuitive level. Hence, they are used in computer science almost everywhere, e.g. as data and control flow diagrams, entity relationship and UML diagrams, Petri nets, visualization of software and hardware architectures, evolution diagrams of non-deterministic processes, SADT diagrams, and many more. Like the *token game* for Petri nets, a graph transformation brings dynamic to all these descriptions, since it can describe the evolution of graphical structures. Therefore, graph transformations become attractive as a *modeling and programming paradigm* for complex-structured software and graphical interfaces. In particular, graph rewriting is promising as a comprehensive framework in which the transformation of all these very different structures can be modeled and studied in a uniform way.

Before we go into more detail let us discuss the basic question

What is Graph Transformation?

In fact, graph transformation has at least three different roots

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

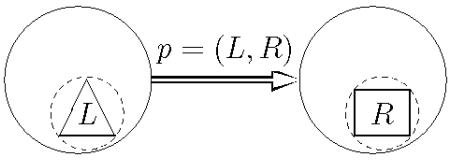

Figure 1: Rule-based Modification of Graphs

Altogether we use the notion of graph transformation to comprise the concepts of graph grammars and graph rewriting. In any case, the main idea of graph transformation is the rule-based modification of graphs as shown in figure 1.

The core of a rule or production $p = (L, R)$ is a pair of graphs (L, R), called left hand side L and right hand side R. Applying the rule $p = (L, R)$ means to find a match of L in the source graph and to replace L by R leading to the target graph of the graph transformation. The main technical problem is how to connect R with the context in the target graph. In fact, there are different solutions how to handle this problem leading to different graph transformation approaches, which are summarized below.

Overview of Different Approaches

The main graph grammar and graph transformation approaches developed in the literature so far are presented in (Rozenberg 1997).

1. The *node label replacement approach*, mainly developed by Rozenberg, Engelfriet and Janssens, allows replacing a single node as the left hand side L by an arbitrary graph R. The connection of R with the context is determined by embedding rules depending on node labels.
2. The *hyperedge replacement approach*, mainly developed by Habel, Kreowski and Drewes, has as the left hand side L a labeled hyperedge, which is replaced by an arbitrary hypergraph R with designated attachment nodes corresponding to the nodes of L. The gluing of R with the context at the corresponding attachment nodes leads to the target graph.
3. The *algebraic approach* is based on pushout constructions, where pushouts are used to model the gluing of graphs. In fact, there are two main variants of the algebraic approach, the double and the single pushout approach. The double pushout approach, mainly developed by Ehrig, Schneider and the Berlin- and Pisa-groups, is introduced in section 2.
4. The *logical approach*, mainly developed by Courcelle and Bouderon, allows expressing graph transformation and graph properties in monadic second order logic.
5. The *theory of 2-structures* was initiated by Rozenberg and Ehrenfeucht as a framework for decomposition and transformation of graphs.
6. The *programmed graph replacement approach* by Schürr used programs in order to control the nondeterministic choice of rule applications.

Aims and Paradigms for Graph Transformation

Computing was originally done on the level of the *von Neumann Machine* which is based on machine instructions and registers. This kind of low level computing was considerably improved by assembler and high level imperative languages. From the conceptual - but not yet from the efficiency point of view - these languages were further improved by functional and logical programming languages. This newer kind of computing is mainly based on term rewriting, which - in the terminology of graphs and graph transformations - can be considered as a concept of tree transformations. Trees, however, do not allow sharing of common substructures, which is one of the main reasons for efficiency problems concerning functional and logical programs. This leads to consider graphs rather than trees as the fundamental structure of computing.

The main idea is to advocate graph transformations for the whole range of computing. Our concept of *Computing by Graph Transformations* is not limited to programming but includes also specification and implementation by graph transformation as well as graph algorithms and computational models and computer architectures for graph transformations.

This concept of *Computing by Graph Transformations* has been developed as a basic paradigm in the ESPRIT Basic Research Actions COMPUGRAPH and APPLIGRAPH as well as in the TMR Network GETGRATS in the years 1990-2002. It can be summarized in the following way:

Computing by graph transformation is a fundamental concept for
- programming
- specification
- concurrency
- distribution
- visual modeling
- model transformation.

The aspect to support visual modeling by graph transformation is one of the main intentions of the ESPRIT TMR Network SEGRAVIS (2002-2006). In fact, there is a wide range of applications to support visual modeling techniques, especially in the context of the UML, by graph transformation techniques. A state of the art report for applications, languages and tools for graph transformation on the one hand and for concurrency, parallelism and distribution on the other hand is given in (Ehrig, Engels, Kreowski & Rozenberg 1999) and (Ehrig et al. 1999).

2 Main Ideas of the Algebraic Graph Transformation Approach

As mentioned already above, the algebraic graph transformation approach is based on pushout constructions, where pushouts are used to model the gluing of graphs. In the algebraic approach, initiated by Ehrig, Pfender and Schneider in (Ehrig et al. 1973), two gluing constructions are used to model a graph transformation step as shown in figure 3. For this reason this approach is also known as the double-pushout approach, short DPO approach, in contrast to the single-pushout (SPO) approach.

The DPO Approach

In the DPO approach roughly spoken, a production is given by $p = (L, K, R)$, where L and R are the left and right hand side graphs and K is the common interface of L and R. Given a production $p = (L, K, R)$ and a context graph D, which includes also the interface K, the source graph G of a graph transformation $G \Rightarrow H$ via p is given by the gluing of L and D via K, written $G = L +_K D$, and the target graph H by the gluing of R and D via K, written $H = R +_K D$. More precisely we use graph morphisms $K \rightarrow L$, $K \rightarrow R$ and $K \rightarrow D$ to express how K is included in L, R and D respectively. This allows to define the gluing constructions $G = L +_K D$ and $H = R +_K D$ as pushout constructions (1) and (2) leading to a double pushout in figure 2.

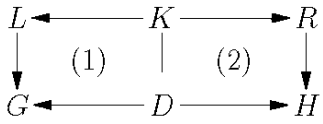

Figure 2: DPO Graph Transformation

A typical example of a DPO graph transformation step is given in figure 3, corresponding to the general scheme in figure 2. Note that in diagram (PO1) G is the gluing of the graphs L and D along K, where the numbering of the nodes indicates how K is embedded into L resp. D in (PO1). Similarly H is the gluing of R and D along K in (PO2) leading to a graph transformation $G \Rightarrow H$ via p. In fact, the diagrams (PO1) and (PO2) are pushouts in the category **Graphs** of graphs and graph morphisms.

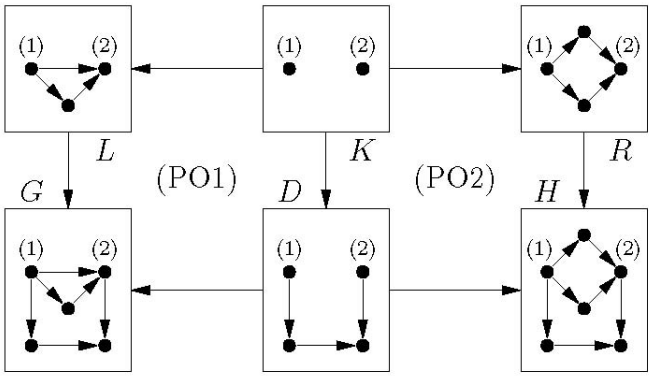

Figure 3: Example of DPO Graph Transformation

The Algebraic Roots

A graph $G = (V, E, s, t)$ is a special case of an algebra with two base sets V (vertices) and E (edges) and operations $s : E \rightarrow V$ (source) and $t : E \rightarrow V$ (target). Graph morphisms $f : G_1 \rightarrow G_2$ are special cases of algebra homomorphisms $f = (f_V : V_1 \rightarrow V_2, f_E : E_1 \rightarrow E_2)$. This means that f_V and f_E are required to be compatible with the operations, i.e. $f_V \circ s_1 = s_2 \circ f_E$ and $f_V \circ t_1 = t_2 \circ f_E$. In figure 3 all arrows between the boxes are graph morphisms. Moreover the gluing construction of

graphs can be considered as an algebraic quotient algebra construction. This algebraic view of graphs and graph transformations is one of the main ideas of the algebraic graph transformation approach introduced in (Ehrig et al. 1973).

From Graphs to High-Level Structures

The algebraic approach of graph transformation is not restricted to graphs of the form $G = (V, E, s, t)$ considered above, but has been generalized to a large variety of different types of graphs and other kinds of high-level structures, like labeled graphs, typed graphs, hypergraphs, attributed graphs, Petri nets and algebraic specifications. This extension from graphs to high-level structures - in contrast to strings and trees as lower-level structures - was initiated in (Ehrig et al. 1991) leading to the theory of high-level replacement (HLR) systems. In (Ehrig et al. 2004) the concept of high-level replacement systems was joined with that of adhesive categories introduced in (Lack & Sobociński 2004) leading to the concept of adhesive HLR categories and systems. The theory of adhesive HLR systems can be instantiated in particular to typed attributed graph transformation systems, which are especially important for applications to visual languages and software engineering.

References

Ehrig, H., Engels, G., Kreowski, H.-J. & Rozenberg, G., eds (1999), *Handbook of Graph Grammars and Computing by Graph Transformation, Vol 2: Applications, Languages and Tools*, World Scientific.

Ehrig, H., Habel, A., Kreowski, H. & Parisi-Presicce, F. (1991), 'Parallelism and Concurrency in High-Level Replacement Systems', *Mathematical Structures in Computer Science* 1 (3), 361–404.

Ehrig, H., Habel, A., Padberg, J. & Prange, U. (2004), Adhesive High-Level Replacement Categories and Systems, *in* H. Ehrig, G. Engels, F. Parisi-Presicce & G. Rozenberg, eds, 'Proceedings of ICGT 2004', Vol. 3256 of *LNCS*, Springer, pp.144–160.

Ehrig, H., Kreowski, H., Montanari, U. & Rozenberg, G., eds (1999), *Handbook of Graph Grammars and Computing by Graph Transformation, Vol 3: Concurrency, Parallelism and Distribution*, World Scientific.

Ehrig, H., Pfender, M. & Schneider, H. (1973), Graph Grammars: an Algebraic Approach, *in* '14th Annual IEEE Symposium on Switching and Automata Theory', IEEE, pp. 167– 180.

Lack, S. & Sobociński, P. (2004), Adhesive Categories, in I. Walukiewicz, ed., 'Proceedings of FOSSACS 2004', Vol. 2987 of *LNCS*, Springer, pp.273–288.

Rozenberg, G., ed. (1997), *Handbook of Graph Grammars and Computing by Graph Transformation, Vol 1: Foundations*, World Scientific.