# On the Interleaving Semantics of Transformation Units – A Step into GRACE

Hans-Jörg Kreowski, Sabine Kuske*
Universität Bremen, Fachbereich 3
Postfach 33 04 40, D-28334 Bremen
email: {kreo,kuske}@informatik.uni-bremen.de

**Abstract.** The aim of the paper is to introduce the notion of a transformation unit together with its interleaving semantics and to study it as a means of constructing large graph transformation systems from small ones in a structured and systematic way. A transformation unit comprises a set of rules, descriptions of initial and terminal graphs, and a control condition. Moreover, it may import other transformation units for structuring purposes. Its semantics is a binary relation between initial and terminal graphs which is given by interleaving sequences. As a generalization of ordinary derivations, an interleaving sequence consists of direct derivation steps interleaved with calls of imported transformation units. It must obey the control condition and may be seen as a kind of structured derivation. The introduced framework is independent of a particular graph transformation approach and, therefore, it may enhance the usefulness of graph transformations in many contexts.

## 1 Introduction

The significance of graphs and rules in many areas of computer science is evident: On the one hand, graphs constitute appropriate means for the description of complex relationships between objects. Trees, forests, Petri nets, circuit diagrams, finite automata, flow charts, data flow graphs, and entity-relationship diagrams are some typical examples. On the other hand, rules are used to describe "permitted" manipulations on objects like, for example, in the areas of functional and logic programming, formal languages, algebraic specification, theorem proving, and rule-based systems.

The intention of bringing graphs and rules together – motivated by several application areas – has led to the theory of *graph grammars* and *graph transformation* (see [CER79, ENR83, ENRR87, EKR91, SE93] for a survey). A wide spectrum of approaches exists within this theory and some of them are implemented (see, for example, PROGRES [Sch91a, Sch91b], Graph$^{\mathrm{Ed}}$ [Him91], Dactl [GKS91], and AGG [LB93, TB93]).

With the aim of enhancing the usefulness of graph transformation, we propose

*This work has been supported by COMPUGRAPH II, ESPRIT Basic Research Working Group 7183.

a new approach-independent structuring method for building up large systems of graph transformation rules from small pieces. The method is based on the notion of a transformation unit and its interleaving semantics. A transformation unit is allowed to use other units such that a system of graph transformation rules can be structured hierarchically and existing transformation units can be re-used. The transformation unit is a basic concept of the new graph and rule centered language GRACE that is being developed by researchers from Aachen, Berlin, Bremen, and Leiden (see also [Kre95, Kus95, Sch95]). Nevertheless, the notion is meaningful in its own right because – independently of GRACE – it can be employed as a structuring principle in most graph transformation approaches one encounters in the literature where graph transformation is also called graph rewriting.

The paper is organized as follows. Section 2 introduces the notion of a transformation unit together with its interleaving semantics. In Section 3, the concepts of a transformation unit are illustrated with an example. Section 4 presents how some operations on binary relations can be modelled by suitable operations on transformation units. In Section 5, some normal forms of transformation units are considered. The paper ends with some concluding remarks. To avoid wrong expectations, we would like to point out that the goal of the paper is to shed some light on the usefulness of the introduced structuring method rather than to come up with deep theory.

## 2 Transformation Units with Interleaving Semantics

The key operation in graph rewrite approaches is the transformation of a graph into a graph by applying a rule such that sets of rules specify graph transformations by iterated rule applications. This derivation process is highly nondeterministic in general and runs on arbitrary graphs which is both not always desirable. For example, if one wants to generate graph languages, one may start in a particular axiom and end with certain terminal objects only. Or if a more functional behaviour is required, one may prefer to control the derivation process and to cut down its non-determinism. The latter can be achieved by control mechanisms for the derivation process like application conditions or programmed graph transformation (see, e.g., [Bun79, Nag79, EH86, KR90, MW91, Sch91a, SZ91, Kre93, LM93, HHT95], cf. also [DP89] for regulation concepts in string grammars) and the former by the use of graph class expressions that specify subclasses of graphs. Moreover, in practical cases, one may have to handle hundreds or thousands of rules which cannot be done in a transparent and reasonable way without a structuring principle.

To cover all these aspects, we introduce the notion of a transformation unit that allows to specify new rules, initial and terminal graphs, as well as a control condition, and to import other transformation units. Semantically, a transformation unit describes a graph transformation, i.e. a binary relation on graphs given by the interleaving of the graph transformations with each other and with rule applications. Moreover, interleaving sequences must start in initial graphs, end in terminal graphs and satisfy the control condition. If nothing is imported, the interleaving semantics coincides with the derivation relation.

To make the concept independent of a particular graph rewriting framework, we assume an abstract notion of a graph transformation approach comprising a class of graphs, a class of rules, a rule application operator, a class of graph class expressions, and a class of control conditions. The semantic effect of control conditions is described depending on so-called environments that associate binary relations on graphs to identifiers. In this way, it can be defined without forward reference to transformation units. Intuitively, one may think of an environment as rules with their corresponding direct derivation relations (cf. 2.5).

## 2.1 Graph Transformation Approach

A *graph transformation approach* is a system $\mathcal{A} = (\mathcal{G}, \mathcal{R}, \Longrightarrow, \mathcal{E}, \mathcal{C})$ where

- $\mathcal{G}$ is a class of *graphs*,
- $\mathcal{R}$ is a class of *rules*,
- $\Longrightarrow$ is a *rule application operator* yielding a binary relation $\underset{r}{\Longrightarrow} \subseteq \mathcal{G} \times \mathcal{G}$ for every $r \in \mathcal{R}$,
- $\mathcal{E}$ is a class of *graph class expressions* such that each $e \in \mathcal{E}$ specifies a subclass $SEM(e) \subseteq \mathcal{G}$, and
- $\mathcal{C}$ is a class of *elementary control conditions* over some set $ID$ of identifiers such that each $c \in \mathcal{C}$ specifies a binary relation $SEM_E(c) \subseteq \mathcal{G} \times \mathcal{G}$ for each mapping $E\colon ID \longrightarrow \mathcal{P}(\mathcal{G} \times \mathcal{G})$.

A pair $(G, G') \in \underset{r}{\Longrightarrow}$, usually written as $G \underset{r}{\Longrightarrow} G'$, is called a *direct derivation* from $G$ to $G'$ through $r$. For a set $P \subseteq \mathcal{R}$ the union of all relations $\underset{r}{\Longrightarrow}$ $(r \in P)$ is denoted by $\underset{P}{\Longrightarrow}$ and its reflexive and transitive closure by $\overset{*}{\underset{P}{\Longrightarrow}}$. A pair $(G, G') \in \overset{*}{\underset{P}{\Longrightarrow}}$, usually written as $G \overset{*}{\underset{P}{\Longrightarrow}} G'$, is called a *derivation* from $G$ to $G'$ over $P$. A mapping $E\colon ID \longrightarrow \mathcal{P}(\mathcal{G} \times \mathcal{G})$ is called an *environment*. In the following, we will use boolean expressions over $\mathcal{C}$ as *control conditions* with elementary control conditions as basic elements and disjunction, conjunction, and negation as boolean operators. Moreover, we assume the constant *true*. The semantic relations of elementary control conditions are easily extended to boolean expressions by

$$\begin{aligned}
&SEM_E(true) = \mathcal{G} \times \mathcal{G},\\
&SEM_E(e_1 \vee e_2) = SEM_E(e_1) \cup SEM_E(e_2),\\
&SEM_E(e_1 \wedge e_2) = SEM_E(e_1) \cap SEM_E(e_2),\\
&SEM_E(\overline{e}) = \mathcal{G} \times \mathcal{G} - SEM_E(e).
\end{aligned}$$

The set of control conditions over $\mathcal{C}$ is denoted by $\mathcal{B}(\mathcal{C})$.

Note that we refer to the meaning of graph class expressions and control conditions by the overloaded operator *SEM*. This should do no harm because it is always clear from the context which is which.

All the graph grammar and graph rewrite approaches one encounters in the literature provide notions of graphs and rules and a way of directly deriving a graph from a graph by applying a rule (cf. e.g. [Ehr79, Nag79, JR80, Cou90, KR90, Sch91b, Him91, Hab92, Löw93]). Therefore, all of them can be considered as graph transformation approaches in the above sense, if one chooses the components $\mathcal{E}$ and $\mathcal{C}$ in some standard way. The singleton set $\{all\}$ with $SEM(all) = \mathcal{G}$ may provide the only graph class expression, and the class of elementary control conditions may be empty. Non-trivial choices for $\mathcal{E}$ and $\mathcal{C}$ are discussed in the next paragraphs.

## 2.2 Graph Class Expressions

There are various standard ways to choose graph class expressions that can be combined with many classes of graphs and hence used in many graph transformation approaches.

1. In most cases, one deals with some kind of finite graphs with some explicit representations. Then single graphs (or finite enumerations of graphs) may serve as graph class expressions. Semantically, each graph $G$ represents itself, i.e. $SEM(G) = \{G\}$. The axiom of a graph grammar is a typical example of this type.

2. A graph $G$ is *reduced* with respect to a set of rules $P \subseteq \mathcal{R}$ if there is no $G' \in \mathcal{G}$ with $G \underset{r}{\Longrightarrow} G'$ and $r \in P$. In this way, $P$ can be considered as a graph class expression with $SEM(P) = RED(P)$ being the set of all reduced graphs with respect to $P$. Reducedness is often used in term rewriting and term graph rewriting as a halting condition.

3. If $\mathcal{G}$ is a class of labelled graphs with label alphabet $\Sigma$, then a set $T \subseteq \Sigma$ is a suitable graph class expression specifying the graph class $SEM(T) = \mathcal{G}_T$ consisting of all graphs labelled in $T$ only. This way of distinguishing terminal objects is quite popular in formal language theory.

4. Graph theoretic properties can be used as graph class expressions. In particular, monadic second order formulas for directed graphs or hypergraphs or undirected graphs are suitable candidates (see e.g. [Cou90]).

## 2.3 Control Conditions

Every description of a binary relation on graphs may be used as a control condition. Here, we introduce some typical examples.

1. Let $E{:}\,ID \rightarrow \mathcal{P}(\mathcal{G} \times \mathcal{G})$ be an environment. Then $E$ can be extended to the powerset of the set of strings over $ID$ in a natural, straight-forward way. $\widehat{E}{:}\,\mathcal{P}(ID^*) \rightarrow \mathcal{P}(\mathcal{G} \times \mathcal{G})$ is defined by $\widehat{E}(L) = \bigcup_{w \in L} \overline{E}(w)$ for $L \subseteq ID^*$ where $\overline{E}{:}\,ID^* \rightarrow \mathcal{P}(\mathcal{G} \times \mathcal{G})$ is recursively given by $\overline{E}(\lambda) = \Delta\mathcal{G}$ [1], and $\overline{E}(xv) = E(x) \circ$

[1] $\Delta\mathcal{G}$ denotes the identity relation on $\mathcal{G}$.

$\overline{E}(v)$ [2] for $x \in ID$ and $v \in ID^*$. Hence, every grammar, every automaton and every expression specifying a language over $ID$ can serve as a control condition. If it is not necessary to distinguish between syntax and semantics, we may represent a control condition of this language type by the language itself with $SEM_E(L) = \widehat{E}(L)$ for all $L \subseteq ID^*$ and $E : ID \rightarrow \mathcal{P}(\mathcal{G}\times\mathcal{G})$. In this case, the class of elementary control conditions is $\mathcal{P}(ID^*)$.
2. In particular, the class of regular expressions over $ID$ can be used for this purpose. For explicit use below, $REG(ID)$ is recursively given by $\emptyset, \epsilon \in REG(ID)$[3], $ID \subseteq REG(ID)$, and $(e_1 \,;\, e_2)$, $(e_1 \,|\, e_2)$, $(e^*) \in REG(ID)$ if $e, e_1, e_2 \in REG(ID)$. Outermost parentheses of regular expressions will be omitted in the following.
3. Let $\mathcal{E}$ be a class of graph class expressions. Then each pair $(e_1, e_2) \in \mathcal{E} \times \mathcal{E}$ defines a binary relation on graphs by $SEM((e_1, e_2)) = SEM(e_1) \times SEM(e_2)$ and, therefore, it can be used as a control condition which is independent of the choice of an environment, i.e. $SEM_E((e_1, e_2)) = SEM((e_1, e_2))$ for all environments $E$.

## 2.4 Transformation Units

A transformation unit encapsulates a specification of initial graphs, a set of transformation units to be used, a set of rules, a control condition, and a specification of terminal graphs.

Let $\mathcal{A} = (\mathcal{G}, \mathcal{R}, \Longrightarrow, \mathcal{E}, \mathcal{C})$ be a graph transformation approach. A *transformation unit* over $\mathcal{A}$ is a system $trut = (I, U, R, C, T)$ where $I, T \in \mathcal{E}$, $U$ is a set of transformation units over $\mathcal{A}$, $R \subseteq \mathcal{R}$, and $C \in \mathcal{B}(\mathcal{C})$. The components of *trut* may be denoted by $U_{trut}$, $I_{trut}$, $R_{trut}$, $C_{trut}$, and $T_{trut}$, respectively.

This is meant as a recursive definition of the set $\mathcal{T}_{\mathcal{A}}$ of transformation units over $\mathcal{A}$. Hence, initially, $U$ must be chosen as the empty set yielding unstructured transformation units without import that may be used in the next iteration, and so on.

If $I$ specifies a single graph (cf. 2.2.1), $U$ is empty, and $C$ is the constant *true*, one gets the usual notion of a graph grammar (in which approach ever) as a special case of transformation units.

## 2.5 Interleaving Semantics

The operational semantics of a transformation unit is a graph transformation, i.e. a binary relation on graphs containing a pair $(G, G')$ of graphs if, first, $G$ is an initial graph and $G'$ is a terminal graph, second, $G'$ can be obtained from $G$ by interleaving direct derivations with the graph transformations specified by the used transformation units, and third, the pair is allowed by the control condition.

Let $trut = (I, U, R, C, T)$ be a transformation unit over the graph transformation approach $\mathcal{A} = (\mathcal{G}, \mathcal{R}, \Longrightarrow, \mathcal{E}, \mathcal{C})$. Assume that the set $ID$ of identifiers associated

[2] Given $\varrho, \varrho' \subseteq \mathcal{G} \times \mathcal{G}$, the sequential composition of $\varrho$ and $\varrho'$ is defined as usual by $\varrho \circ \varrho' = \{(G, G'') \,|\, (G, G') \in \varrho \text{ and } (G', G'') \in \varrho' \text{ for some } G' \in \mathcal{G}\}$.

[3] While $\emptyset$ denotes the empty set $\{\}$, the expression $\epsilon$ denotes the regular set $\{\lambda\}$. We prefer a direct reference to $\{\lambda\}$ rather than to use $\emptyset^*$.

to $C$ contains the disjoint union of $U$ and $R$. Let the interleaving semantics $SEM(t) \subseteq \mathcal{G} \times \mathcal{G}$ for $t \in U$ be already defined. Let $E(trut): ID \to \mathcal{P}(\mathcal{G} \times \mathcal{G})$ be defined by $E(trut)(r) = \underset{r}{\Longrightarrow}$ for $r \in R$, $E(trut)(t) = SEM(t)$ for $t \in U$, and $E(trut)(id) = \{\}$, otherwise. Then the *interleaving semantics* $SEM(trut)$ of *trut* consists of all pairs $(G, G') \in \mathcal{G} \times \mathcal{G}$ such that

1. $G \in SEM(I)$ and $G' \in SEM(T)$,
2. there are graphs $G_0, \ldots, G_n \in \mathcal{G}$ with $G_0 = G$, $G_n = G'$ and for $i = 1, \ldots, n$, $G_{i-1} \underset{R}{\Longrightarrow} G_i$ or $(G_{i-1}, G_i) \in SEM(t)$ for some $t \in U$,
3. $(G, G') \in SEM_{E(trut)}(C)$.

The sequence of graphs in point 2 is called an *interleaving sequence in trut from G to G'*. Let $RIS(trut)$ denote the binary relation given by interleaving sequences, i.e. the set of all pairs of graphs $(G, G') \in \mathcal{G} \times \mathcal{G}$ such that there is an interleaving sequence in *trut* from $G$ to $G'$ as in point 2. Then the interleaving semantics of *trut* is defined as the intersection of $RIS(trut)$ with $SEM(I) \times SEM(T)$ and $SEM_{E(trut)}(C)$. Note that all three relations may be incomparable with each other. For example, $(G, G') \in SEM_{E(trut)}(C)$ does not imply in general that there is an interleaving sequence in *trut* from $G$ to $G'$, and vice versa.

A control condition $C$ specifies a binary predicate depending on other binary graph relations through the notion of environments, but independent of a particular transformation unit. As a component of *trut*, only the *environment of trut* given by $E(trut)$ is effective, meaning that $C$ can restrict the semantics by specifying certain properties of the direct derivation relations of rules in *trut*, the interleaving semantics of imported transformation units, and the interrelation of all of them. If transformation units are used in a specification language, it will be more realistic to assume that $ID$ is a countable set of predefined identifiers out of which the elements of $U$ and $R$ are named, rather than to assume that $U$ and $R$ are subsets of $ID$. But we prefer here to avoid an explicit naming mechanism because it is not essential for the introduced concepts.

The definition of the interleaving semantics follows the recursive definition of transformation units. Hence, its well-definedness follows easily by an induction on the structure of transformation units provided that the import structure is acyclic. Initially, if $U$ is empty, an interleaving sequence is just a derivation such that one gets in this case

$$SEM(trut) = \overset{*}{\underset{R}{\Longrightarrow}} \cap (SEM(I) \times SEM(T)) \cap SEM_{E(trut)}(C).$$

In other words, interleaving semantics generalizes the ordinary operational semantics of sets of rules given by derivations. If, furthermore, $C$ is *true* and $I$ is a single graph, then *trut* is a graph grammar (cf. 2.4), and the following holds for its interleaving semantics:

$$SEM(trut) = \{(I, G) \mid G \in L(trut)\}$$

with $L(trut) = \{G \in SEM(T) \mid I \overset{*}{\underset{R}{\Longrightarrow}} G\}$. In this sense, the interleaving semantics covers the usual notion of graph languages generated by graph grammars.

## 2.6 Application Sequences

In interleaving sequences, rules are applied and imported transformation units are called in some order. Such sequences of applied rules and called transformation units help to clarify the role of control conditions of the language type as defined in 2.3.1.

Let *trut* $= (I, U, R, C, T)$ be a transformation unit over the graph transformation approach $\mathcal{A} = (\mathcal{G}, \mathcal{R}, \Longrightarrow, \mathcal{E}, \mathcal{C})$. Assume that $U$ and $R$ are disjoint subsets of the set $ID$ associated to $\mathcal{C}$. Then $x_1 \ldots x_n \in (U \cup R)^*$ $(x_i \in U \cup R)$ is called an *application sequence* of $(G, G') \in \mathcal{G} \times \mathcal{G}$ if there is an interleaving sequence $G_0, \ldots, G_n$ with $G_0 = G$, $G_n = G'$ and, for $i = 1, \ldots, n$, $G_{i-1} \underset{x_i}{\Longrightarrow} G_i$ if $x_i \in R$ and $(G_{i-1}, G_i) \in SEM(x_i)$ if $x_i \in U$. In the case $n = 0$, the application sequence is the empty string $\lambda$. Using these notions and notations, the following observation states that a language over $U \cup R$, used as a control condition due to 2.3.1, controls the order in which rules are applied and imported transformation units are actually used.

**Observation**
Let $\mathcal{C} = \mathcal{P}(ID^*)$ be the class of control conditions of language type, and let *trut* $= (I, U, R, L, T)$ with $L \subseteq (U \cup R)^* \subseteq ID^*$. Then for all $G, G' \in \mathcal{G}$, the following statements are equivalent.

1. $(G, G') \in SEM(trut)$.
2. $(G, G') \in SEM_{E(trut)}(L) \cap SEM(I) \times SEM(T)$.
3. There is an application sequence $w$ of $(G, G')$ with $w \in L$ and $(G, G') \in SEM(I) \times SEM(T)$.

**Proof.** Let $(G, G') \in SEM(trut)$. Then by definition, there is an interleaving sequence in *trut* from $G$ to $G'$, $(G, G) \in SEM_{E(trut)}(L)$, and $(G, G) \in SEM(I) \times SEM(T)$. Hence, point 1 implies point 2.

To show that point 2 implies point 3 and that point 3 implies point 1, we prove first the following claim: $(G, G') \in SEM_{E(trut)}(L)$ iff there is an application sequence $w \in L$ of $(G, G')$.

By definition, we have $(G, G') \in SEM_{E(trut)}(L) = \widehat{E(trut)}(L)$ iff $(G, G') \in \overline{E(trut)}(w)$ for some $w \in L$. We show now by induction on the structure of $w$ that $(G, G') \in \overline{E(trut)}(w)$ iff $w$ is an application sequence of $(G, G')$.

If $w = \lambda$, we get $(G, G') \in \overline{E(trut)}(\lambda) = \Delta\mathcal{G}$, i.e. $G = G'$ which defines the interleaving sequence with $n = 0$ and $G_0 = G = G'$ and the corresponding application sequence $\lambda$ (and vice versa).

Assume now that the statement holds for $v \in (U \cup R)^*$.

And consider $w = xv$ with $x \in U \cup R$. Then $(G, G') \in \overline{E(trut)}(xv) = \overline{E(trut)}(x) \circ \overline{E(trut)}(v)$, and there is some $\overline{G} \in \mathcal{G}$ with $(G, \overline{G}) \in E(trut)(x)$ and $(\overline{G}, G') \in \overline{E(trut)}(v)$. The latter implies by induction that $v$ is an application sequence of $(\overline{G}, G')$ such that there is an interleaving sequence $G_0, \ldots, G_n$ with $\overline{G} = G_0$ and $G' = G_n$. The former means $G \underset{x}{\Longrightarrow} \overline{G}$ if $x \in R$ and $(G, \overline{G}) \in SEM(x)$

if $x \in U$. Altogether, $G, G_0, \ldots, G_n$ defines an interleaving sequence with $xv$ as corresponding application sequence. Conversely, an application sequence $xv$ of $(G, G')$ is related to an interleaving sequence $G_0, \ldots, G_n$ with $G = G_0$, $G' = G_n$ and, in particular, $G_0 \underset{x}{\Longrightarrow} G_1$ if $x \in R$ or $(G_0, G_1) \in \mathit{SEM}(x)$ if $x \in U$ such that $(G, G_1) \in E(\mathit{trut})(x)$ in any case. Moreover, $v$ is an application sequence of $(G_1, G_n)$ because $G_1, \ldots, G_n$ is an interleaving sequence. By induction hypothesis, we get $(G_1, G') \in E(\mathit{trut})(v)$. The composition yields $(G, G') \in E(\mathit{trut})(x) \circ E(\mathit{trut})(v) = E(\mathit{trut})(xv)$. This completes the proof of the claim.

From the just proved claim follows directly that point 2 implies point 3.

Furthermore, let $w \in L$ be an application sequence of $(G, G')$ with $(G, G') \in \mathit{SEM}(I) \times \mathit{SEM}(T)$. Then by definition, we have that there is an interleaving sequence in *trut* from $G$ to $G'$ with $(G, G') \in \mathit{SEM}(I) \times \mathit{SEM}(T)$, and by the claim, $w \in \mathit{SEM}_{E(\mathit{trut})}(L)$. Hence, $(G, G') \in \mathit{SEM}(\mathit{trut})$. This completes the proof. □

## 2.7 Initial and Terminal Graphs

As pointed out in 2.3.3, two graph class expressions form a control condition. Hence, the specifications of initial and terminal graphs may be handled as a control condition. In the interleaving semantics of a transformation unit, the product of initial and terminal graphs as well as the relation specified by the actual control condition must be intersected with the relation established by the interleaving sequences. This proves the following observation.

**Observation**
Let $\mathit{trut} = (I, U, R, C, T)$ be a transformation unit over $\mathcal{A} = (\mathcal{G}, \mathcal{R}, \Longrightarrow, \mathcal{E}, \mathcal{C})$. Then $\mathit{trut}' = (\mathit{all}, U, R, C \wedge (I, T), \mathit{all})$ is a transformation unit over $\mathcal{A}' = (\mathcal{G}, \mathcal{R}, \Longrightarrow, \{\mathit{all}\}, \mathcal{C} \cup \mathcal{E} \times \mathcal{E})$ with $\mathit{SEM}(\mathit{trut}) = \mathit{SEM}(\mathit{trut}')$.

This means that the components $I$ and $T$ of a transformation unit are not necessary. Nevertheless, we keep them because we would like to distinguish between input and output conditions and other control conditions explicitly and to emphasize the different intuitions behind.

# 3 Butterfly Networks – an Example

In this section, we illustrate the concepts of transformation units by specifying the set of butterfly networks. Such high-bandwidth processor organizations – well-known from the area of VLSI theory – are well suited for performing highly parallel computations (see e.g. Ullman [Ull84] and Lengauer [Len90]).

A *butterfly network* of size $k$ for some $k \in \mathbb{N}$ consists of $(k+1)$ *ranks* of $2^k$ nodes each. Let $v_{ir}$ be the $i^{th}$ node on rank $r$ and let $b_{i_1} \ldots b_{i_k}$ be the binary representation of $i$ ($0 \leq i < 2^k$, $0 \leq r \leq k$). Then for $r > 0$, $v_{ir}$ is directly connected to $v_{jr-1}$ if either $i = j$ or $b_{i_1} \ldots b_{i_k}$ is equal to the binary representation of $j$ up to $b_{i_r}$. In the following, we call the nodes on rank 0 *bottom nodes* and label them with $b$. All other nodes of a butterfly network are unlabelled.

To make the paper self-contained, we tailor a graph transformation approach for this example. This approach can be easily expressed in many of the general graph transformation approaches in the literature, like, e.g., PROGRES (see [Sch91a]) or graph grammars with negative application conditions (see [HHT95]).

- A *graph* is a system $G = (V_G, E_G, l_G, m_G)$ where $V_G$ is the set of *nodes*, $E_G$ is the set of *edges* being 2-element subsets of $V_G$, $l_G\colon V_G \to C$, and $m_G\colon E_G \to C$ are labelling functions for nodes and edges respectively with $C = \{*, b, c\}$. The symbol $*$ stands for *unlabelled*, $b$ for *bottom*, and $c$ for *copied*. Subgraph relation and isomorphy are defined and denoted in the usual way.
- A *rule* is a triple $r = (N, L, R)$ of graphs with $L \subseteq N$ and $V_L \subseteq V_R$, i.e. $L$ is a subgraph of $N$, and each node of $L$ is a node of $R$, but its label in $L$ may be different from that in $R$.
- A *rule* $r = (N, L, R)$ is applied to a graph $G$ according to the following steps.
  1. Choose $L' \subseteq G$ with $L' \cong L$.
  2. Check the negative context condition: It fails if there is any $N' \subseteq G$ with $N' \cong N$ and $L' \subseteq N'$.
  3. Remove $E_{L'}$ from $G$.
  4. Glue the remaining graph with $R$ by merging each node of $v \in V_L (\subseteq V_R)$ with the corresponding node in $L'$ and labelling it with $l_R(v)$.
- We use the constant *all*, the graph consisting of a single $b$-labelled node and sets of rules (to specify reduced graphs) as graph class expressions.
- We use regular expressions over the alphabet $\{copy, next\_rank, copy\_items, delete\}$ as elementary control conditions.

In the following, transformation units are presented by indicating the components with respective keywords. Trivial components (i.e. no import, no rules, the graph class expression *all*, and the control condition *true*) are omitted. A rule $r = (N, L, R)$ is represented in the form $N \to R$ where the items of $N$ that do not belong to $L$ are dotted. Two nodes in $r$ are drawn with the same shape and fill style if and only if they are equal. The label $*$ is not depicted.

The transformation unit *butterfly* uses the transformation units *copy* and *next_rank* and applies them in that order arbitrarily often. The initial graph is the butterfly network of size 0. After $k$ calls of *copy* and *next_rank* a butterfly network of size $k$ is generated.

*butterfly*

initial: ● $b$
uses: *copy*, *next_rank*
conds: $(copy\ ;\ next\_rank)^*$

If the input graph of the transformation unit *copy* is a butterfly network $B$ of size $k$, it is copied and, additionally, a $c$-labelled edge is inserted between each bottom node and its copy. The output graph of *copy* without the $c$-labelled edges and the $b$-labels corresponds, roughly speaking, to a butterfly network of size $k+1$ where the bottom nodes together with all connections to them are missing. The transformation unit *next_rank* transforms this graph into a butterfly network of size $k+1$ by adding these missing bottom nodes and connecting edges.

The transformation unit *copy* uses the transformation units *copy_items* and *delete* which are applied exactly once in this order. If the butterfly network $B$ is the input of the transformation unit *copy_items*, then $B$ is transformed into a graph $B'$ by copying each node and edge of $B$ together with its label and generating a $c$-labelled edge between each node of $B$ and its copy. The transformation unit *delete* removes each $c$-labelled edge provided that it connects $*$-labelled nodes. Note that the node labels and the $c$-labelled edges, used in *copy_items*, *delete*, and *next_rank*, serve to control rule application: They prevent undesired multiple rule applications to the same subgraph and mark subgraphs rules may be applied to. The term *reduced* in the terminal components of *copy_items*, *delete*, and *next_rank* indicates that the terminal graphs are reduced with respect to the actual set of rules. This means all the rules of the respective transformation unit are applied as long as possible.

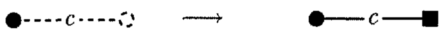
*copy*

uses: *copy_items*, *delete*

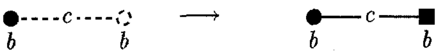
conds: *copy_items* ; *delete*

*copy_items*

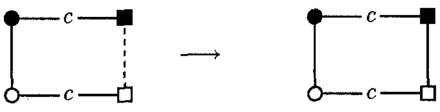
rules: $r_1$:



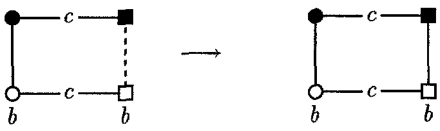
$r_2$:



$r_3$:



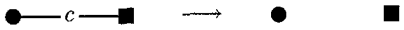
$r_4$:



terminal: *reduced*

*delete*

rules: $r_1$:

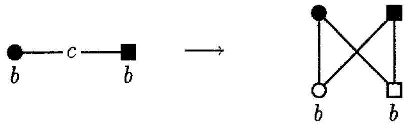


terminal: *reduced*

*next_rank*

rules:

$r_1$:



terminal: *reduced*

Figure 1 shows an interleaving sequence of *butterfly* generating the butterfly networks of size 0, 1, and 2, where $G \longrightarrow_{trut} G'$ means that $(G, G') \in SEM(trut)$ for some transformation unit *trut* and some graphs $G, G'$.

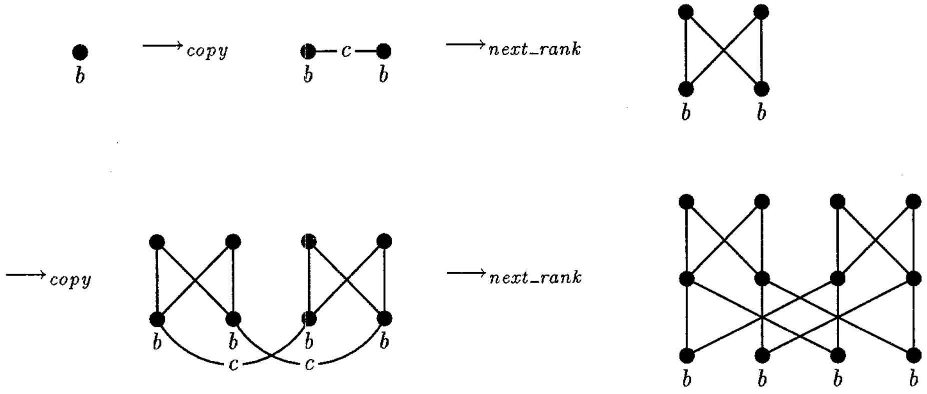


Figure 1: An interleaving sequence of *butterfly*

By induction on the length of the interleaving sequences in *butterfly* and on the size of butterfly networks (using some lemmata concerning the imported transformation units) one can prove the following correctness result.

$(\bullet b, G) \in SEM(butterfly)$ iff $G$ is a butterfly network of some size.

## 4 Operations on Transformation Units

The concept of transformation units may be seen as an operation on transformation units that describes the interleaving of the semantic relations of the imported transformation units with each other and with a derivation relation. This somewhat complicated operation on binary relations is motivated by the idea that transformation units encapsulate hierarchically sets of rules and the interleaving semantics generalizes the derivation process accordingly. But there are many other operations on binary relations like union, conversion, complement, transitive closure, etc. that may be of interest in various situations. Hence, one may wonder whether and how certain operations on binary relations can be achieved by suitable operations on transformation units. We show in this section that various standard operations can be specified in terms of transformation units.

## 4.1 Operations Without Conversion

The definition of the interleaving semantics of transformation units is based on the union, the sequential composition and the reflexive and transitive closure of relations on graphs. If regular expressions are employed as control conditions, these operations can be modelled as constructions on transformation units in an obvious way, because the effect of regular expressions is directly related to them.

**Observation**
Let $\mathcal{A} = (\mathcal{G}, \mathcal{R}, \Longrightarrow, \mathcal{E}, \mathcal{C})$ be a graph transformation approach where $REG(ID) \subseteq \mathcal{C}$. Then for all $t, t' \in \mathcal{T}_{\mathcal{A}}$, one can construct transformation units $trans(t)$, $refl(t)$, $union(t,t')$, $sc(t,t') \in \mathcal{T}_{\mathcal{A}}$ such that $SEM(trans(t)) = SEM(t)^{+}$ [4], $SEM(refl(t)) = SEM(t) \cup \Delta\mathcal{G}$, $SEM(union(t,t')) = SEM(t) \cup SEM(t')$, and $SEM(sc(t,t')) = SEM(t) \circ SEM(t')$,

**Proof.** Define $trans(t)$, $refl(t)$, $union(t,t')$, $sc(t,t')$ as follows.

$trans(t)$
uses: $t$
conds: $t \ ; (t^*)$

$refl(t)$
uses: $t$
conds: $t \mid \epsilon$

$union(t,t')$
uses: $t, t'$
conds: $t \mid t'$

$sc(t,t')$
uses: $t, t'$
conds: $t \ ; t'$

For a regular expression $c \in \mathcal{C}$, let $L(c)$ be the language described by $c$. Then by definition, we get $SEM_{E(trans(t))}(t\,;(t^*)) = E(\widehat{trans}(t))(L(t\,;(t^*))) = E(trans(t))(t) \circ E(\widehat{trans}(t))(L(t^*)) = SEM(t) \circ E(\widehat{trans}(t))(L(t^*))$. By induction on the structure of the words in $L(t^*)$, we get $E(\widehat{trans}(t))(L(t^*)) = (E(trans(t))(t))^*$ [4]. Hence, $SEM_{E(trans(t))}(t\,;(t^*)) = SEM(t) \circ SEM(t)^* = SEM(t)^{+}$. Moreover, from the observation in 2.6 it follows that for all $(G, G') \in SEM_{E(trans(t))}(t\,;(t^*))$, there is an interleaving sequence in $trans(t)$ from $G$ to $G'$. Hence, $SEM(trans(t)) = SEM(t)^{+}$. The correctness of $refl(t)$, $union(t,t')$, $sc(t,t')$ can be shown analogously. □

**Remark**
There is a more general version of this observation for arbitrary regular expressions.

Let $e \in REG(X)$ for $X \subseteq \mathcal{T}_{\mathcal{A}} \cap ID$. Let $\varrho_x \subseteq \mathcal{G} \times \mathcal{G}$ for $x \in X$, and $E_\varrho : ID \to \mathcal{P}(\mathcal{G} \times \mathcal{G})$ given by $E_\varrho(x) = \varrho_x$ for $x \in X$ and $E_\varrho(x) = \emptyset$, otherwise.

[4] For a binary relation $R$, $R^{+}$ denotes the transitive closure of $R$ and $R^*$ denotes its reflexive and transitive closure.

And consider the transformation unit $rho(X)$ that uses $X$ and gets $e$ as control condition. Then we have $SEM(rho(X)) = \widehat{E_\varrho}(L(e))$ where $L(e)$ is the language described by $e$.

## 4.2 Operations with Conversion

Under the assumption that all rules of a transformation unit are invertible and that there is a suitable control condition, the conversion and the symmetric closure of the binary relation on graphs induced by a transformation unit can also be modelled. Moreover, if one puts together the reflexive, symmetric and transitive closures, one gets a transformation unit specifying the equivalence induced by the semantic relation of a given transformation unit.

**Observation**
Let $\mathcal{A} = (\mathcal{G}, \mathcal{R}, \Longrightarrow, \mathcal{E}, \mathcal{C})$ be a graph grammar approach in which, for each $r \in \mathcal{R}$, there is an $r^{-1} \in \mathcal{R}$ such that $(\underset{r}{\Longrightarrow})^{-1} = \underset{r^{-1}}{\Longrightarrow}$ [5], and let $t \in \mathcal{T}_\mathcal{A}$. Then one can construct transformation units $conv(t)$, $sym(t)$ and $equiv(t) \in \mathcal{T}_\mathcal{A}$ such that $SEM(conv(t)) = SEM(t)^{-1}$, $SEM(sym(t)) = SEM(t) \cup SEM(t)^{-1}$, and $SEM(equiv(t)) = equiv(SEM(t))$ [5], provided that there is a control condition $C_{conv(t)} \in \mathcal{B}(\mathcal{C})$ with $SEM_{E(conv(t))}(C_{conv(t)}) = (SEM_{E(t)}(C_t))^{-1}$.

**Proof** (by induction on the recursion depth of $t$). Let $I_{conv(t)} = T_t$, $U_{conv(t)} = \{conv(t') \mid t' \in U_t\}$, $R_{conv(t)} = \{r^{-1} \mid r \in R_t\}$, $T_{conv(t)} = I_t$, and assume that $SEM_{E(conv(t))}(C_{conv(t)}) = (SEM_{E(t)}(C_t))^{-1}$ for some $C_{conv(t)} \in \mathcal{B}(\mathcal{C})$. Then by induction on the length of derivations, we get $\overset{*}{\underset{R_{conv(t)}}{\Longrightarrow}} = (\overset{*}{\underset{R_t}{\Longrightarrow}})^{-1}$. Hence, by definition of $conv(t)$, it follows that $SEM(conv(t)) = SEM(t)^{-1}$ if $U_t = \{\}$. Assume that the statement holds for all $t' \in U_t$. Then, for $G_0, \ldots, G_n \in \mathcal{G}$ we have $G_0, \ldots, G_n$ is an interleaving sequence in $conv(t)$ iff, for $i = 1, \ldots, n$, $G_{i-1} \overset{*}{\underset{R_{conv(t)}}{\Longrightarrow}} G_i$ or $(G_{i-1}, G_i) \in SEM(conv(t'))$ for some $t' \in U_t$ iff, for $i = 1, \ldots, n$, $G_i \overset{*}{\underset{R_t}{\Longrightarrow}} G_{i-1}$ or $(G_i, G_{i-1}) \in SEM(t')$, for some $t' \in U_t$ iff $G_n, \ldots, G_0$ is an interleaving sequence in $t$. By definition of $conv(t)$, it follows that $SEM(conv(t)) = SEM(t)^{-1}$.

Moreover, define $sym(t)$ and $equiv(t)$ as follows.

| $sym(t)$ | | $equiv(t)$ | |
|---|---|---|---|
| uses: | $union(conv(t), t)$ | uses: | $trans(refl(sym(t)))$ |
| conds: | $union(conv(t), t)$ | conds: | $trans(refl(sym(t)))$ |

Because of the correctness of $conv(t)$ and the observation in 4.1, we get $SEM(sym(t)) = SEM(t) \cup SEM(t)^{-1}$ and $SEM(equiv(t)) = equiv(SEM(t))$. □

**Remark**
If $C_t$ contains no negations and only context-free grammars as elementary

[5] For a binary relation $R$, $R^{-1}$ denotes the conversion of $R$, i.e. $R^{-1} = \{(G, G') \mid (G', G) \in R\}$, and $equiv(R)$ denotes the equivalence closure of $R$.

control conditions (in the sense of 2.3.1), it can be shown that $C_{conv(t)}$ can be constructed from $C_t$ such that $SEM_{E(conv(t))}(C_{conv(t)}) = (SEM_{E(t)}(C_t))^{-1}$ holds.

## 5 Normal Forms of Transformation Units

In this section, we present two unary operations on transformation units each of which constructs a normal form without changing the interleaving semantics. The presented normal forms give some information of how the different components of a transformation unit are related to each other. It should be noticed that the result in 2.7 is also of this type, because it presents a way to get rid of the initial and terminal graph specifications as extra components.

### 5.1 Encapsulating Rules

Let $\mathcal{A} = (\mathcal{G}, \mathcal{R}, \Longrightarrow, \mathcal{E}, \mathcal{C})$ be a graph transformation approach. Then, for each rule $r \in \mathcal{R}$, consider the transformation unit *encapsulate*$(r)$ with $r$ as rule and regular control condition. It is easy to see that $SEM(encapsulate(r)) = \underset{r}{\Longrightarrow}$.

Based on this fact, the application of a rule $r \in \mathcal{R}$ corresponds to the call of *encapsulate*$(r)$. Hence, for each $t \in \mathcal{T}_{\mathcal{A}}$, one can construct a new transformation unit $t' \in \mathcal{T}_{\mathcal{A}}$ such that each rule of $R_t$ is encapsulated in a used transformation unit of $t'$, and $t'$ has the same interleaving semantics as $t$ provided that there is a suitable control condition.

To formulate this observation we need the set $FLAT(t)$ containing all transformation units occurring in the import structure of $t$, i.e.

$$FLAT(t) = U_t \cup (\bigcup_{t' \in U_t} FLAT(t'))$$

**Observation**
Let $\mathcal{A} = (\mathcal{G}, \mathcal{R}, \Longrightarrow, \mathcal{E}, \mathcal{C})$ be a graph transformation approach and let $t \in \mathcal{T}_{\mathcal{A}}$. Then one can construct a transformation unit $disperse(t) \in \mathcal{T}_{\mathcal{A}}$ with $R_{disperse(t)} = \{\}$, and, for $t' \in FLAT(disperse(t))$, either $R_{t'} = \{\}$ or $t' = encapsulate(r)$ for some $r \in R_t$ such that $SEM(disperse(t)) = SEM(t)$, provided that there is a control condition $C_{disperse(t)} \in \mathcal{B}(\mathcal{C})$ with $SEM_{E(disperse(t))}(C_{disperse(t)}) = SEM_{E(t)}(C_t)$.

**Proof.** Define $I_{disperse(t)} = I_t$, $U_{disperse(t)} = \{encapsulate(r) \mid r \in R_t\} \cup \{disperse(t') \mid t' \in U_t\}$, $R_{disperse(t)} = \{\}$, and $T_{disperse(t)} = T_t$.

Then it can be shown by induction on the recursion depth of $t$ that for all $G, G' \in \mathcal{G}$, there is an interleaving sequence in $t$ from $G$ to $G'$ iff there is an interleaving sequence in $disperse(t)$ from $G$ to $G'$. Hence, $SEM(t) = SEM(disperse(t))$ if $SEM_{E(disperse(t))}(C_{disperse(t)}) = SEM_{E(t)}(C_t)$ for some $C_{disperse(t)} \in \mathcal{B}(\mathcal{C})$. □

**Remark**
If $C_t$ contains only grammars of type 0 as elementary control conditions, it can

be shown that $C_{disperse(t)}$ can be constructed from $C_t$ such that the condition $SEM_{E(disperse(t))}(C_{disperse(t)}) = SEM_{E(t)}(C_t)$ holds.

### 5.2 Flattening Transformation Units

Transformation units can be flattened meaning that if the control condition behaves properly one can get rid of the import structure by putting together all the occurring rules.

**Observation**
Let $\mathcal{A} = (\mathcal{G}, \mathcal{R}, \Longrightarrow, \mathcal{E}, \mathcal{C})$ be a graph transformation approach and let $t \in \mathcal{T}_\mathcal{A}$ such that, for each $(G, G') \in SEM_{E(t)}(C_t)$, there is an interleaving sequence in $t$ from $G$ to $G'$. Then one can construct a transformation unit $\mathit{flatten}(t) \in \mathcal{T}_\mathcal{A}$ with $U_{\mathit{flatten}(t)} = \{\}$, such that $SEM(\mathit{flatten}(t)) = SEM(t)$, provided that there is a control condition $C_{\mathit{flatten}(t)} \in \mathcal{B}(\mathcal{C})$ with $SEM_{E(\mathit{flatten}(t))}(C_{\mathit{flatten}(t)}) = SEM_{E(t)}(C_t)$.
**Proof.** Define $I_{\mathit{flatten}(t)} = I_t$, $U_{\mathit{flatten}(t)} = \{\}$, $R_{\mathit{flatten}(t)} = R_t \cup (\bigcup_{t' \in FLAT(t)} R_{t'})$, $T_{\mathit{flatten}(t)} = T_t$, and assume that $SEM_{E(\mathit{flatten}(t))}(C_{\mathit{flatten}(t)}) = SEM_{E(t)}(C_t)$ for some $C_{\mathit{flatten}(t)} \in \mathcal{B}(\mathcal{C})$.

By assumption, for all $(G, G') \in SEM_{E(t)}(C_t)$, there is an interleaving sequence in $t$ from $G$ to $G'$. Moreover, $SEM(I_{\mathit{flatten}(t)}) \times SEM(T_{flatten(t)}) = SEM(I_t) \times SEM(T_t)$. By induction on the recursion depth of $t$, it can be shown that for each interleaving sequence in $t$ from $G$ to $G'$, $G \underset{R_{\mathit{flatten}(t)}}{\overset{*}{\Longrightarrow}} G'$. Altogether, we get $SEM(\mathit{flatten}(t)) = SEM(t)$. □

**Remark**
If $C_t$ contains only context-free grammars as elementary control conditions and if for all $t' \in FLAT(t)$, $SEM(I_{t'}) = SEM(T_{t'}) = \mathcal{G}$, it can be shown that $C_{\mathit{flatten}(t)}$ can be constructed from $C_t$ such that $SEM_{E(\mathit{flatten}(t))}(C_{\mathit{flatten}(t)}) = SEM_{E(t)}(C_t)$. Note that in this case, $(G, G') \in SEM_{E(t)}(C_t)$ implies that there is an interleaving sequence in $t$ from $G$ to $G'$ such that all assumptions of the observation above are fulfilled.

## 6 Conclusion

In this paper, the notion of a transformation unit together with its interleaving semantics has been introduced, illustrated and studied with respect to operations on transformation units and some normal forms. Transformation units provide an approach-independent structuring method for building up large graph transfomation systems from small ones. The very first results indicate that transformation units may also be helpful tools for proving correctness of graph transformation systems with respect to given binary relations on graphs.

As mentioned in the introduction, transformation units are intended to be one of the basic concepts of the new graph and rule centered language GRACE which is under development by the BrAaBeLei-initiative (consisting of researchers from Bremen, Aachen, Berlin, and Leiden). The adequate use of transformation units in GRACE (and outside) requires further investigations and considerations including the following points and questions.

- Case-studies may help to gather experience with handling large sets of rules.
- Each construction in section 4 or 5 builds transformation units from transformation units where the shape of all resulting ones depends only on the imported transformation units. Hence, the construction can be described syntactically by a single transformation unit if one allows identifiers as formal parameters instead of imported transformation units. A particular result of the construction is then obtained by replacing formal parameters by actual transformation units. It may be convenient to introduce such a concept of parameterized transformation units explicitly.
- The recursion depth of transformation units and the lengths of interleaving sequences provide induction principles. Under which assumptions and how can these be turned into proof rules and a proof theory that may lead to a proof system for GRACE?
- A transformation unit describes a single binary relation on graphs by using other binary relations possibly. This excludes $n$-ary relations for $n \neq 2$ and sets or families of relations as results. Hence, one may wonder which further concepts of modularization should be investigated and how they may coexist with transformation units.
- And, finally, one should study how the notion of transformation units is related to the few other structuring principles for graph rewriting systems encountered in the literature like, for example, the module concepts proposed by Ehrig and Engels (see [EE93]), by Taentzer and Schürr (see [TS95]), or the notion of a transaction introduced by Schürr and Zündorf in the framework of PROGRES (see [SZ91]).

**Acknowledgement.** We are grateful to the two anonymous referees for their hints on possible improvements of the paper, to Frank Drewes for his suggestions concerning the environment concept in the semantics of control conditions, and to our friends in the BrAaBeLei group – in particular, Ralf Betschko, Hartmut Ehrig, Gregor Engels, Annegret Habel, Berthold Hoffmann, Jürgen Müller, Detlef Plump, Andy Schürr, and Gabriele Taentzer – for fruitful discussions on the subject of this paper.

## References

[Bun79] H. Bunke. Programmed graph grammars. In Claus et al. [CER79], 155–166.

[CER79] V. Claus, H. Ehrig, G. Rozenberg, eds. Graph Grammars and Their Application to Computer Science and Biology, Lecture Notes in Computer Science 73, 1979.

[Cou90] B. Courcelle. Graph rewriting: An algebraic and logical approach. In J. van Leeuwen, ed., Handbook of Theoretical Computer Science, volume Vol. B., 193–242. Elsevier, Amsterdam, 1990.

[DP89] J. Dassow, G. Păun. Regulated Rewriting in Formal Language Theory, volume 18 of EATCS Monographs on Theoretical Computer Science. Springer-Verlag, 1989.

[Ehr79] H. Ehrig. Introduction to the algebraic theory of graph grammars. In Claus et al. [CER79], 1–69.

[EE93] H. Ehrig, G. Engels. Towards a module concept for graph transformation systems. Technical Report 93-34, Leiden, 1993.

[EH86] H. Ehrig, A. Habel. Graph grammars with application conditions. In G. Rozenberg, A. Salomaa, eds., The Book of L, 87–100. Springer-Verlag, Berlin, 1986.

[EKR91] H. Ehrig, H.-J. Kreowski, G. Rozenberg, eds. Graph Grammars and Their Application to Computer Science, Lecture Notes in Computer Science 532, 1991.

[ENR83] H. Ehrig, M. Nagl, G. Rozenberg, eds. Graph-Grammars and Their Application to Computer Science, Lecture Notes in Computer Science 153, 1983.

[ENRR87] H. Ehrig, M. Nagl, G. Rozenberg, A. Rosenfeld, eds. Graph-Grammars and Their Application to Computer Science, Lecture Notes in Computer Science 291, 1987.

[GKS91] J.R.W. Glauert, J.R. Kennaway, M.R. Sleep. Dactl: An experimental graph rewriting language. In Ehrig et al. [EKR91], 378–395.

[Hab92] A. Habel. Hyperedge replacement: Grammars and languages. Lecture Notes in Computer Science 643, 1992.

[HHT95] A. Habel, R. Heckel, G. Taentzer. Graph grammars with negative application conditions. Fundamenta Informaticae, 1995. To appear.

[Him91] M. Himsolt. Graph-Ed: An interactive tool for developing graph grammars. In Ehrig et al. [EKR91], 61–65.

[JR80] D. Janssens, G. Rozenberg. On the structure of node-label-controlled graph languages. Information Sciences 20, 191–216, 1980.

[Kre93] H.-J. Kreowski. Five facets of hyperedge replacement beyond context-freeness. In Z. Ésik, ed., Fundamentals of Computation Theory, Lecture Notes in Computer Science 710, 69–86, 1993.

[Kre95] H.-J. Kreowski. Graph grammars for software specification and programming: An eulogy in praise of GRACE. In Rosselló and Valiente [RV95], 55–61.

[KR90] H.-J. Kreowski, G. Rozenberg. On structured graph grammars, I and II. Information Sciences 52, 185–210 and 221–246, 1990.

[Kus95] S. Kuske. Semantic aspects of the graph and rule centered language GRACE. In Rosselló and Valiente [RV95], 63–69.

[Len90] T. Lengauer. VLSI theory. In J. van Leeuwen, ed., Handbook of Theoretical Computer Science, volume A. Elsevier Science Publishers B.V., 1990.

[LM93] I. Litovsky, Y. Métivier. Computing with graph rewriting systems with priorities. Theoretical Computer Science 115, 191–224, 1993.

[Löw93] M. Löwe. Algebraic approach to single-pushout graph transformation. Theoretical Computer Science 109, 181–224, 1993.

[LB93] M. Löwe, M. Beyer. AGG — an implementation of algebraic graph rewriting. In C. Kirchner, ed., Rewriting Techniques and Applications, Lecture Notes in Computer Science 690, 451–456, 1993.

[MW91] A. Maggiolo-Schettini, J. Winkowski. Programmed derivations of relational structures. In Ehrig et al. [EKR91], 582–598.

[Nag79] M. Nagl. Graph-Grammatiken: Theorie, Anwendungen, Implementierungen. Vieweg, Braunschweig, 1979.

[RV95] F. Rosselló, G. Valiente, eds. Proceedings Colloquium on Graph Transformation and its Application in Computer Science, Technical Report UIB-DMI-B-19. University of the Balearic Islands, 1995.

[SE93] H.J. Schneider, H. Ehrig, eds. Graph Transformations in Computer Science, Lecture Notes in Computer Science 776, 1993.

[Sch91a] A. Schürr. Operationales Spezifizieren mit programmierten Graphersetzungssystemen. Deutscher Universitäts-Verlag, Wiesbaden, 1991.

[Sch91b] A. Schürr. PROGRES: A VHL-language based on graph grammars. In Ehrig et al. [EKR91], 641–659.

[Sch95] A. Schürr. Programmed graph transformations and graph transformation units in GRACE, 1995. This volume.

[SZ91] A. Schürr, A. Zündorf. Nondeterministic control structures for graph rewriting systems. In G. Schmidt, R. Berghammer, eds., Graph-Theoretic Concepts in Computer Science, Lecture Notes in Computer Science 570, 48–62, 1991.

[TB93] G. Taentzer, M. Beyer. Amalgamated graph transformation systems and their use for specifying AGG – an algebraic graph grammar system. In Schneider and Ehrig [SE93], 380–394.

[TS95] G. Taentzer, A. Schürr. DIEGO, another step towards a module concept for graph transformation systems. Electronic Notes in Theoretical Computer Science, 1995. To appear.

[Ull84] J.D. Ullman. Computational Aspects of VLSI. Computer Science Press, Rockville, MD, 1984.