Graph Transformation Units and Modules
Hans-Jörg Kreowski and Sabine Kuske

University of Bremen, Department of Computer Science
P.O.Box 33 04 40, 28334 Bremen, Germany
{kreo,kuske}@informatik.uni-bremen.de

Published in:
Hartmut Ehrig, Gregor Engels, Hans-Jörg Kreowski, Grzegorz Rozenberg, editors, Handbook of
Graph Grammars and Computing by Graph Transformation, Vol. 2: Applications, Languages and

To be used only in the SegraVis School 2006!
Chapter 15

GRAPH TRANSFORMATION UNITS AND MODULES

H.-J. KREOWSKI    S. KUSKE

Universität Bremen, Fachbereich 3
Postfach 330449, D-28334 Bremen, Germany
email: {kreo, kuske}@informatik.uni-bremen.de

The aim of this chapter is to introduce the notions of transformation units and transformation modules as 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 use other transformation units for structuring purposes. Its semantics is a binary relation between initial and terminal graphs which is given by interleaving ordinary direct derivation steps with calls of imported transformation units. Putting transformation units together yields transformation modules. A module may have an import interface consisting of formal parameter units and an export interface consisting of the units that are made available to the environment. The formal parameter units can be instantiated by exported units of other modules. The introduced framework is independent of a particular graph transformation approach and, therefore, it may enhance the usefulness of graph transformations in many contexts.

Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>15.1</td>
<td>Introduction</td>
<td>606</td>
</tr>
<tr>
<td>15.2</td>
<td>Transformation Units</td>
<td>607</td>
</tr>
<tr>
<td>15.3</td>
<td>Shortest Paths—An Example</td>
<td>616</td>
</tr>
<tr>
<td>15.4</td>
<td>Transformation Modules</td>
<td>629</td>
</tr>
<tr>
<td>15.5</td>
<td>Conclusion</td>
<td>632</td>
</tr>
<tr>
<td>References</td>
<td>633</td>
<td></td>
</tr>
</tbody>
</table>
CHAPTER 15. TRANSFORMATION UNITS AND MODULES

15.1 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, Petri nets, circuit diagrams, data flow graphs, state charts, and entity-relationship diagrams are some typical examples. On the other hand, rules are used to describe “permitted” actions on objects as, 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 volume I of the Handbook and [3,10,11,9,30,1,5] for a survey). A wide spectrum of approaches exists within this theory and some of them are implemented (see, for example, Progres [31], GraphEd [18], Dactl [13], and Agg [26]) and Chapters 13 and 14 of this volume.

With the aim of enhancing the usefulness of graph transformation, we introduce approach-independent structuring methods for building up large systems of graph transformation rules from small pieces. The methods are 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 and existing transformation units can be re-used.

To make the structuring more flexible, a cluster of transformation units can be encapsulated into a module. This allows to distinguish between main units to be exported and made available to the environment and auxiliary units that are hidden in the body. Moreover, a module can import formal parameter units that allow to leave parts of a system unspecified for later actualization.

Transformation units and modules are basic concepts of the new graph and rule centered language Grace that is being developed by researchers from Berlin, Bremen, Erlangen, München, Oldenburg, and Paderborn (see also [21, 1,32,22]). 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 often called graph rewriting.

The chapter is organized as follows. In Section 15.2 we discuss the notion of a transformation unit together with its interleaving semantics. In Section 15.3, the concepts of a transformation unit are illustrated with the sample specification of a shortest-path algorithm. Finally, we introduce the notion of transformation modules and their composition in Section 15.4. The chapter ends with some concluding remarks.
15.2 Transformation Units

The key operation in graph transformation approaches is the direct derivation being the transformation of a graph into a graph by applying a rule. In other words, each rule yields a binary relation on graphs. Hence, each set of rules specifies a binary relation on graphs by iterated rule applications. This derivation process is highly non-deterministic 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., [2,29,8,23,27,34,20,24,16,28,33], cf. also [6] 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 imported 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 transformation 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 depends on so-called environments. In this way, it can be defined without forward reference to transformation units.

Examples of graph class expressions and control conditions are given after the introduction of transformation units and their interleaving semantics. At the end of this section, we consider a certain class of control conditions which consists of languages over rules and transformation units and point out its relation to interleaving sequences.
15.2.1 Graph Transformation Approach

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

- $\mathcal{G}$ is a class of graphs,
- $\mathcal{R}$ is a class of rules,
- $\Rightarrow$ is a rule application operator yielding a binary relation $\Rightarrow_r \subseteq \mathcal{G} \times \mathcal{G}$ 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 $\text{SEM}(e) \subseteq \mathcal{G}$, and
- $\mathcal{C}$ is a class of elementary control conditions over some set $\text{ID}$ of identifiers such that each $c \in \mathcal{C}$ specifies a binary relation $\text{SEM}_E(c) \subseteq \mathcal{G} \times \mathcal{G}$ for each mapping $E : \text{ID} \rightarrow 2^{\mathcal{G} \times \mathcal{G}}$.\(^1\)

A pair $(G, G') \in \Rightarrow_r$, usually written as $G \Rightarrow r G'$, establishes a direct derivation from $G$ to $G'$ through $r$. For a set $P \subseteq \mathcal{R}$ the union of all relations $\Rightarrow_r$, $(r \in P)$ is denoted by $\Rightarrow_P$ and its reflexive and transitive closure by $\Rightarrow^*_P$. A pair $(G, G') \in \Rightarrow^*_P$, usually written as $G \Rightarrow_P^* G'$, is called a derivation from $G$ to $G'$ over $P$. A mapping $E : \text{ID} \rightarrow 2^{\mathcal{G} \times \mathcal{G}}$ 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 make use of the constant true. The semantic relations of elementary control conditions are easily extended to boolean expressions by

\[
\begin{align*}
\text{SEM}_E(\text{true}) &= \mathcal{G} \times \mathcal{G}, \\
\text{SEM}_E(e_1 \lor e_2) &= \text{SEM}_E(e_1) \cup \text{SEM}_E(e_2), \\
\text{SEM}_E(e_1 \land e_2) &= \text{SEM}_E(e_1) \cap \text{SEM}_E(e_2), \\
\text{SEM}_E(\lnot e) &= \mathcal{G} \times \mathcal{G} - \text{SEM}_E(e).
\end{align*}
\]

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 $\text{SEM}$. This should do no harm because it is always clear from the context which is which.

All the graph grammar and graph transformation 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. [7, 29, 19, 4, 23, 8, 15, 25]). 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

\(^1\)The power set of a set $S$ is denoted by $2^S$.\[\text{SEM}_E(\text{true}) = \mathcal{G} \times \mathcal{G},\text{SEM}_E(e_1 \lor e_2) = \text{SEM}_E(e_1) \cup \text{SEM}_E(e_2),\text{SEM}_E(e_1 \land e_2) = \text{SEM}_E(e_1) \cap \text{SEM}_E(e_2),\text{SEM}_E(\lnot e) = \mathcal{G} \times \mathcal{G} - \text{SEM}_E(e).\]

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

All the graph grammar and graph transformation 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. [7, 29, 19, 4, 23, 8, 15, 25]). 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 \(SEM(\text{all}) = G\) may provide the only graph class expression, and the class of elementary control conditions may be empty. Non-trivial choices for \(E\) and \(C\) are discussed in Subsections 15.2.4 and 15.2.5.

### 15.2.2 Transformation Units

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

Let \(\mathcal{A} = (G, \mathcal{R}, \Rightarrow, E, C)\) be a graph transformation approach. A transformation unit over \(\mathcal{A}\) is a system \(\text{trut} = (I, U, R, C, T)\) where \(I, T \in \mathcal{E}\), \(U\) is a finite set of identifiers, \(R \subseteq \mathcal{R}\) is a finite set of rules, and \(C \in \mathcal{B}(C)\). The components of \(\text{trut}\) may be denoted by \(U_{\text{trut}}, I_{\text{trut}}, R_{\text{trut}}, C_{\text{trut}},\) and \(T_{\text{trut}}\), respectively. The class of all transformation units over \(\mathcal{A}\) is denoted by \(\mathcal{T}_A\). The component \(U\) may be seen as a set of formal parameters that can be instantiated by transformation units.

To keep the technicalities simple, one may assume that only defined transformation units are imported. Hence, initially, \(U\) must be chosen as the empty set yielding unstructured transformation units without import. Such transformation units of level 0 may be used in transformation units of level 1. Iteratively, one obtains a transformation unit of level \(i + 1\) for some \(i \in \mathbb{N}\) if one imports transformation units up to level \(i\). In this way, the import structures of transformation units become acyclic. This provides a principle of hierarchical structuring. The case of an arbitrary import structure is studied in [22].

If \(I\) specifies a single graph (cf. item 1 of 15.2.4), \(U\) is empty, and \(C\) is the constant \(\text{true}\), one gets the usual notion of a graph grammar (in which approach ever) as a special case of transformation units.

### 15.2.3 Interleaving Semantics

The 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 \(\text{trut} = (I, U, R, C, T)\) be a transformation unit over the graph transformation approach \(\mathcal{A} = (G, \mathcal{R}, \Rightarrow, E, C)\). Assume that the set \(ID\) of identifiers
associated to \( C \) contains the disjoint union of \( U \) and \( R \). Let the interleaving semantics \( \text{SEM}(t) \subseteq \mathcal{G} \times \mathcal{G} \) for \( t \in U \) be already defined. Let \( E(\text{trut}) : \text{ID} \rightarrow 2^{\mathcal{G} \times \mathcal{G}} \) be defined by \( E(\text{trut})(r) = \Rightarrow_r \) for \( r \in R \), \( E(\text{trut})(t) = \text{SEM}(t) \) for \( t \in U \), and \( E(\text{trut})(\text{id}) = \{ \} \), otherwise. Then the interleaving semantics \( \text{SEM}(\text{trut}) \) of \( \text{trut} \) consists of all pairs \((G, G') \in \mathcal{G} \times \mathcal{G}\) such that

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

The sequence of graphs in point 2 is called an interleaving sequence in \( \text{trut} \) from \( G \) to \( G' \). Let \( \text{RIS}_{\text{trut}} \) denote the binary relation given by interleaving sequences, i.e. \( \text{RIS}_{\text{trut}} = (\Rightarrow \cup \bigcup_{t \in U} \text{SEM}(t))^* \). Then the interleaving semantics of \( \text{trut} \) is defined as the intersection of \( \text{RIS}_{\text{trut}} \) with \( \text{SEM}(I) \times \text{SEM}(T) \) and \( \text{SEM}_{E(\text{trut})}(C) \). Note that all three relations may be incomparable with each other. For example, \((G, G') \in \text{SEM}_{E(\text{trut})}(C)\) does not imply in general that there is an interleaving sequence in \( \text{trut} \) 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 \( \text{trut} \), only the environment of \( \text{trut} \) given by \( E(\text{trut}) \) is effective, meaning that \( C \) can restrict the semantics by specifying certain properties of the direct derivation relations of rules in \( \text{trut} \), the interleaving semantics of imported transformation units, and the interrelation of all of them. If transformation units are employed as structuring concepts in a specification language, it would be reasonable to assume that rules may be named and that only their names belong to the set of identifiers rather than the rules themselves. But the naming of rules is not needed here.

The definition of the interleaving semantics follows the recursive definition of transformation units. Hence, its well-definedness follows easily by an induction on the import structure, i.e. on the levels of transformation units.

Initially, if \( U \) is empty, an interleaving sequence is just a derivation such that one gets in this case

\[
\text{SEM}(\text{trut}) = \Rightarrow_h^* \cap (\text{SEM}(I) \times \text{SEM}(T)) \cap \text{SEM}_{E(\text{trut})}(C).
\]

In other words, interleaving semantics generalizes the ordinary semantics of sets of rules given by derivations.

If \( I \) is a single graph (specifying itself as initial graph in the sense of 15.2.4.1), the first component of the interleaving semantics of \( \text{trut} \) is insignificant. Then
all second components form a graph language that can be considered as the
language generated by the transformation unit, i.e.
\[
L(trut) = \{ G \in SEM(T) \mid (I,G) \in SEM(trut) \}.
\]
In this case, the transformation unit is called language-generating. If, furthermore, \( U \) is empty and \( C \) is true, \( trut \) is a graph grammar (cf. Subsection 15.2.2), and its generated language consists, as usual, of all terminal graphs derivable
from the initial graph, i.e.
\[
L(trut) = \{ G \in SEM(T) \mid I \Rightarrow^*_R G \}.
\]
In this sense, the interleaving semantics covers the usual notion of graph languages
generated by graph grammars.
The interleaving semantics of a transformation unit is defined for any choice
of the imported transformation units. If one fixes the import, the interleaving
semantics is a binary relation on graphs. But if one does not fix the import,
the interleaving semantics can be considered as an operator yielding a binary
relation on graphs for each choice of binary relations on graphs for the import
parameters.

15.2.4 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 trans-
formation 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 \( G \) rep-
resents itself, i.e. \( SEM(G) = \{ G \} \). The axiom of a graph grammar is a
typical example of this type.

2. A graph \( G \) is \textit{reduced} with respect to a set of rules \( P \subseteq R \) if there is
no \( G' \in G \) with \( G \Rightarrow^r G' \) 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 \( G \) 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 \( SEM(T) = G_T \)
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, hypergraphs or undirected graphs are suitable candidates (see e.g. [4]).

5. Graph schemata, as used in the graph transformation approach Progres are graph class expressions that allow to specify generic graph classes (see Chapter 13 for more details).

6. A language-generating transformation unit \( \text{trut} \) as introduced in Subsection 15.2.3 can be used as a graph class expression with \( \text{SEM} (\text{trut}) = L (\text{trut}) \).

15.2.5 Control Conditions

A control condition is meant to restrict the derivation process. A typical example is to allow only interleaving sequences where the sequences of applied rules and called transformation units belong to a particular control language. Therefore, a regular expression over the set of identifiers can be considered as a control condition because it specifies a language. In general, every description of a binary relation on graphs may be used as a control condition. Here, we give some examples.

1. Let \( E : \text{ID} \to 2^{\mathcal{G} \times \mathcal{G}} \) be an environment. Then \( E \) can be extended to the set of languages over \( \text{ID} \), i.e. the power set of the set of strings over \( \text{ID} \) in a natural, straight-forward way. \( \hat{E} : 2^{\text{ID}^*} \to 2^{\mathcal{G} \times \mathcal{G}} \) is defined by \( \hat{E}(L) = \bigcup_{w \in L} \hat{E}(w) \) for \( L \subseteq \text{ID}^* \) where \( \hat{E} : \text{ID}^* \to 2^{\mathcal{G} \times \mathcal{G}} \) is recursively given by \( \hat{E}(\lambda) = \Delta \mathcal{G} \), and \( \hat{E}(xv) = E(x) \circ \hat{E}(v) \) for \( x \in \text{ID} \) and \( v \in \text{ID}^* \). Hence, \( \mathcal{L} \) can be used as control condition with \( \text{SEM}_E (\mathcal{L}) = \hat{E} (\mathcal{L}) \) for all \( E : \text{ID} \to 2^{\mathcal{G} \times \mathcal{G}} \). In this case, the class of elementary control conditions is \( 2^{\text{ID}^*} \). We refer to conditions in this class as control conditions of language type.

2. As a consequence of point 1, every grammar, automaton or expression \( x \) which specifies a language \( L(x) \) over \( \text{ID} \) can serve as a control condition with \( \text{SEM}_E (x) = \text{SEM}_E (L(x)) = \hat{E} (L(x)) \) for all environments \( E \).

3. In particular, the class of regular expressions over \( \text{ID} \) can be used for this purpose. For explicit use below, \( \text{REG} (\text{ID}) \) is recursively given by \( \emptyset, \epsilon \in \text{REG} (\text{ID}), \text{ID} \subseteq \text{REG} (\text{ID}), \) and \( (e_1; e_2), (e_1 \mid e_2), (e^* \in \text{REG} (\text{ID}) \) if \( e, e_1, e_2 \in \text{REG} (\text{ID}) \). To save parentheses we assume that \( \ast \) has a stronger

\( \Delta \mathcal{G} \) denotes the identity relation on \( \mathcal{G} \). Given \( \rho, \rho' \subseteq \mathcal{G} \times \mathcal{G} \), the sequential composition of \( \rho \) and \( \rho' \) is defined as usual by \( \rho \circ \rho' = \{(G, G') \mid (G, G') \in \rho \text{ and } (G', G'') \in \rho' \text{ for some } G' \in \mathcal{G} \} \).
binding than ; which in turn has a stronger binding than \(|\). The language \( L(e) \) specified by some regular expression \( e \) is defined as \( L(\emptyset) = \{\} \), \( L(\varepsilon) = \{\lambda\} \), \( L(id) = \{id\} \) for all \( id \in ID \), \( L(e_1 ; e_2) = L(e_1) \cdot L(e_2) \), \( L(e_1 | e_2) = L(e_1) \cup L(e_2) \) and \( L(e^*) = L(e)^* \).

4. A pair \((G, G')\) of graphs is reduced with respect to a control condition \( c \in C \) and an environment \( E \) if there is no graph \( G'' \) with \((G', G'') \in SEM_E(c)\). Thus, \( c! \) defines a control condition where \( SEM_E(c!) \) is the set of all pairs in \( SEM_E(c) \) which are reduced with respect to \( c \) and \( E \).

5. A special case of such a control condition corresponds to the notion of reduced graphs and is given by a set of rules \( R \). \( R! \) means that the rules of \( R \) must be applied as long as possible.

6. Each transformation unit \( brut \) can serve as a control condition because semantically it specifies a binary relation on graphs. For each environment \( E \), the semantics of the control condition \( brut \) is given by the semantics of \( brut \), i.e. by all pairs \((G, G')\) of graphs such that \( G \) can be transformed into \( G' \) with the transformation unit \( brut \).

7. Each pair \((e_1, e_2) \in E \times E\) 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. \( SEM((e_1, e_2)) = SEM((e_1, e_2)) \) for all environments \( E \). In particular, let \( brut = (I, U, R, C, T) \) be a transformation unit. Then the pair \((I, T)\) forms a control condition.

8. For readers familiar with the graph transformation language \( PROGRES \), it shall be mentioned that the deterministic and non-deterministic control structures of \( PROGRES \) serve as control conditions. They allow to define imperative commands over control conditions.

9. Another type of control conditions are priorities among the rules of a transformation unit. See Litovski and Métivier \cite{24} for a particular approach of this kind.

As the following observation shows, the semantic relations given by regular expressions as control conditions can be constructed easily according to the recursive structure of regular expressions without reference to the languages generated by the expressions.

---

3While \( \emptyset \) denotes the empty set \( \{\} \), the expression \( \varepsilon \) denotes the regular set \( \{\lambda\} \). We prefer a direct reference to \( \{\lambda\} \) rather than to use \( \emptyset^* \).

4Given \( L, L' \subseteq \mathcal{E}^* \), the concatenation of \( L \) and \( L' \) is defined as usual by \( L \cdot L' = \{wu' \mid w \in L, u' \in L'\} \), and the Kleene closure of \( L \) is defined as usual by \( L^* = \bigcup_{i=0}^{\infty} L^i \) where \( L^0 = \{\lambda\} \) and \( L^{i+1} = L \cdot L^i \).
CHAPTER 15. TRANSFORMATION UNITS AND MODULES

Observation 15.2.1
For all environments $E$, all $id \in ID$ and all $e, e_1, e_2 \in REG(ID)$ the following holds:

1. $SEM_E(\emptyset) = \{\}.$
2. $SEM_E(e) = \Delta \bar{G}.$
3. $SEM_E(id) = E(id).$
4. $SEM_E(e_1 \cdot e_2) = SEM_E(e_1) \circ SEM_E(e_2).$
5. $SEM_E(e_1 | e_2) = SEM_E(e_1) \cup SEM_E(e_2).$
6. $SEM_E(e^*) = SEM_E(e)^*.$

Proof

1. $SEM_E(\emptyset) =_{def} \hat{E}(\emptyset) =_{def} \{\}.$
2. $SEM_E(e) =_{def} \hat{E}(\{\lambda\}) =_{def} \bar{E}(\lambda) =_{def} \Delta \bar{G}.$
3. $SEM_E(id) =_{def} \hat{E}(\{id\}) =_{def} \bar{E}(id) =_{def} E(id).$
4. To show this, we use the following statement which is shown by induction on the length of $w_1$. Let $w_1, w_2 \in ID^*$; then $\bar{E}(w_1w_2) = \bar{E}(w_1) \circ \bar{E}(w_2).$

$\bar{E}(\lambda w_2) =_{def} \bar{E}(w_2) = \Delta \bar{G} \circ \bar{E}(w_2) =_{def} \bar{E}(\lambda) \circ \bar{E}(w_2).$ Assume that the statement holds for $w_1 \in ID^*$, and consider some $a \in ID$. Then $\bar{E}(aw_1w_2) =_{def} E(a) \circ \bar{E}(w_1w_2) =_{ind} E(a) \circ (\bar{E}(w_1) \circ \bar{E}(w_2)) = (E(a) \circ \bar{E}(w_1)) \circ \bar{E}(w_2) =_{def} E(aw_1) \circ \bar{E}(w_2).$ Now we get

$SEM_E(e_1 \cdot e_2) =_{def} \hat{E}(L(e_1) \cdot L(e_2)) =_{def} \bigcup_{w_1 \in L(e_1), w_2 \in L(e_2)} \bar{E}(w_1w_2)$

$= \bigcup_{w_1 \in L(e_1), w_2 \in L(e_2)} \bar{E}(w_1) \circ \bar{E}(w_2)$

$= \bigcup_{w_1 \in L(e_1)} \bar{E}(w_1) \circ \bigcup_{w_2 \in L(e_2)} \bar{E}(w_2) =_{def} \hat{E}(L(e_1)) \circ \bar{E}(L(e_2))$

$=_{def} SEM_E(e_1) \circ SEM_E(e_2).$

5. $SEM_E(e_1 | e_2) =_{def} \hat{E}(L(e_1) \cup L(e_2)) =_{def} \bigcup_{w \in L(e_1) \cup L(e_2)} \bar{E}(w)$

$= \bigcup_{w \in L(e_1)} \bar{E}(w_1) \cup \bigcup_{w \in L(e_2)} \bar{E}(w_2) =_{def} \hat{E}(L(e_1)) \cup \bar{E}(L(e_2))$

$=_{def} SEM_E(e_1) \cup SEM_E(e_2).$

6. To show point 6, we first prove by induction on $i$ that for $i \geq 0,$

$\hat{E}(L(e)^i) =_{def} SEM_E(e)^i.$ If $i = 0$ we have

$\hat{E}(L(e)^0) =_{def} \hat{E}(\{\lambda\}) =_{def} \Delta \bar{G} =_{def} SEM_E(e)^0.$

---

6 For $\rho \subseteq G \times G$, $\rho^*$ denotes the reflexive and transitive closure of $\rho$ that is $\rho^* = \bigcup_{i=0}^{\infty} \rho^i$ where $\rho^0 = \Delta G$ and $\rho^{i+1} = \rho \circ \rho^i$.  
7 $=_{ind}$ stands for equal by induction hypothesis.
15.2. Transformation Units

Moreover,

\[
\hat{E}(L(e)^{i+1}) =_{d.f} \hat{E}(L(e) \cdot L(e)^i) =_{d.f} \hat{E}(L(e)) \circ \hat{E}(L(e)^i) =_{ind} SEM_E(e) \circ SEM_E(e)^i =_{d.f} SEM_E(e)^{i+1}.
\]

Hence,

\[
SEM_E(e^*) =_{d.f} \hat{E}(L(e)^*) =_{d.f} \hat{E}(\bigcup_{i=0}^{\infty} L(e)^i) = \bigcup_{i=0}^{\infty} \hat{E}(L(e)^i) =_{d.f} SEM_E(e)^*.
\]

15.2.6 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 15.2.5.1.

Let \( \text{trut} = (I, U, R, C, T) \) be a transformation unit over some graph transformation approach \( A = (G, R, \Rightarrow, E, C) \). Assume that \( U \) and \( R \) are disjoint subsets of the set \( ID \) associated to \( C \). Then \( x_1 \cdots x_n \in (U \cup R)^* \) \((x_i \in U \cup R)\) 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 \( i = 1, \ldots, n, G_{i-1} \Rightarrow x_i G_i \) 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 15.2.5.1, controls the order in which rules are applied and imported transformation units are actually used.

Observation 15.2.2

Let \( \mathcal{C} = 2^{ID^*} \) be the class of control conditions of language type, and let \( \text{trut} = (I, U, R, L, T) \) with \( L \subseteq (U \cup R)^* \subseteq ID^* \). Then for all \( G, G' \in \mathcal{G} \), the following statements are equivalent.

1. \((G, G') \in SEM(\text{trut})\).
2. \((G, G') \in SEM_E(\text{trut})(L) \cap SEM(I) \times SEM(T)\).
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 \((G, G') \in SEM(\text{trut})\). 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 \((G,G') \in SEM_{E_{(trut)}}(L) = E(\overline{trut})(L) \) iff \((G,G') \in E(\overline{trut})(w)\) for some \( w \in L \). We show now by induction on the structure of \( w \) that \((G,G') \in E(\overline{trut})(w) \) iff \( w \) is an application sequence of \((G,G')\).

If \( w = \lambda \), we get \((G,G') \in E(\overline{trut})(\lambda) \) iff \((G,G') \in \Delta G \) iff \( G = G' \) iff \( \lambda \) is an application sequence of \((G,G')\).

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

And consider \( w = xv \) with \( x \in U \cup R \). Then \((G,G') \in E(\overline{trut})(xv) = E(trut)(x) \circ E(\overline{trut})(v)\), means that there is some \( G \in \mathcal{G} \) with \((G,G') \in E(\overline{trut})(x)\) and \((G',G) \in E(\overline{trut})(v)\). The latter implies by induction that \( v \) is an application sequence of \((G',G)\) such that there is an interleaving sequence \( G_0, \ldots, G_n \) with \( G = G_0 \) and \( G' = G_n \). The former means \( G \Rightarrow_x G' \) if \( x \in R \) and \((G,G) \in SEM(x)\) 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 \( G = G_0 \) and \( G' = G_n \) and, in particular, \( G_0 \Rightarrow_x G_1 \) if \( x \in R \) and \((G_0,G_1) \in SEM(x)\) if \( x \in U \) such that \((G,G_1) \in E(\overline{trut})(x)\) 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 \((G_1,G_n) \in E(\overline{trut})(v)\). The composition yields \((G,G') \in E(trut)(x) \circ E(\overline{trut})(v) = E(\overline{trut})(xv)\). 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 \((G,G') \in SEM(I) \times SEM(T)\). Then by definition there is an interleaving sequence in \( trut \) from \( G \) to \( G' \) with \((G,G') \in SEM(I) \times SEM(T)\), and by the claim, \((G,G') \in SEM_{E_{(trut)}}(L)\). Hence, \((G,G') \in SEM(trut).\) This completes the proof.

15.3 Shortest Paths An Example

In this section, we specify the shortest-path algorithm of Floyd (cf. [12]) as a graph transformation unit to illustrate the usefulness of the concept. The algorithm is informally described in the next subsection. Its specification in form of a transformation unit is presented in Subsection 15.3.3 after the underlying
15.3. **SHORTEST PATHS AN EXAMPLE**

Graph transformation approach is introduced. The rest of the section concerns the correctness and complexity of the specified algorithm. In Subsection 15.3.4, we discuss the graph theoretical background of the algorithm more formally to be able to prove correctness of the specification in Subsection 15.3.5 (together with 15.3.7). Finally, the issue of complexity is studied in Subsection 15.3.6.

15.3.1 *Informal Description of the Algorithm*

Given a directed graph \( G \) and a pair \( v \) and \( v' \) of nodes, the algorithm computes the distance of a shortest path from \( v \) to \( v' \) in \( G \). The main idea is to compute the distance of a shortest path from \( v \) to \( v' \) that avoids certain nodes in intermediate steps from the distances of shortest paths that avoid more nodes.

In more detail, the shortest paths from \( v \) to \( v' \) that avoid all nodes in intermediate steps are the edges from \( v \) to \( v' \) with minimum distance. Starting with these edges and distances, one may admit more and more nodes as intermediate ones. Whenever a new intermediate node \( \tau \) is admitted, one adds up the distance of the edge from \( v \) to \( \tau \) and the distance of the edge from \( \tau \) to \( v' \) (if there are any) and labels a new edge from \( v \) to \( v' \) with this sum. If there is already an edge, the one with the smaller distance is kept. The algorithm terminates after all nodes are admitted. Then an edge from \( v \) to \( v' \) is labelled with the distance of a shortest path from \( v \) to \( v' \) in the original graph, and there is no edge from \( v \) to \( v' \) if originally there is no path from \( v \) to \( v' \). A formal treatment of the shortest-path algorithm is given in Subsection 15.3.4.

15.3.2 *Graph Transformation Approach*

For the purposes of this illustration, a particular graph transformation approach is tailored. But the example is easily adapted to most of the general graph transformation approaches one encounters in the literature.

1. The class of graphs considered consists of directed graphs of the form

   \[ G = (V, E, s, t, l, \text{dist}) \]

   where \( V \) is the set of nodes, \( E \) is the set of edges, \( s : E \rightarrow V \) and \( t : E \rightarrow V \) are mappings associating each edge \( e \in E \) with a source \( s(e) \) and a target \( t(e) \), \( l : V \rightarrow \{0, 1, 2\} \) is a node labelling and \( \text{dist} : E \rightarrow \mathbb{N} \) is an edge labelling, called **distance**. Loops are forbidden, i.e. there is no edge \( e \) with \( s(e) = t(e) \). The distance is essential, the node labelling serves only auxiliary purposes.

   Let \( G = (V, E, s, t, l, \text{dist}) \) and \( G' = (V', E', s', t', l', \text{dist}') \) be two graphs. \( G \) is a **subgraph** of \( G' \) if \( V \subseteq V' \), \( E \subseteq E' \) and \( s(e) = s'(e), t(e) = t'(e) \) and \( \text{dist}(e) = \text{dist}'(e) \) for all \( e \in E \) as well as \( l(v) = l'(v) \) for all \( v \in V \).
G and G' are isomorphic if there are bijective mappings f : V → V' and
g : E → E' with f(s(e)) = s'(g(e)), f(t(e)) = t'(g(e)) and dist(e) =
dist'(g(e)) for all e ∈ E as well as l(e) = l'(f(e)) for all e ∈ V.

2. The class of rules considered consists of pairs of graphs r = (L, R) where
L and R share the set of nodes. Moreover, the left-hand side L of r may
be equipped with an extra context edge.
Rules are presented in the form L → R where an extra context edge in
L is dashed. Nodes of L and R are drawn in the same fill style if they are
equal. A node label is omitted if it is parametric, i.e. if it can be chosen
arbitrarily before applying the rule.

3. A rule r = (L, R) is applied to a graph G directly deriving G' in four
steps: (1) look for a subgraph L_0 of G isomorphic to L, (2) remove the
edges of L_0 from G, (3) add the edges of R (by using the isomorphism
between L and L_0 to place the edges), and (4) change the node labels
where they are different in L and R (again by using the isomorphism). If
the left-hand side has got an extra context edge, this serves as a negative
context condition meaning that the direct derivation is only admitted
if G does not contain such an edge between the respective nodes. This
altogether is denoted by G ⇒_r G'.

4. The only two used graph class expressions are the restrictions of the
node label alphabet to {0} and {2} respectively. If the two expressions
are denoted by node label i for i ∈ {0, 2}, SEM(node label i) contains all
graphs the nodes of which are constantly labelled with i.

5. The used control conditions are regular expressions and as long as pos-
sible where the latter applies the rules of a transformation unit as long
as possible (cf. 15.2.5.5).

In the following, transformation units are presented by indicating the compon-
ents with respective keywords. Trivial components (i.e. no import, no rules,
the graph class expression all, and the control condition true) are omitted.

15.3.3 Specification of the Algorithm

Floyd's algorithm is specified in terms of transformation units following the
informal description of the algorithm. To be able to distinguish between for-
bidden nodes, a just admitted node and formerly admitted nodes, the node
labels 0, 1 and 2 resp. are used. Initially, all nodes carry the 0-label.
The main transformation unit shortest-path uses the transformation units min-
umum, sum, change(0, 1) and change(1, 2) in a certain order which is given by
a regular control expression. The transformation unit \textit{minimum} takes care of parallel edges. Its control condition \textit{as long as possible} makes sure that all parallel edges are removed. The transformation unit \textit{sum} sums up successive edges if their intermediate node is 1-labeled. Here the control condition \textit{as long as possible} makes sure that all possible summations are performed, while the negative context condition associated to the rule prevents that the same summation is done twice. Finally, \textit{change}(i, j) relabels a node from $i$ to $j$. The control condition \textit{once} guarantees that the rule of the transformation unit is applied exactly once in each of its derivations. The term \textit{once} is a synonym for the rule considered as a regular expression. Note that in the transformation units \textit{minimum} and \textit{sum} the edge labels as well as most of the node labels of the rules are parametric, i.e. they can be chosen arbitrarily before each rule application.

Wherever parallel edges occur, only an edge with the minimum label is kept. This is achieved by the transformation unit \textit{minimum}.

The summations of the distances of successive edges is performed by the transformation unit \textit{sum} whenever the intermediate node is labelled with 1.

Before \textit{sum} is applied, a node with label 0 gets label 1, and afterwards this
label is changed into 2 by using the transformation unit \( \text{change}(i, j) \) for any two labels \( i, j \).

\[
\begin{array}{c}
\text{change}(i, j) \\
\text{rules:} \\
\bullet \quad \rightarrow \quad \bullet \\
\text{conds:} \quad \text{once}
\end{array}
\]

After \( \text{minimum} \) has done its job once, the described sequence of transformation units

\( \text{change}(0, 1), \text{sum}, \text{change}(1, 2), \text{minimum} \)

can be iterated until all nodes are admitted. This yields the shortest-path algorithm.

<table>
<thead>
<tr>
<th>shorted-path</th>
</tr>
</thead>
<tbody>
<tr>
<td>initial:</td>
</tr>
<tr>
<td>uses:</td>
</tr>
<tr>
<td>conds:</td>
</tr>
<tr>
<td>terminal:</td>
</tr>
</tbody>
</table>

Figure 15.1 shows an interleaving sequence of \( \text{shortest-path} \) where the effect of the \( \text{change} \) units are not presented separately, but composed with \( \text{minimum} \). Note that the regular expression \( \text{ch}(1, 2) ; \text{min} ; \text{ch}(0, 1) \) in the picture is an abbreviation for the control condition \( \text{change}(1, 2) ; \text{minimum} ; \text{change}(0, 1) \).

15.3.4 Graphtheoretic Background

To facilitate the correctness proof for the algorithm, the graphtheoretic background is needed explicitly.

Let \((V, E, s, t)\) be an unlabelled directed graph without loops, and let \( \text{dist} : E \rightarrow \mathbb{N} \) be a distance function on the edges.

A sequence of edges \( p = e_1 \cdots e_n \) \((n \geq 1)\), is a path from \( v \) to \( v' \) with \( \text{dist}(p) = \sum_{i=1}^{n} \text{dist}(e_i) \) if \( s(e_1) = v, t(e_n) = v' \) and \( t(e_i) = s(e_{i+1}) \) for \( i = 1, \ldots, n - 1 \).
15.3. **Shortest Paths**  

An Example

Figure 15.1: Example of an interleaving sequence in shortest path
The set of the latter nodes \( \{t(e_1), \ldots, t(e_{n-1})\} \), called intermediate nodes, is denoted by \( \text{inter}(p) \). Moreover, for \( v \in V \), the empty sequence \( \lambda \) is considered as a path from \( v \) to \( v \) with \( \text{dist}(\lambda) = 0 \) and \( \text{inter}(\lambda) = \emptyset \). For \( U \subseteq V \), a path \( p \) is said to avoid \( U \) if \( \text{inter}(p) \cap U = \emptyset \).

Let \( \text{PATH}(v, v', U) \) denote the set of all paths from \( v \) to \( v' \) avoiding \( U \). If \( \text{PATH}(v, v', U) \neq \emptyset \), the minimum distance is denoted by \( \text{short}(v, v', U) \), i.e.

\[
\text{short}(v, v', U) = \min \{ \text{dist}(p) \mid p \in \text{PATH}(v, v', U) \}.
\]

A path \( p_0 \in \text{PATH}(v, v', U) \) with \( \text{dist}(p_0) = \text{short}(v, v', U) \) is called shortest path from \( v \) to \( v' \) avoiding \( U \). (Note that \( \text{short}(v, v', U) \) is undefined if \( \text{PATH}(v, v', U) = \emptyset \).)

\( \text{PATH}(v, v', \emptyset) \) consists of all paths from \( v \) to \( v' \) such that \( \text{short}(v, v', \emptyset) \) is the minimum distance of all paths from \( v \) to \( v' \). \( \text{PATH}(v, v, U) \) contains the empty path \( \lambda \) such that \( \text{short}(v, v, U) = 0 \) for all \( v \in V \) and \( U \subseteq V \). \( \text{PATH}(v, v', V) \) with \( v \neq v' \) consists of all edges \( e \) with \( s(e) = v \) and \( t(e) = v' \). If this set of edges is denoted by \( E(v, v') \), one has

\[
\text{short}(v, v', V) = \min \{ \text{dist}(e) \mid e \in E(v, v') \}
\]

for \( v \neq v' \) and \( E(v, v') \neq \emptyset \).

Let \( \text{PATH}(v, v', U) \neq \emptyset \) with \( v \neq v' \) and \( U = U \cup \{\overline{v}\} \) for some \( \overline{v} \in V - U \). Then the following hold.

1. If \( \text{PATH}(v, v', \overline{U}) \neq \emptyset \), then

\[
\text{short}(v, v', U) \leq \text{short}(v, v', \overline{U})
\]

because \( \text{PATH}(v, v', \overline{U}) \subseteq \text{PATH}(v, v', U) \).

2. If \( \text{PATH}(v, \overline{v}, \overline{U}) \neq \emptyset \neq \text{PATH}(\overline{v}, v', \overline{U}) \), then

\[
\text{short}(v, v', U) \leq \text{short}(v, \overline{v}, \overline{U}) + \text{short}(\overline{v}, v', \overline{U})
\]

because the composition of a path from \( v \) to \( \overline{v} \) avoiding \( \overline{U} \) with a path from \( \overline{v} \) to \( v' \) avoiding \( \overline{U} \) yields a path from \( v \) to \( v' \) avoiding \( U \).

3. If \( p_0 = e_1 \cdots e_n \) is a shortest path from \( v \) to \( v' \) avoiding \( U \) and \( \overline{v} \notin \text{inter}(p_0) \), then \( p_0 \in \text{PATH}(v, v', \overline{U}) \), and hence

\[
\text{short}(v, v', \overline{U}) \leq \text{dist}(p_0) = \text{short}(v, v', U).
\]
15.3. **SHORTEST PATHS  AN EXAMPLE**

(4) Otherwise \( \overline{v} \in \text{inter}(p_0) \). In this case, there are indices \( i \) and \( j \) such that \( e_1 \cdots e_i \in \text{PATH}(v, \overline{v}, U) \) and \( e_j \cdots e_n \in \text{PATH}(\overline{v}, v', U) \) where \( e_i \) is the first edge on \( p_0 \) entering \( \overline{v} \) and \( e_j \) is the last one leaving \( \overline{v} \). Hence \( i \leq j \) and

\[
\text{short}(v, \overline{v}, U) + \text{short}(\overline{v}, v', U) \leq \text{dist}(e_1 \cdots e_i) + \text{dist}(e_j \cdots e_n) \\
\leq \text{dist}(p_0) \\
= \text{short}(v, v', U).
\]

(5) Altogether, one gets either

\[
\text{short}(v, v', U) = \text{short}(v, v', U)
\]

or

\[
\text{short}(v, v', U) = \text{short}(v, \overline{v}, U) + \text{short}(\overline{v}, v', U).
\]

(6) And if both right-hand sides are defined, one has

\[
\text{short}(v, v', U) = \min \{ \text{short}(v, v', U), \text{short}(v, \overline{v}, U) + \text{short}(\overline{v}, v', U) \}.
\]

15.3.5 **Correctness**

In this section, the correctness of the transformation unit \( \text{shortest-path} \) is shown with respect to the function \( \text{short} \). To achieve this, the interleaving semantics of all involved transformation units are characterized in graph-theoretic terms. Given a graph \( G = (V, E, s, t, l, \text{dist}) \), its components may be denoted by \( V_G, E_G, s_G, t_G, l_G \) and \( \text{dist}_G \) respectively.

1. The interleaving sequences of \( \text{change}(i, j) \) are the direct derivations in which the only rule is applied to an arbitrary graph because nothing is imported and there is no restriction of initial and terminal graphs, but exactly one rule application is allowed. If the rule is applied, a node with label \( i \) gets label \( j \) obviously. Hence one gets:

\[
(G, H) \in \text{SEM}(\text{change}(i, j)) \text{ iff } V_G = V_H, E_G = E_H, s_G = s_H, t_G = t_H, \text{dist}_G = \text{dist}_H \text{ and there is some } v_0 \in V_G \text{ with } l_G(v_0) = i, l_H(v_0) = j \text{ and } l_G(v) = l_H(v) \text{ for all } v \neq v_0.
\]
2. If the rule of the transformation unit minimum is applied, two parallel edges are replaced by one edge labelled with the minimum distance of the replaced edges. The control condition makes sure that this is iterated as long as possible such that resulting graphs are simple (i.e., without parallel edges). This leads to the following characterization of the interleaving semantics of minimum.

\[(G, H) \in SEM(\text{minimum}) \iff V_G = V_H, l_G = l_H, \text{ and for all } v, v' \in V_G, E_C(v, v') \neq \emptyset \iff |E_H(v, v')| = 1 \text{ and } dist_H(e_0) = \min \{dist_G(e) \mid e \in E_G(v, v')\}, \text{ where } \{e_0\} = E_H(v, v').\]

3. If the rule of the transformation unit sum is applied, a new edge from a node \(v\) to a node \(v'\) is added with the label \(x + y\) provided that there is not already such an edge and there are an \(x\)-labelled edge from \(v\) some node \(\tau\) (with label 1) and a \(y\)-labelled edge from \(\tau\) to \(v'\). Due to the control condition this is iterated as long as possible. In other words, one gets the following characterization of the interleaving semantics of sum.

\[(G, H) \in SEM(\text{sum}) \iff V_G = V_H, l_G = l_H, E_G \subseteq E_H, s_G = s_H |_{E_G}, t_G = t_H |_{E_G}, \text{ and for all } v, v' \in V_G, e_0 \in E_H(v, v') - E_G(v, v') \text{ with } dist_H(e_0) = c \iff \text{there are } \tau \in V_G \text{ with } l_G(\tau) = 1, e \in E_G(v, \tau), e' \in E_G(\tau, v') \text{ with } dist_G(e) + dist_G(e') = c \text{ and there is no } \rho \in E_G(v, v') \text{ with dist}_G(\rho) = c.\]

4. \((G, H) \in SEM(\text{shortest-path})\) means according to the control condition that there is an interleaving sequence of the form \(G, G_0, \ldots, G_{4n}\) for some \(n \in \mathbb{N}\) with \(G_{4n} = H\) and

\[
\begin{align*}
(1) & \quad (G, G_0) \in SEM(\text{minimum}), \\
(2i) & \quad (G_{4i}, G_{4i+1}) \in SEM(\text{change}(0, 1)) \\
(3i) & \quad (G_{4i+1}, G_{4i+2}) \in SEM(\text{sum}) \\
(4i) & \quad (G_{4i+2}, G_{4i+3}) \in SEM(\text{change}(1, 2)) \\
(5i) & \quad (G_{4i+3}, G_{4i+4}) \in SEM(\text{minimum})
\end{align*}
\]

Due to 1. to 3., the set of nodes is invariant.

\(G\) is an initial graph meaning that \(l_G(v) = 0\) for all \(v \in V_G\). Using again the characterizations in 1. to 3., one gets \(l_G = l_{G_0}\), and, for \(i =\)

\[^8|E_H(v, v')|\) denotes the cardinality of \(E_H(v, v')\).

\[^9\text{For a function } f: A \to B \text{ and a set } C \subseteq A \text{ the restriction of } f \text{ to } C \text{ is denoted by } f|_C.\]
0, ..., n - 1, \ l_{G_{i+1}} = l_{G_{i+2}}, \ l_{G_{i+3}} = l_{G_{i+4}}. Moreover there is a node \ v_{i+1} \in V_{G_u} with \ l_{G_u}(v_{i+1}) = 0, \ l_{G_{i+1}}(v_{i+1}) = 1, \ l_{G_{i+3}}(v_{i+1}) = 2 as well as \ l_{G_u}(v) = l_{G_{i+1}}(v) and \ l_{G_{i+3}}(v) for all \ v \neq v_{i+1}.

Moreover, \ G_{4n} = H is a terminal graph such that \ l_{G_{4n}}(v) = 2 for all \ v \in V_{G_{4n}} = V_H = V_G. Hence \ n is the number of nodes because a single node is relabeled for each \ i = 0, ..., n - 1.

Let \ V_0 = \emptyset and \ V_j = \{v_1, ..., v_j\} for \ j = 1, ..., n. Then the following holds for \ j = 0, ..., n, for \ v, v' \in V_{G_u} with \ v \neq v' and for some \ c \in \mathbb{N} :

\[ e_0 \in E_{G_{i+1}}(v, v') \text{ with } \text{dist}_{G_u}(e_0) = c \iff \text{short}(v, v', V_{G_u} - V_j) = c. \]

The proof of this statement can be found in Subsection 15.3.7.

For \ n = j, this statement provides a characterization of the interleaving semantics of the transformation unit **shortest-path**.

\[(G, H) \in \text{SEM}(\text{shortest-path}) \text{ iff } V_G = V_H, l_G(v) = 0 \text{ and } l_H(v) = 2 \text{ for all } v \in V_G, \text{ and, for all } v, v' \in V_G \text{ with } v \neq v' \text{ and for some } c \in \mathbb{N}, \text{ there is some } e_0 \in E_H(v, v') \text{ with } \text{dist}_H(e_0) = c \iff \text{short}(v, v', V_G - V_j) = c. \]

In other words, **shortest-path** computes the distances of shortest paths in \ G.

### 15.3.6 Complexity

The length of a derivation reflects the complexity of the process to a certain degree because its products with lower and upper bounds for the cost of a direct derivation give corresponding bounds for the derivation. The length of derivations can be generalized to interleaving sequences (keeping in mind that derivations of different lengths may yield the same pair of graphs in the interleaving semantics).

Let \ (G, H) \in \text{SEM}(t_0) for some transformation unit without import. Then \ u(G, H) \ denotes the least upper bound of lengths of derivations from \ G \ to \ H \ (provided that the upper bound exists). Let \ s = G_0, ..., G_n \ be an interleaving sequence in the transformation unit \ t \ and let \ t_1, ..., t_k \ be the import units of \ t. \ Let, for \ i = 1, ..., n, \ \text{val}(G_{i-1}, G_i) = u(G_{i-1}, G_i) \ \text{if } (G_{i-1}, G_i) \in \text{SEM}(t_{j_i}) \ \text{for some } j_i \in \{1, ..., k\} \ \text{and } \text{val}(G_{i-1}, G_i) = 1 \ otherwise. \ Then \ \text{val}(s) = \sum_{i=1}^{n} \text{val}(G_{i-1}, G_i) \ is called the value of \ s.
Let \((G, H) \in SEM(t)\). Then \(u(G, H)\) denotes the least upper bound of values of interleaving sequences from \(G\) to \(H\) in \(t\) (provided that the upper bound exists).

Finally, let \(G\) be some initial graph of \(t\). Then \(u(G)\), called the upper length bound of \(G\), denotes the least upper bound of the \(u(G, H)\) for all graphs \(H\) with \((G, H) \in SEM(t)\) provided that such a graph and the bound exist.

Now the upper length bound for the initial graphs of shortest-path is presented. Let \(G\) be such a graph, \(n\) its number of nodes and \(m\) its multiplicity, i.e. the smallest number of edges one has to remove to transform \(G\) into a simple graph. Then \(u(G)\) is of the order \(\max(m, n^3)\).

This can be seen as follows. Consider the interleaving sequence in 4. (there is always one, and all have this form). Then one gets: (1) \(u(G, G_0) = m\) because each application of the rule of minimum decreases the multiplicity by 1.

(2i) \(u(G_{4i}; G_{4i+1}) = 1\) because of the control condition of change(0, 1).

(3i) \(u(G_{4i+1}, G_{4i+2}) \leq (n-1) \cdot (n-2)\) because \(v_{i+1}\) is the only 1-labelled node which may be adjacent to any of the \((n-1) \cdot (n-2)\) pairs of other nodes. Hence there may be as many occurrences for the rule of sum. But no occurrence can be used twice because of the negative application condition.

(4i) \(u(G_{4i+2}, G_{4i+3}) = 1\) because of the control condition of change (1, 2).

(5i) \(u(G_{4i+3}, G_{4i+4}) \leq (n-1) \cdot (n-2)\) because the derivation in (3i) produces up to \((n-1) \cdot (n-2)\) new edges that may be parallel to old ones. Hence minimum takes up to \((n-1) \cdot (n-2)\) rule applications to get rid of this multiplicity.

Altogether, this amounts to

\[
u(G, H) \leq m + \sum_{i=0}^{n-1} (1 + (n-1)(n-2) + 1 + (n-1)(n-2))
\]

\[
= m + 2n + 2n(n-1)(n-2)
\]

for all \(H\) with \((G, H) \in SEM(shortest-path)\) proving the statement for \(u(G)\).

The upper length bounds for shortest-path are in the order of upper bounds for the computational costs because all involved derivations can be organized in such a way that the cost of a direct derivation is constant. For this purpose, one needs direct access to the nodes making the steps (2i) and (4i) constant as well as to the pairs of nodes making the direct derivations in the other steps constant. This means that the transformation unit specification of Floyd's algorithm is exactly as efficient as the versions of the algorithm in the literature.
15.3. Shortest Paths  An Example

15.3.7 Completing the Proof

This subsection provides the proof of the statement of 15.3.5.4. It is done by
induction on $j$.

For $j = 0$, $e_0 \in E_G(v, v')$ means due to 2. and Subsection 15.3.4 that
we have $\text{dist}_{G_j}(e_0) = \min \{ \text{dist}_{G}(e) \mid e \in E_G(v, v') \} = \text{short}(v, v', V_G) = \text{short}(v, v', V_G - V_0)$.

Assume that the statement holds for some $j \geq 0$ and consider $j + 1$. If
$e_0 \in E_{G_{j+1}}(v, v')$, then either (1) $e_0 \in E_{G_{j+2}}(v, v')$ or (2) there are
e $e_1, e_2 \in E_{G_{j+2}}(v, v')$ with $\text{dist}_{G_{j+2}}(e_0) = \min \{ \text{dist}_{G_{j+2}}(e_1), \text{dist}_{G_{j+2}}(e_2) \}$
(cf. 2.). Moreover, $E_{G_{j+3}} = E_{G_{j+2}}$.

In the first case, either (1.1) $e_0 \in E_{G_{j+1}}(v, v')$ or (1.2) there are $e_3 \in E_{G_{j+1}}(v, v_{j+1})$
and $e_4 \in E_{G_{j+2}}(v_{j+1}, v')$ with $\text{dist}_{G_{j+2}}(e_0) = \text{dist}_{G_{j+1}}(e_3) + \text{dist}_{G_{j+1}}(e_4)$.
Moreover, $E_{G_{j+3}}(v, v') = E_{G_{j+1}}(v, v')$.

Therefore the induction hypothesis can be applied in all cases yielding the following:

(1.1) $\text{dist}_{G_{j+1}}(e_0) = \text{short}(v, v', V_G - V_j)$,

(1.2) $\text{dist}_{G_{j+1}}(e_3) = \text{short}(v, v_{j+1}, V_G - V_j)$ and $\text{dist}_{G_{j+1}}(e_4) = \text{short}(v_{j+1}, v', V_G - V_j)$

(2) $\text{dist}_{G_{j+1}}(e_1) = \text{short}(v, v', V_G - V_j)$, $\text{dist}_{G_{j+1}}(e_3) = \text{short}(v, v_{j+1}, V_G - V_j)$
and $\text{dist}_{G_{j+1}}(e_4) = \text{short}(v_{j+1}, v', V_G - V_j)$.

If $E_{G_{j+1}}(v, v_{j+1}) = \emptyset$ or $E_{G_{j+1}}(v_{j+1}, v') = \emptyset$, then $\text{short}(v, v_{j+1}, V_G - V_j)$
or $\text{short}(v_{j+1}, v', V_G - V_j)$ is undefined (otherwise there would be an edge from $v$
to $v_{j+1}$ and $v_{j+1}$ to $v'$). Hence $\text{short}(v, v', V_G - V_{j+1}) = \text{short}(v, v', V_G - V_j)$ as
shown in Subsection 15.3.4. This yields together with the other parts of case (1.1):

$\text{dist}_{G_{j+1}}(e_0) = \text{dist}_{G_{j+1}}(e_0) = \text{short}(v, v', V_G - V_{j+1}) = \text{short}(v, v', V_G - V_{j+1})$.

This remains true even if $E_{G_{j+1}}(v, v_{j+1}) \neq \emptyset \neq E_{G_{j+1}}(v_{j+1}, v')$ because
$\text{dist}_{G_{j+1}}(e_0) = \text{dist}_{G_{j+1}}(e_3) + \text{dist}_{G_{j+1}}(e_4)$ for $e_3 \in E_{G_{j+1}}(v, v_{j+1})$ and $e_4 \in E_{G_{j+1}}(v, v_{j+1})$. 

$E_{G_{i+1}}(v_{j+1}, v')$ otherwise the sum-step (3j) would produce a new edge from $v$ to $v'$ which is either case (1.2) or (2)).

In case (1.2), $E_{G_{i+1}}(v, v') = \emptyset$ because otherwise one would be faced with case (2). Using the result in Subsection 15.3.4 (together with the reasoning in (1.2)), one gets:

$$\text{dist}_{G_{i+4}}(e_0) = \text{dist}_{G_{i+4}}(e_3) + \text{dist}_{G_{i+4}}(e_4)$$

$$= \text{short}(v, v_{j+1}, V_G - V_j) + \text{short}(v_{j+1}, v', V_G - V_j),$$

$$= \text{short}(v, v', V_G - V_{j+1}).$$

It remains case (2) where one gets analogously:

$$\text{dist}_{G_{i+4}}(e_0) = \min\{\text{dist}_{G_{i+4}}(e_1), \text{dist}_{G_{i+4}}(e_2)\}$$

$$= \min\{\text{dist}_{G_{i+4}}(e_1), \text{dist}_{G_{i+4}}(e_3) + \text{dist}_{G_{i+4}}(e_4)\}$$

$$= \min\{\text{short}(v, v', V_G - V_j),$$

$$\text{short}(v, v_{j+1}, V_G - V_j) + \text{short}(v_{j+1}, v', V_G - V_j)\}$$

$$= \text{short}(v, v', V_G - V_{j+1}).$$

Conversely, let $\text{short}(v, v', V_G - V_{j+1}) = c$. Then there is some $p_0 \in \text{PATH}(v, v', V_G - V_{j+1})$ with $\text{dist}_{G_i}(p_0) = c$. If $v_{j+1} \not\in \text{inter}(p_0)$, then $p_0 \in \text{PATH}(v, v', V_G - V_j)$, and hence $\text{short}(v, v', V_G - V_j) = \text{short}(v, v', V_G - V_{j+1})$ (cf. Subsection 15.3.4). By induction hypothesis, one gets $e_0 \in E_{G_{i+4}}(v, v')$ with $\text{dist}_{G_{i+4}}(e_0) = c$. If $e_0$ is still in $G_{i+4}$, one is done. If $e_0$ is not in $G_{i+4}$, then the following happens: There are $e_1 \in E_{G_{i+4}}(v, v_{j+1})$ and $e_2 \in E_{G_{i+4}}(v_{j+1}, v')$ with $d = \text{dist}_{G_{i+4}}(e_1) + \text{dist}_{G_{i+4}}(e_2) \neq c$ such that $\text{sum}$ in step (3j) produces $e_4 \in E_{G_{i+4}}(v, v')$ with $\text{dist}_{G_{i+4}}(e_4) = d$. Therefore, $e_0$ and $e_4$ are replaced by minimum in step (5j) by $e_5 \in E_{G_{i+4}}(v, v')$ with $\text{dist}_{G_{i+4}}(e_5) = \min\{c, d\}$.

By induction hypothesis and the results in Subsection 15.3.4, one gets

$$d = \text{short}(v, v_{j+1}, V_G - V_j) + \text{short}(v_{j+1}, v', V_G - V_j)$$

$$\geq \text{short}(v, v', V_G - V_{j+1}) = c$$

such that one is done again.

It remains the case $v_{j+1} \in \text{inter}(p_0)$. Using the observations in Subsection 15.3.4, one gets

$$\text{short}(v, v_{j+1}, V_G - V_j) + \text{short}(v_{j+1}, v', V_G - V_j) = c.$$
After step (3j), this guarantees some \( e_3 \in E_{G_{d+2}}(v, v') \) with \( \text{dist}_{G_{d+2}}(e_3) = d_1 + d_2 = c \). If \( e_3 \) is still in \( G_{d+4} \), one is done again. Otherwise there is \( e_0 \in E_{G_d}(v, v') \) with \( d = \text{dist}_{G_d}(e_0) \neq c \), and minimum produces \( e_5 \in E_{G_{d+4}}(v, v') \) with \( \text{dist}_{G_{d+4}}(e_5) = \min\{c, d\} \) in step (5j). By induction hypothesis, and the results in Subsection 15.3.4, one gets

\[
d = \text{short}(v, v', V_G - V_j) \geq \text{short}(v, v', V_G - V_{j+1}) = c
\]
such that one is done also in this last case.

This completes the proof.

15.4 Transformation Modules

In the sample specification above, the transformation unit shortest-path is the unit of interest while all the others are of an auxiliary nature. Moreover, the shortest-path algorithm may be part of a route planning system providing further graph algorithms. Clearly, a specification language based on graph transformation should provide the means to put together several transformation units if they belong to the same application and to distinguish between main and auxiliary transformation units. This is accomplished by the notion of a transformation module that combines a set of transformation units. Some of them may be indicated as members of the export interface whereas the rest is hidden. In addition, there may be an import interface consisting of formal parameter units being transformation units of which only initial and terminal graphs are specified. This allows to leave parts of a system unspecified. They may be filled in later by instantiation.

The following notion of a module is a variant of the simple modules as proposed by Heckel et al. [17], who use a lightened form of units.

15.4.1 Formal Parameter Units and Modules

A transformation unit formal is a formal parameter unit if \( U_{\text{formal}} = \emptyset \), \( R_{\text{formal}} = \emptyset \), and \( C_{\text{formal}} = \text{true} \). While the semantic relation of formal is empty, the binary relation \( \text{SEM}(I_{\text{formal}}) \times \text{SEM}(T_{\text{formal}}) \) describes the upper bound of any actual parameter unit actual subject to the condition \( \text{SEM(actual)} \subseteq \text{SEM}(I_{\text{formal}}) \times \text{SEM}(T_{\text{formal}}) \).

A transformation module is a triple \( MOD = (\text{IMPORT}, \text{BODY}, \text{EXPORT}) \) where \( \text{IMPORT} \) is a set of formal parameter units, \( \text{BODY} \) is a set of transformation units each of which using only units from \( \text{BODY} \) and \( \text{IMPORT} \),
i.e. $U_t \subseteq IMPORT \cup BODY$ for each $t \in BODY$, and $EXPORT$ is a subset of $BODY \cup IMPORT$.

Based on the interleaving semantics of transformation units, the semantics of a transformation module $MOD = (IMPORT, BODY, EXPORT)$ is easily defined. Choosing a relation $SEM(im) \subseteq SEM(I_im) \times SEM(T_im)$ for each $im \in IMPORT$, $SEM(t)$ for each $t \in BODY - IMPORT$ is the interleaving semantics as defined level by level in the second section. Then $SEM(MOD)$ is just the restriction to the export interface, i.e. the family of relations $SEM(ex)$ for each $ex \in EXPORT$.

### 15.4.2 Composition of Modules

If one wants to instantiate some of the formal parameter units of a given module, one can choose exported transformation units of another module as actual parameters. Let $MOD = (IMPORT, BODY, EXPORT)$ and $MOD' = (IMPORT', BODY', EXPORT')$ be two transformation modules and $a: IMPORT \to EXPORT'$ be a partial mapping assigning export units of $MOD'$ to some import units of $MOD$. Then a composition $MOD \circ_a MOD'$ can be constructed by merging $IMPORT$ and $EXPORT'$ with respect to $a$.

In more detail, the domain of definition $dom_a$ of $a$ is removed from $IMPORT$, $BODY$ and $EXPORT$ are actualized according to $a$, meaning that each $im \in dom_a$ is replaced by $a(im)$ wherever it occurs in $BODY$ and $EXPORT$. Denoting the results by $a(BODY)$ and $a(EXPORT)$ resp., everything else is kept as it is. This yields

$$MOD \circ_a MOD' = ((IMPORT - dom_a) \cup IMPORT',$$
$$a(BODY) \cup BODY',$$
$$a(EXPORT')).$$

Alternatively, one could keep the export interface $EXPORT'$ in addition as export of the composition.

Obviously, the composition is again a transformation module such that one can get rid of all formal parameter units eventually by repeated composition.

### 15.4.3 Example

Floyd's algorithm as specified in the previous section can be represented as the transformation module
Floyd
  import:
  body:  shortest-path, minimum, sum,
         change(0, 1), change(1, 2)
  export: shortest-path

where the units minimum, sum, change(0, 1), and change(1, 2) are hidden and only the shortest-path unit is exported. Nothing is imported because the five units are completely defined by themselves.

As an illustration of the use of formal import parameters, consider the module 2-nodes which imports the formal parameter unit

  relabel
  initial:  node label 0
  terminal: node label 2

and the body of which consists of the three transformation units pre, post, and A_to_B where the latter is exported. The unit pre adds to an arbitrary initial graph two new nodes A and B, an edge from A to some old node and an edge from another old node to B. Moreover, the new edges get the distances 0 and all nodes are labelled with 0. The unit post removes all nodes except A and B and all edges except those connecting A and B. The explicit specifications of pre and post are omitted. Finally, the unit A_to_B is given by

A_to_B
  uses:  pre, relabel, post
  conds:  pre ; relabel ; post

The module 2-nodes may be seen as a kind of query concerning two nodes of a graph depending on the actual choice for relabel. Let SEM(relabel) be some relation between graphs with 0-labeled nodes and graphs with 2-labeled nodes. Then the semantic relation of A_to_B, which is also the semantics of the module takes an arbitrary graph, distinguishes two nodes through the call of pre, applies SEM(relabel) and restricts the result to the distinguished nodes (as far as they are still present). If one chooses particularly SEM(shortest-path), one gets the distance of the shortest path between the two distinguished nodes. Clearly, other actual parameters yield other answers.
The instantiation of \textit{relabel} by \textit{shortest-path} can be done explicitly by composition

\begin{verbatim}
import:
body:     pre, post, A\_to\_B, shortest-path,
         minimum, sum, change(0,1), change(1,2)
export:   A\_to\_B
\end{verbatim}

where the actualized form of \textit{A\_to\_B} is

\begin{verbatim}
A\_to\_B
uses:    pre, shortest-path, post
conds:   pre ; shortest-path ; post
\end{verbatim}

All other units are not affected by the instantiation. The actualization mapping is omitted because \textit{relabel} is the only formal parameter and \textit{shortest-path} the only exported unit that can be assigned.

\section{Conclusion}

In this chapter, we have introduced and illustrated the syntactic and semantic features of transformation units and transformation modules as structuring concepts for graph transformation systems. We have restricted ourselves to the case of acyclic use relations. The more complicated case of networks of transformation units which may use each other arbitrarily is studied in Krewski, Kuske, and Schiirr [22]. In this case, the semantic relations of the units in the network are aggregated by iterated interleaving semantics and yield a fixed point semantics under suitable assumptions on the control conditions. Moreover, the reader can find an investigation of operations on transformation units and corresponding operations on semantic relations in [21].

Transformation units and modules are the main structuring concepts of the rule and graph centered specification language \texttt{GRACE} (see [1,32,21]) which is not based on a particular graph transformation approach as other graph transformation languages. \texttt{GRACE} is planned as a visual language with a graphical interface that supports the visualized edition of graphs and rules and the animated execution of interleaving sequences. Moreover, \texttt{GRACE}
will provide an explanation component supporting structured correctness of proofs in particular. There is a very first experimental implementation of GrACE available: GrACeland provides a 3D editor and interpreter for the GrACE language including transformation units and modules, but is not yet approach-independent. The system is part of Martin Faust’s diploma thesis (see http://www.informatik.uni-bremen.de/grp/ag-ti/GRA Celand for more details).

In Chapter 17, transformation units and (a variant of) transformation modules are compared with other module concepts within the framework of graph transformation in various respects. It should be noted that most other notions are based on particular graph transformation approaches. Moreover, the comparison does not concern the semantic level explicitly while in our presentation the semantics is essential.

Although transformation units and transformation modules seem to work very well, they do not cover all aspects and needs of structuring. First, they allow to structure the set of rules of a graph transformation system, but the graphs are kept as whole entities which may get very large. A compatible structuring concept for cutting large graphs into small pieces is missing. Second, the underlying graph transformation approach may provide parallel rules and in this way parallel applications of rules. In all other respects, the interleaving semantics is sequential and does not describe concurrent system activities. Third, the interleaving semantics yields binary relations on graphs. This includes graph language generation and recognition if one fixes the first or second component of the relation. But what about n-ary relations on graphs with n > 2 or relations involving other data types than graphs? Fourth, the introduced notions are approach-independent. But all transformation units within a module are based on the same approach. The interconnection and interaction of units and modules would become more sophisticated if one would allow the coexistence of various approaches within a module supporting the switch from directed to undirected graphs for example. Fifth, a specification in terms of transformation modules describes the system on a fixed level of abstraction. A compatible notion of refinement is needed if one wants to get rid of this restriction (cf. [14]). There is still a lot to do. Future research will fill these and further deficiencies in one way or other.

References

1. Marc Andries, Gregor Engels, Annegret Habel, Berthold Hoffmann, Hans-Jörg Kreowski, Sabine Kuske, Detlef Plump, Andy Schürr, and
CHAPTER 15. TRANSFORMATION UNITS AND MODULES


REFERENCES


16. Annegret Habel, Reiko Heckel, and Gabriele Taentzer. Graph grammars with negative application conditions. *Fundamenta Informaticae, XXVII:1/2, 1996.*


32. Andy Schürr. Programmed graph transformations and graph transformation units in GRACE. In Cuny et al. [5], pages 122–136.