Formal Aspects of Computing (1999) 11: 690–723

Formal Aspects of Computing

Graph Transformation Units with Interleaving Semantics[1]

Hans-Jörg Kreowski and Sabine Kuske

University of Bremen, Department of Mathematics and Computer Science, Bremen, Germany

Keywords: Graph transformation, Transformation unit, Structuring, Formal methods, Interleaving semantics

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

Correspondence and offprint requests to: University of Bremen, Department of Mathematics and Computer Science, P.O. Box 33 04 40, D-28334 Bremen, email: kreo@informatik.uni-bremen.de

[1] This work was partially supported by the Deutsche Forschungsgemeinschaft, the ESPRIT Working Group Applications of Graph Transformation (APPLIGRAPH), and the EC TMR Network GETGRATS (General Theory of Graph Transformation Systems).

some typical examples. On the other hand, rules are used to describe "permitted" manipulations 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 the three volumes of the Handbook of Graph Grammars and Computing by Graph Transformation [Roz97, EEK99, EKM99] 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^{Ed} [Him91], Dactl [GKS91], and AGG [LöB93, TaB94]).

With the aim of enhancing the usefulness of graph transformation, we propose 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 Berlin, Bremen, Erlangen, München, Oldenburg, and Paderborn (see also [AEH99, Sch96]). 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 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.

A short draft version of this paper without proofs is published as [KrK96].

2. Transformation Units with Interleaving Semantics

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., [Bun79, Nag79, EhH86, KrR90, MaW91, Sch91a, ScZ91, Kre93, LiM93, HHT96, MaW96], cf. also [DaP89] 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 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 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.

2.1. Graph Transformation Approach

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

- $\mathscr{G}$ is a class of *graphs*,
- $\mathscr{R}$ is a class of *rules*,
- $\Rightarrow$ is a *rule application operator* yielding a binary relation $\Rightarrow_r \subseteq \mathscr{G} \times \mathscr{G}$ for every $r \in \mathscr{R}$,
- $\mathscr{E}$ is a class of *graph class expressions* such that each $e \in \mathscr{E}$ specifies a subclass $SEM(e) \subseteq \mathscr{G}$, and
- $\mathscr{C}$ is a class of *elementary control conditions* over some set ID of identifiers such that each $c \in \mathscr{C}$ specifies a binary relation $SEM_E(c) \subseteq \mathscr{G} \times \mathscr{G}$ for each mapping $E: ID \longrightarrow 2^{\mathscr{G} \times \mathscr{G}}$.[1]

A pair $(G, G') \in \Rightarrow_r$, usually written as $G \Rightarrow_r G'$, is called a *direct derivation* from G to G' through r. For a set $P \subseteq \mathscr{R}$ 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 $E: ID \longrightarrow 2^{\mathscr{G} \times \mathscr{G}}$ is called an *environment*. In the following, we will use boolean expressions over $\mathscr{C}$ as *control conditions* with elementary control conditions as basic elements and disjunction, conjunction, and negation as boolean operators.

[1] The power set of a set S is denoted by 2^S.

Moreover, we make use of the constant *true*. The semantic relations of elementary control conditions are easily extended to boolean expressions by

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

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

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 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. [Ehr79, Nag79, JaR80, Cou90, KrR90, Sch91b, Him91, Hab92, Löw93]). Therefore, all of them can be considered as graph transformation approaches in the above sense if one chooses the components $\mathscr{E}$ and $\mathscr{C}$ in some standard way. The singleton set $\{all\}$ with $SEM(all) = \mathscr{G}$ may provide the only graph class expression, and the class of elementary control conditions may be empty. Non-trivial choices for $\mathscr{E}$ and $\mathscr{C}$ are discussed in Subsections 2.4 and 2.5.

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

This should be taken as a recursive definition of the set $\mathscr{T}_{\mathscr{A}}$ of transformation units over $\mathscr{A}$. 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. Note that this yields an acyclic import structure with finite recursion depth.

If I specifies a single graph (cf. 2.4.1), 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.

2.3. 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 $\mathscr{A} = (\mathscr{G}, \mathscr{R}, \Rightarrow, \mathscr{E}, \mathscr{C})$. Assume that the set *ID* of identifiers

associated to $\mathscr{C}$ contains the disjoint union of U and R. Let the interleaving semantics $SEM(t) \subseteq \mathscr{G} \times \mathscr{G}$ for $t \in U$ be already defined. Let $E(trut): ID \to 2^{\mathscr{G} \times \mathscr{G}}$ be defined by $E(trut)(r) = \Rightarrow_r$ 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 $(G, G') \in \mathscr{G} \times \mathscr{G}$ such that

1. $G \in SEM(I)$ and $G' \in SEM(T)$,
2. there are graphs $G_0, \ldots, G_n \in \mathscr{G}$ 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 $(G_{i-1}, G_i) \in SEM(t)$ 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. $RIS_{trut} = (\Rightarrow_R \cup \bigcup_{t \in U} SEM(t))^*$. 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 because the import structure is assumed to be acyclic. Initially, if U is empty, an interleaving sequence is just a derivation such that one gets in this case

$$SEM(trut) = \Rightarrow_R^* \cap (SEM(I) \times SEM(T)) \cap SEM_{E(trut)}(C).$$

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

If I is a single graph (specifying itself as initial graph in the sense of 2.4.1), the first component of the interleaving semantics of *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. 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.

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 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 (up to isomorphism), i.e. $SEM(G) = \{G' \in \mathscr{G} \mid G' \cong G\}$. 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 $P \subseteq \mathscr{R}$ if there is no $G' \in \mathscr{G}$ 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 $\mathscr{G}$ is a class of labelled graphs with label alphabet Σ, then a set $T \subseteq \Sigma$ is a suitable graph class expression specifying the graph class $SEM(T) = \mathscr{G}_T$ 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]).
5. Graph schemata, as used in the graph transformation approach PROGRES are graph class expressions that allow to specify generic graph classes (see [Sch91a, Sch91b] for more details).
6. In a transformation unit where a single initial graph is transformed into terminal graphs, only the class of all transformed graphs is significant.[2] Hence, such a transformation unit can be used as a graph class expression specifying the second components of all pairs in its semantics.

2.5. Control Conditions

A control condition is meant to restrict the derivation process. A typical example is to allow only iterated rule applications where the sequences of applied rules belong to a particular control language. Therefore, a regular expression can be considered as a control condition because it identifies 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 $E: ID \to 2^{\mathscr{G} \times \mathscr{G}}$ be an environment. Then E can be extended to the power set of the set of strings over ID in a natural, straight-forward way. $\widehat{E}: 2^{ID^*} \to 2^{\mathscr{G} \times \mathscr{G}}$ is defined by $\widehat{E}(L) = \bigcup_{w \in L} \overline{E}(w)$ for $L \subseteq ID^*$ where $\overline{E}: ID^* \to 2^{\mathscr{G} \times \mathscr{G}}$ is recursively given by $\overline{E}(\lambda) = \Delta\mathscr{G}$, and $\overline{E}(xv) = E(x) \circ \overline{E}(v)$ for $x \in ID$ and $v \in ID^*$.[3] Hence, L can be used as control condition with $SEM_E(L) = \widehat{E}(L)$ for

[2] An example of such a transformation unit is the unit *butterfly* given in Section 3.

[3] $\Delta\mathscr{G}$ denotes the identity relation on $\mathscr{G}$. Given $\rho, \rho' \subseteq \mathscr{G} \times \mathscr{G}$, the sequential composition of ρ and ρ' is defined as usual by $\rho \circ \rho' = \{(G, G'') \mid (G, G') \in \rho \text{ and } (G', G'') \in \rho' \text{ for some } G' \in \mathscr{G}\}$.

all $E: ID \to 2^{\mathscr{G} \times \mathscr{G}}$. In this case, the class of elementary control conditions is 2^{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 ID can serve as a control condition with $SEM_E(x) = SEM_E(L(x)) = \widehat{E}(L(x))$ for all environments E. Whenever grammars are used as control conditions in the following, it is meant in this sense.
3. 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 $\emptyset, \epsilon \in REG(ID)$[4], $ID \subseteq REG(ID)$, and $(e_1\ ;\ e_2), (e_1 \mid e_2), (e^*) \in REG(ID)$ if $e, e_1, e_2 \in REG(ID)$. In order to omit parentheses we assume that * has a stronger binding than ; and | , and that ; has a stronger binding than | . The language $L(e)$ specified by some regular expression e is defined as $L(\emptyset) = \{\}$, $L(\epsilon) = \{\lambda\}$, $L(id) = \{id\}$ for all $id \in ID$, $L(e_1\ ;\ e_2) = L(e_1) \cdot L(e_2)$, $L(e_1 \mid e_2) = L(e_1) \cup L(e_2)$ and $L(e^*) = L(e)^*$[5].
4. A pair (G, G') of graphs is *reduced* with respect to a control condition $c \in \mathscr{C}$ and an environment E if there is no graph G'' with $(G', G'') \in SEM_E(c)$. In this way, $c\,!$ defines a control condition where $SEM_E(c\,!)$ is the set of all reduced pairs with respect to c and E. Note that if c is a set of graph transformation rules with $SEM_E(c) = \Rightarrow_c^*$ for all environments E, the set of the second components of the pairs in $SEM_E(c)$ corresponds to the graph class expression c (specifying all reduced graphs with respect to c) introduced in Subsection 2.4.
5. A pair $(C, <)$ consisting of a set C of control conditions and a partial order $<$ on C called *priority* is a control condition. For a given environment E, the semantics of $(C, <)$ is defined as follows: Let $S_E((C, <))$ consist of all pairs (G, G') of graphs such that there is a condition $c \in C$ with $(G, G') \in SEM_E(c)$ and for all $c' \in C$ with $c < c'$, $\{G'' \in \mathscr{G} \mid (G, G'') \in SEM_E(c')\} = \emptyset$. Then $SEM_E(C, <)$ is the reflexive and transitive closure of $S_E((C, <))$.
6. In all major graph transformation approaches a rule r is applied to a graph G at a so-called *occurrence* which is a subgraph of G. In general, there may be several occurrences for r in G. Given such an approach, another kind of control condition with priorities is $pr(P, <)$ where P is a set of rules and $<$ is a partial order on P. For each environment E, $SEM_E(pr(P, <))$ is the reflexive and transitive closure of $S_E(pr(P, <))$ where $S_E(pr(P, <))$ consists of all pairs (G, G') of graphs such that G' is obtained from G by applying a rule $r \in P$ at an occurrence occ, and for all $r' \in P$ with $r < r'$, no occurrence of r' in G overlaps with occ. These control conditions with priorities are introduced for a particular graph transformation approach in [LiM93].
7. Each transformation unit *trut* 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 *trut* is given by the semantics of *trut*, i.e. by all pairs (G, G') of graphs such that G can be transformed into G' with the transformation unit *trut*.
8. Each pair $(e_1, e_2) \in \mathscr{E} \times \mathscr{E}$ defines a binary relation on graphs by

$$SEM((e_1, e_2)) = SEM(e_1) \times SEM(e_2)$$

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

[5] Given $L, L' \subseteq ID^*$, the concatenation of L and L' is defined as usual by $L \cdot L' = \{ww' \mid w \in L\ ,\ w' \in L'\}$, and the Kleene closure of L is defined as usual by $L^* = \bigcup_{i=0}^{\infty} L^i$ where $L^0 = \{\lambda\}$ and $L^{i+1} = L \cdot L^i$.

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.

9. 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.

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.

Observation 2.1. For all environments E, all $id \in ID$ and all regular expressions $e, e_1, e_2 \in REG(ID)$ the following holds.

1. $SEM_E(\emptyset) = \{\}$.
2. $SEM_E(\epsilon) = \Delta\mathscr{G}$.
3. $SEM_E(id) = E(id)$.
4. $SEM_E(e_1 \,;\, e_2) = SEM_E(e_1) \circ SEM_E(e_2)$.
5. $SEM_E(e_1 \mid e_2) = SEM_E(e_1) \cup SEM_E(e_2)$.
6. $SEM_E(e^*) = SEM_E(e)^*$.[6]

Proof.

1. $SEM_E(\emptyset) =_{def} \widehat{E}(\{\}) =_{def} \{\}$.[7]
2. $SEM_E(\epsilon) =_{def} \widehat{E}(\{\lambda\}) =_{def} \overline{E}(\lambda) =_{def} \Delta\mathscr{G}$.
3. $SEM_E(id) =_{def} \widehat{E}(\{id\}) =_{def} \overline{E}(id) =_{def} E(id)$.
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 $\overline{E}(w_1 w_2) = \overline{E}(w_1) \circ \overline{E}(w_2)$. $\overline{E}(\lambda w_2) =_{def} \overline{E}(w_2) = \Delta\mathscr{G} \circ \overline{E}(w_2) =_{def} \overline{E}(\lambda) \circ \overline{E}(w_2)$. Assume that the statement holds for $w_1 \in ID^*$, and let $a \in ID$. Then $\overline{E}(a w_1 w_2) =_{def} E(a) \circ \overline{E}(w_1 w_2) =_{ind} E(a) \circ (\overline{E}(w_1) \circ \overline{E}(w_2)) = (E(a) \circ \overline{E}(w_1)) \circ \overline{E}(w_2) =_{def} \overline{E}(a w_1) \circ \overline{E}(w_2)$.[8] Now we get

$$\begin{aligned} SEM_E(e_1 \,;\, e_2) &=_{def} \widehat{E}(L(e_1) \cdot L(e_2)) =_{def} \textstyle\bigcup_{w_1 \in L(e_1), w_2 \in L(e_2)} \overline{E}(w_1 w_2) \\ &= \textstyle\bigcup_{w_1 \in L(e_1), w_2 \in L(e_2)} \overline{E}(w_1) \circ \overline{E}(w_2) \\ &= \textstyle\bigcup_{w_1 \in L(e_1)} \overline{E}(w_1) \circ \bigcup_{w_2 \in L(e_2)} \overline{E}(w_2) =_{def} \widehat{E}(L(e_1)) \circ \widehat{E}(L(e_2)) \\ &=_{def} SEM_E(e_1) \circ SEM_E(e_2). \end{aligned}$$

5. $$\begin{aligned} SEM_E(e_1 \mid e_2) &=_{def} \widehat{E}(L(e_1) \cup L(e_2)) =_{def} \textstyle\bigcup_{w \in L(e_1) \cup L(e_2)} \overline{E}(w) \\ &= \textstyle\bigcup_{w_1 \in L(e_1)} \overline{E}(w_1) \cup \bigcup_{w_2 \in L(e_2)} \overline{E}(w_2) =_{def} \widehat{E}(L(e_1)) \cup \widehat{E}(L(e_2)) \\ &=_{def} SEM_E(e_1) \cup SEM_E(e_2). \end{aligned}$$
6. To show point 6, we first prove by induction on i that for $i \geqslant 0$, $\widehat{E}(L(e)^i) = SEM_E(e)^i$. If $i = 0$ we have $\widehat{E}(L(e)^0) =_{def} \widehat{E}(\{\lambda\}) =_{2.} \Delta\mathscr{G} =_{def} SEM_E(e)^0$. Moreover,

[6] For $\rho \subseteq \mathscr{G} \times \mathscr{G}$, ρ^* denotes the reflexive and transitive closure of ρ that is $\rho^* = \bigcup_{i=0}^{\infty} \rho^i$ where $\rho^0 = \Delta\mathscr{G}$ and $\rho^{i+1} = \rho \circ \rho^i$.

[7] $=_{def}$ stands for *equal by definition.*

[8] $=_{ind}$ stands for *equal by induction hypothesis.*

$$\widehat{E}(L(e)^{i+1}) =_{def} \widehat{E}(L(e) \cdot L(e)^i) =_{4.} \widehat{E}(L(e)) \circ \widehat{E}(L(e)^i)$$
$$=_{ind} SEM_E(e) \circ SEM_E(e)^i =_{def} SEM_E(e)^{i+1}.$$

Hence,

$$SEM_E(e^*) =_{def} \widehat{E}(L(e)^*) =_{def} \widehat{E}(\bigcup_{i=0}^{\infty} L(e)^i) = \bigcup_{i=0}^{\infty} \widehat{E}(L(e)^i)$$
$$= \bigcup_{i=0}^{\infty} SEM_E(e)^i =_{def} SEM_E(e)^*.$$

□

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 2.5.1.

Let $trut = (I, U, R, C, T)$ be a transformation unit over the graph transformation approach $\mathscr{A} = (\mathscr{G}, \mathscr{R}, \Rightarrow, \mathscr{E}, \mathscr{C})$. Assume that U and R are disjoint subsets of the set ID associated to $\mathscr{C}$. Then $x_1 \cdots x_n \in (U \cup R)^*$ $(x_i \in U \cup R)$ is called an *application sequence* of $(G, G') \in \mathscr{G} \times \mathscr{G}$ 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 λ.

Using these notions and notations, the following observation states that a language over $U \cup R$, used as a control condition due to 2.5.1, controls the order in which rules are applied and imported transformation units are actually used.

Observation 2.2. Let $\mathscr{C} = 2^{ID^*}$ 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 $G, G' \in \mathscr{G}$, 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 $(G, G') \in SEM_{E(trut)}(L) = \widehat{\overline{E(trut)}}(L)$ 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 $(G, G') \in \overline{E(trut)}(\lambda)$ iff $(G, G') \in \Delta\mathscr{G}$ iff $G = G'$ iff λ 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 $(G, G') \in \overline{E(trut)}(xv) = E(trut)(x) \circ \overline{E(trut)}(v)$, means that there is some $\overline{G} \in \mathscr{G}$ 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 $G \Rightarrow_x \overline{G}$ 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, $G_0 \Rightarrow_x G_1$ if $x \in R$ and $(G_0, G_1) \in SEM(x)$ if $x \in U$ such that $(G, G_1) \in E(trut)(x)$ 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 $(G_1, G') \in \overline{E(trut)}(v)$. The composition yields $(G, G') \in E(trut)(x) \circ \overline{E(trut)}(v) = \overline{E(trut)}(xv)$. 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, we have that 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. □

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 ($0 \leqslant i < 2^k$, $0 \leqslant r \leqslant k$). Then for $r > 0$, v_{ir} is directly connected to $v_{j(r-1)}$ if either $i = j$ or $b_i(1) \ldots b_i(k)$ is equal to the binary representation of j except for bit $b_i(r)$. A butterfly network of size k is denoted by B_k. 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. The butterfly networks B_0 to B_3 can be graphically represented as in Fig. 1.

Note that butterfly networks are defined up to isomorphism. The set of all butterfly networks is denoted by $L_{butterfly}$.

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 [HHT96]).

- The class $\mathcal{G}$ consists of all graphs $G = (V, E, l, m)$ where V is the set of *nodes*, E is the set of *edges* being 2-element subsets of V, $l: V \to C_V$, and $m: E \to C_E$ are labelling functions for nodes and edges respectively with $C_V = \{*, b\}$ and $C_E = \{*, c\}$. The symbol $*$ stands for *unlabelled*, b for *bottom*, and c for *copied*. The components of G are also denoted by V_G, E_G, l_G, and m_G, respectively. 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.
- The application of a *rule* $r = (N, L, R)$ to a graph $G \in \mathcal{G}$ yields a graph in $\mathcal{G}$ and is performed according to the following steps.

  1. Choose $L' \subseteq G$ with $L' \cong L$.

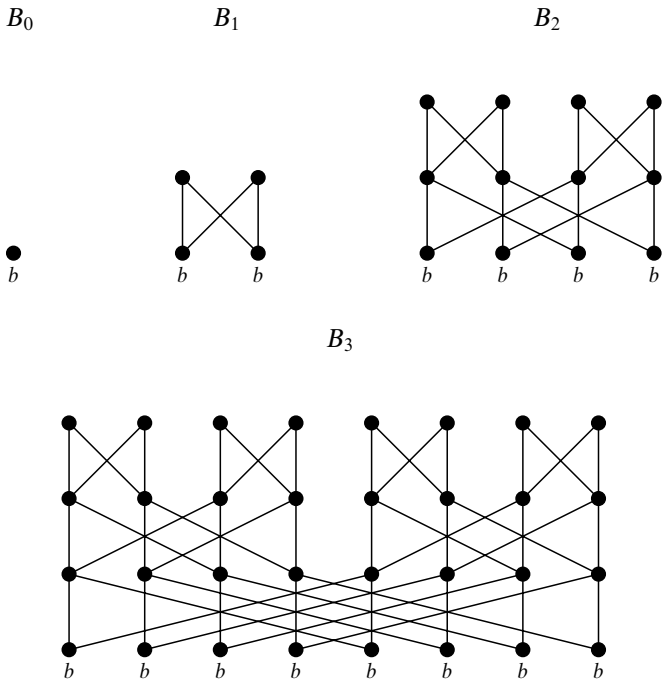


Fig. 1. Butterfly networks of sizes 0 to 3.

2. Only if L is a proper subgraph of N, 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 $v \in V_L$ ($\subseteq V_R$) 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.

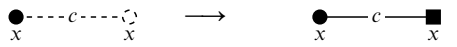
The transformation unit *butterfly* uses the transformation units *copy* and *next_rank* and applies them repeatedly in that order which is guaranteed by the control condition. The initial graph is the butterfly network of size 0. After k calls of *copy* and *next_rank* the butterfly network of size k is generated (cf. the theorem below).

butterfly
initial: ● *b*
uses: *copy*, *next_rank*
conds: (*copy* ; *next_rank*)*

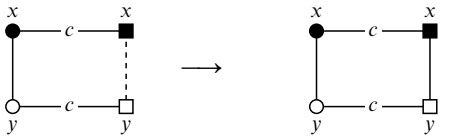
The transformation unit *copy* uses the transformation units *copy_items* and *delete* which are applied exactly once in this order.

copy
uses: *copy_items*, *delete*
conds: *copy_items* ; *delete*

The transformation unit *copy_items* transforms an input graph B into a graph B' by copying each node and $*$-labelled edge of B (together with the labels) and generates a c-labelled edge between each node of B and its copy.

copy_items
rules: r_1 : 

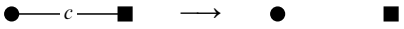
$x \in \{*, b\}$

r_2: 

$x \in \{*, b\}$

terminal: *reduced*

The transformation unit *delete* removes each c-labelled edge provided that it connects $*$-labelled nodes.

delete
rules: 

terminal: *reduced*

The rule set of *copy_items* is the union of the two rule sets r_1 and r_2 where r_1 contains two rules for copying the nodes of B (one rule for the $*$-labelled and the other one for the bottom nodes). The two rules of r_2 copy the edges of B (one rule for the edges between $*$-labelled nodes and the other one for those connecting $*$-labelled with bottom nodes). The rule set of *delete* contains a single rule deleting c-labelled edges between unlabelled nodes. Note that the node labels and the c-labelled edges, used in the rules of *copy_items* and *delete* 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* and *delete* 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.

If the input graph of the transformation unit *copy* is a butterfly network B of size k the output graph of *copy* without the c-labelled edges and the b-labels corresponds 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*

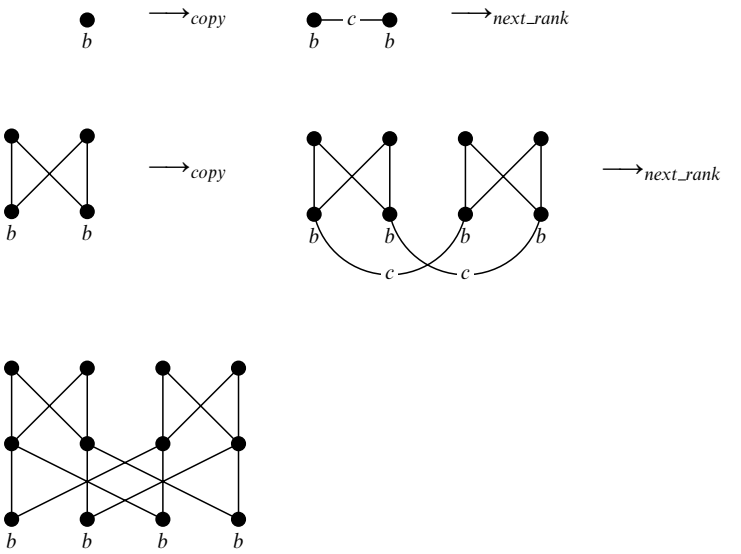
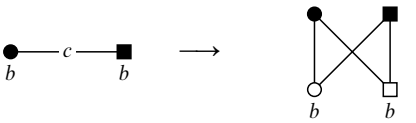


Fig. 2. An interleaving sequence of *butterfly*.

transforms this graph into a butterfly network of size $k+1$ by adding these missing bottom nodes and connecting edges.

next_rank
rules:



terminal: *reduced*

In Fig. 2, an interleaving sequence of *butterfly* is shown. It generates 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'.

The second *copy* step is given by the interleaving sequence in Fig. 3 where one possible derivation for the *copy_items* step is depicted in Fig. 4.

The transformation unit *butterfly* is language generating, and its language can be shown to consist exactly of all butterfly networks i.e., we have the following correctness result.

Theorem. $L_{butterfly} = L(butterfly)$.

The proof is given in the appendix. It makes use of the structure of *butterfly*

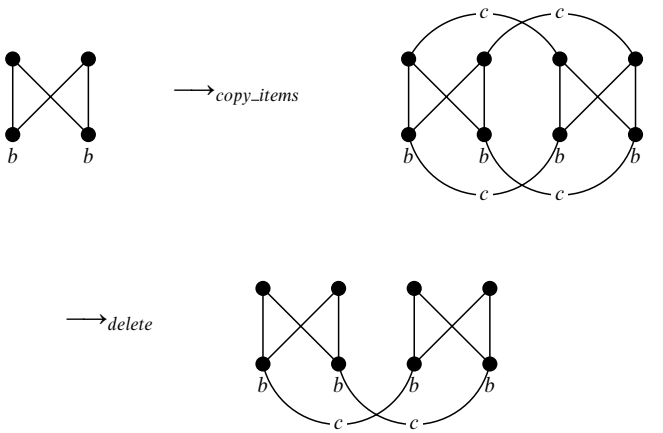


Fig. 3. An interleaving sequence of *copy*.

by giving corresponding correctness results for the used transformation units and putting them together in a suitable way.

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, inversion, 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 Inversion

The definition of the interleaving semantics of transformation units is based on the union, the intersection, 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 4.1. Let $\mathscr{A} = (\mathscr{G}, \mathscr{R}, \Rightarrow, \mathscr{E}, \mathscr{C})$ be a graph transformation approach

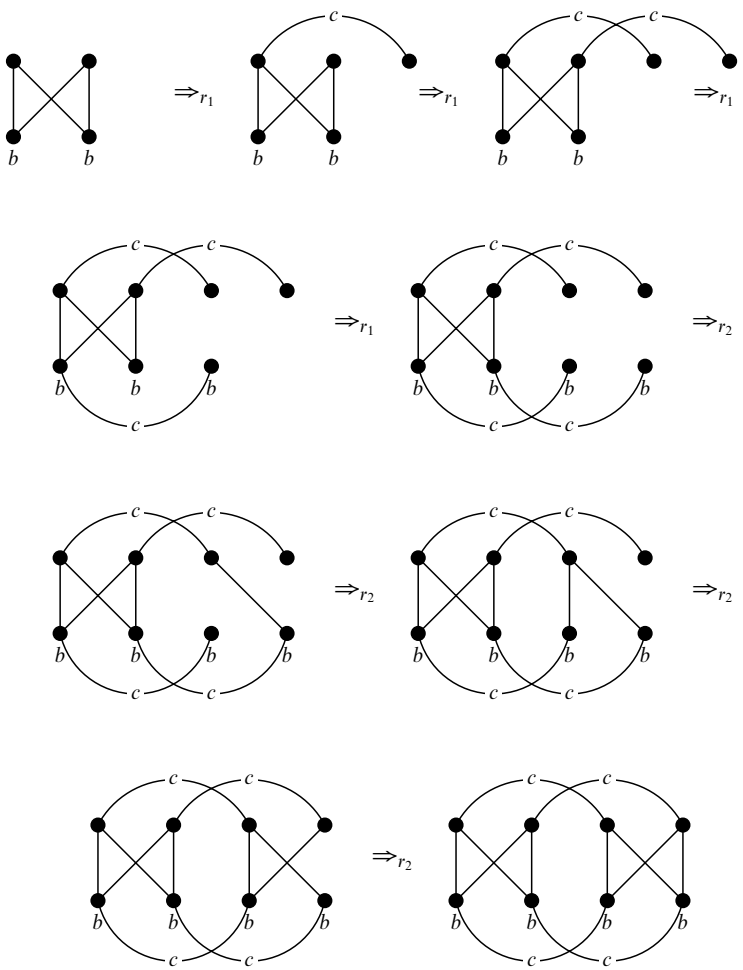


Fig. 4. An interleaving sequence of *copy_items*.

where $REG(ID) \subseteq \mathscr{C}$ and let $t, t' \in \mathscr{T}_{\mathscr{A}}$. Let $trans(t)$, $refl(t)$, $union(t,t')$, $sc(t,t') \in \mathscr{T}_{\mathscr{A}}$ be defined 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:	$t\ ;\ t'$

Then

1. $SEM(trans(t)) = SEM(t)^+$ [9],
2. $SEM(refl(t)) = SEM(t) \cup \Delta\mathscr{G}$,
3. $SEM(union(t,t')) = SEM(t) \cup SEM(t')$,
4. $SEM(sc(t,t')) = SEM(t) \circ SEM(t')$.

Proof. For $trut \in \{trans(t), refl(t), union(t,t'), sc(t,t')\}$ we have

$$\begin{aligned} SEM(trut) &=_{obs.2.2} (SEM(I_{trut}) \times SEM(T_{trut})) \cap SEM_{E(trut)}(C_{trut}) \\ &=_{def} (\mathscr{G} \times \mathscr{G}) \cap SEM_{E(trut)}(C_{trut}) = SEM_{E(trut)}(C_{trut}). \end{aligned}$$ [10]

Then the four statements are shown as follows:

1. $$\begin{aligned} SEM(trans(t)) &= SEM_{E(trans(t))}(t\,;t^*) \\ &=_{obs.2.1} SEM_{E(trans(t))}(t) \circ SEM_{E(trans(t))}(t^*) \\ &=_{obs.2.1} SEM_{E(trans(t))}(t) \circ SEM_{E(trans(t))}(t)^* =_{def} SEM(t) \circ SEM(t)^* \\ &=_{def} SEM(t) \circ \textstyle\bigcup_{i=0}^{\infty} SEM(t)^i \\ &= \textstyle\bigcup_{i=1}^{\infty} SEM(t)^i =_{def} SEM(t)^+. \end{aligned}$$
2. $$\begin{aligned} SEM(refl(t)) &= SEM_{E(refl(t))}(t \mid \epsilon) \\ &=_{obs.2.1} SEM_{E(refl(t))}(t) \cup SEM_{E(refl(t))}(\epsilon) \\ &=_{obs.2.1} SEM_{E(refl(t))}(t) \cup \Delta\mathscr{G} =_{def} SEM(t) \cup \Delta\mathscr{G}. \end{aligned}$$
3. $$\begin{aligned} SEM(union(t,t')) &= SEM_{E(union(t,t'))}(t \mid t') \\ &=_{obs.2.1} SEM_{E(union(t,t'))}(t) \cup SEM_{E(union(t,t'))}(t') \\ &=_{def} SEM(t) \cup SEM(t'). \end{aligned}$$
4. $$\begin{aligned} SEM(sc(t,t')) &= SEM_{E(sc(t,t'))}(t\,;\,t') \\ &=_{obs.2.1} SEM_{E(sc(t,t'))}(t) \circ SEM_{E(sc(t,t'))}(t') =_{def} SEM(t) \circ SEM(t'). \end{aligned}$$

This completes the proof. □

In a similar way, the intersection of transformation units can be modelled.

Observation 4.2. Let $\mathscr{A} = (\mathscr{G}, \mathscr{R}, \Rightarrow, \mathscr{E}, \mathscr{C})$ be a graph transformation approach where $REG(ID) \subseteq \mathscr{C}$ and let $t, t' \in \mathscr{T}_{\mathscr{A}}$. Let $intersect(t,t') \in \mathscr{T}_{\mathscr{A}}$ be defined as follows:

$intersect(t,t')$
uses: t, t'
conds: $t \wedge t'$

Then $SEM(intersect(t,t')) = SEM(t) \cap SEM(t')$.

Proof.

$$\begin{aligned} SEM(intersect(t,t')) &=_{def} (SEM(t) \cup SEM(t'))^* \cap SEM_{E(intersect(t,t'))}(t \wedge t') \\ &=_{def} (SEM(t) \cup SEM(t'))^* \cap \\ &(SEM_{E(intersect(t,t'))}(t) \cap SEM_{E(intersect(t,t'))}(t')) \\ &=_{def} (SEM(t) \cup SEM(t'))^* \cap (SEM(t) \cap SEM(t')) = SEM(t) \cap SEM(t'). \end{aligned}$$

□

[9] For a binary relation ρ, ρ^+ denotes the transitive closure of ρ that is $\rho^+ = \bigcup_{i=1}^{\infty} \rho^i$.
[10] $=_{obs.2.2}$ stands for *equal by Observation 2.2.*

4.2. Operations with Inversion

Under the assumption that all rules of a transformation unit are invertible and that there is a suitable control condition, the inversion 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.

The control condition assumed in the transformation unit for inversion can be explicitely constructed in special cases (cf. Observation 4.4).

Observation 4.3. Let $\mathscr{A} = (\mathscr{G}, \mathscr{R}, \Rightarrow, \mathscr{E}, \mathscr{C})$ be a graph transformation approach in which, for each $r \in \mathscr{R}$, there is an $r^{-1} \in \mathscr{R}$ such that $(\Rightarrow_r)^{-1} = \Rightarrow_{r^{-1}}$ [11], and let $t \in \mathscr{T}_{\mathscr{A}}$. Let $inv(t)$, $sym(t)$ and $equiv(t) \in \mathscr{T}_{\mathscr{A}}$ be defined as follows:

$inv(t)$

initial:	T_t
uses:	$\{inv(t') \mid t' \in U_t\}$
rules:	$\{r^{-1} \mid r \in R_t\}$
conds:	C_t^{-1}
terminal:	I_t

where C_t^{-1} is some control condition with

$$SEM_{E(inv(t))}(C_t^{-1}) = (SEM_{E(t)}(C_t))^{-1}.$$

$sym(t)$

uses:	$union(t, inv(t))$
conds:	$union(t, inv(t))$

$equiv(t)$

uses:	$trans(refl(sym(t)))$
conds:	$trans(refl(sym(t)))$

Then

1. $SEM(inv(t)) = SEM(t)^{-1}$,
2. $SEM(sym(t)) = SEM(t) \cup SEM(t)^{-1}$,
3. $SEM(equiv(t)) = equiv(SEM(t))$.[11]

Proof. In this and the following proof we make use of the facts that for all binary relations P, Q over some set R, $(P \cap Q)^{-1} = P^{-1} \cap Q^{-1}$, $(P \cup Q)^{-1} = P^{-1} \cup Q^{-1}$, $(P \circ Q)^{-1} = Q^{-1} \circ P^{-1}$, $(P^*)^{-1} = (P^{-1})^*$, and $((R \times R) - P)^{-1} = (R \times R) - P^{-1}$.

1. (By induction on the recursion depth of t.)
   If $U_t = \{\}$ then by definition $U_{inv(t)} = \{\}$ and we get

$$\begin{aligned} SEM(inv(t)) &=_{def} (SEM(T_t) \times SEM(I_t)) \cap \Rightarrow^*_{R_{inv(t)}} \cap SEM_{E(inv(t))}(C_t^{-1}) \\ &=_{def} (SEM(I_t) \times SEM(T_t))^{-1} \cap ((\Rightarrow_{R_t})^{-1})^* \cap SEM_{E(t)}(C_t)^{-1} \\ &= (SEM(I_t) \times SEM(T_t))^{-1} \cap (\Rightarrow^*_{R_t})^{-1} \cap SEM_{E(t)}(C_t)^{-1} \\ &= ((SEM(I_t) \times SEM(T_t)) \cap \Rightarrow^*_{R_t} \cap SEM_{E(t)}(C_t))^{-1} = SEM(t)^{-1}. \end{aligned}$$

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

If $U_t \neq \{\}$ assume that for all $t' \in U_t$ $SEM(inv(t')) = SEM(t')^{-1}$. Then for the semantics of $inv(t)$ we get

$$\begin{aligned}
SEM(inv(t)) &=_{def} (SEM(T_t) \times SEM(I_t)) \cap \\
&\quad (\Rightarrow_{R_{inv(t)}} \cup \bigcup_{t' \in U_t} SEM(inv(t')))^* \cap SEM_{E(inv(t))}(C_t^{-1}) \\
&=_{ind} (SEM(I_t) \times SEM(T_t))^{-1} \cap ((\Rightarrow_{R_t})^{-1} \cup \bigcup_{t' \in U_t} SEM(t')^{-1})^* \cap \\
&\quad SEM_{E(t)}(C_t)^{-1} \\
&= (SEM(I_t) \times SEM(T_t))^{-1} \cap \\
&\quad ((\Rightarrow_{R_t} \cup \bigcup_{t' \in U_t} SEM(t'))^{-1})^* \cap SEM_{E(t)}(C_t)^{-1} \\
&= (SEM(I_t) \times SEM(T_t))^{-1} \cap ((\Rightarrow_{R_t} \cup \bigcup_{t' \in U_t} SEM(t'))^*)^{-1} \cap \\
&\quad SEM_{E(t)}(C_t)^{-1} \\
&= ((SEM(I_t) \times SEM(T_t) \cap (\Rightarrow_{R_t} \cup \bigcup_{t' \in U_t} SEM(t'))^* \cap \\
&\quad SEM_{E(t)}(C_t))^{-1} =_{def} SEM(t)^{-1}.
\end{aligned}$$

2. $$\begin{aligned}
SEM(sym(t)) &=_{obs.2.1} SEM_{E(sym(t))}(union(t, inv(t))) \\
&=_{def} SEM(union(t, inv(t))) =_{obs.4.1} SEM(t) \cup SEM(inv(t)) \\
&=_{1.} SEM(t) \cup SEM(t)^{-1}.
\end{aligned}$$

3. $$\begin{aligned}
SEM(equiv(t)) &=_{obs.2.1} SEM_{E(equiv(t))}(trans(refl(sym(t)))) \\
&=_{def} SEM(trans(refl(sym(t)))) =_{obs.4.1} SEM(refl(sym(t)))^+ \\
&=_{obs.4.1} (SEM(sym(t)) \cup \Delta\mathscr{G})^+ =_{2.} (SEM(t) \cup SEM(t)^{-1} \cup \Delta\mathscr{G})^+ \\
&=_{def} equiv(SEM(t)).
\end{aligned}$$

This completes the proof. □

If t contains as control condition a boolean expression over the class CFG of context-free grammars over ID, we can explicitly construct a control condition for $inv(t)$ which fulfills the condition in Observation 4.3.

Observation 4.4. Let $\mathscr{A} = (\mathscr{G}, \mathscr{R}, \Rightarrow, \mathscr{E}, \mathscr{C})$ be a graph transformation approach such that $CFG \subseteq \mathscr{C}$ and for each $r \in \mathscr{R}$, there is an $r^{-1} \in \mathscr{R}$ with $(\Rightarrow_r)^{-1} = \Rightarrow_{r^{-1}}$. Let $t \in \mathscr{T}_{\mathscr{A}}$ and let $C \in \mathscr{B}(CFG)$. Then a control condition $C^{-1} \in \mathscr{B}(CFG)$ can be constructed from C such that $SEM_{E(inv(t))}(C^{-1}) = SEM_{E(t)}(C)^{-1}$.

Proof. Let $f: ID \to ID$ be defined as follows.

$$f(x) = \begin{cases} x^{-1} \text{ if } x \in R_t \\ inv(x) \text{ if } x \in U_t \\ x, \text{ otherwise} \end{cases}$$

For $L \subseteq ID^*$, let $L^{-1} = \{w^{-1} \mid w \in L\}$, where $\lambda^{-1} = \lambda$ and $aw^{-1} = w^{-1}a$ for all $a \in ID$ and all $w \in ID^*$. For $C \in CFG$, let C^{-1} be a context-free grammar such that $L(C^{-1}) = F(L(C))^{-1}$.[12] (Note that C^{-1} can be constructed in a straightforward way.) Then the following claim holds:

Claim Let $C \in CFG$ and let $t \in \mathscr{T}_{\mathscr{A}}$. Then $SEM_{E(t)}(C)^{-1} = SEM_{E(inv(t))}(C^{-1})$.

Proof of the claim. By induction on the length of strings we get that $\overline{E(t)}(w)^{-1} = \overline{E(inv(t))}(f^*(w^{-1}))$ for all $w \in ID^*$ as follows:

[12] For a grammar C, $L(C)$ denotes its generated language. For a function $f: ID \to ID$, $F: 2^{ID^*} \to 2^{ID^*}$ denotes its extension to languages, i.e. $F(L) = \{f^*(w) \mid w \in L\}$ for each $L \subseteq ID^*$ where $f^*: ID^* \to ID^*$ is the natural extension of f to strings, i.e. $f^*(\lambda) = \lambda$ and $f^*(aw) = f(a)f^*(w)$ for each $a \in ID$, $w \in ID^*$.

$$\overline{E(t)}(\lambda)^{-1} = \Delta_{\mathcal{G}} =_{def} \overline{E(inv(t))}(\lambda) =_{def} \overline{E(inv(t))}(f^*(\lambda^{-1})).$$

Moreover, for each $a \in ID$, we get by definition of f and $inv(t)$, that $E(t)(a)^{-1} = E(inv(t))(f(a))$. Hence,

$$\begin{aligned}
\overline{E(t)}(aw)^{-1} &=_{def} (E(t)(a) \circ \overline{E(t)}(w))^{-1} = \overline{E(t)}(w)^{-1} \circ E(t)(a)^{-1} \\
&= \overline{E(t)}(w)^{-1} \circ E(inv(t))(f(a)) \\
&=_{ind} \overline{E(inv(t))}(f^*(w^{-1})) \circ E(inv(t))(f(a)) \\
&= \overline{E(inv(t))}(f^*(w^{-1})f(a)) =_{def} \overline{E(inv(t))}(f^*(w^{-1}a)) \\
&=_{def} \overline{E(inv(t))}(f^*(aw^{-1})).
\end{aligned}$$

Hence,

$$\begin{aligned}
SEM_{E(t)}(C)^{-1} &=_{def} \widehat{\overline{E(t)}}(L(C))^{-1} =_{def} (\bigcup_{w \in L(C)} \overline{E(t)}(w))^{-1} \\
&= \bigcup_{w \in L(C)} \overline{E(t)}(w)^{-1} = \bigcup_{w \in L(C)} \overline{E(inv(t))}(f^*(w^{-1})) \\
&=_{def} \bigcup_{w \in F(L(C))} \overline{E(inv(t))}(w^{-1}) =_{def} \bigcup_{w \in F(L(C))^{-1}} \overline{E(inv(t))}(w) \\
&=_{def} \widehat{\overline{E(inv(t))}}(L(C^{-1})) =_{def} SEM_{E(inv(t))}(C^{-1}).
\end{aligned}$$

Let $true^{-1} = true$, and for all $e, e_1, e_2 \in \mathcal{B}(CFG)$ let $(e_1 \vee e_2)^{-1} = e_1^{-1} \vee e_2^{-1}$, $(e_1 \wedge e_2)^{-1} = e_1^{-1} \wedge e_2^{-1}$, and $(\overline{e})^{-1} = \overline{e^{-1}}$. Then by induction, $C^{-1} \in \mathcal{B}(CFG)$ for each $C \in \mathcal{B}(CFG)$.

We now show that $SEM_{E(t)}(C)^{-1} = SEM_{E(inv(t))}(C^{-1})$ if $C \in \mathcal{B}(CFG)$.

- If $C_t = true$ the statement obviously holds.
- If $C \in CFG$ then by the claim $SEM_{E(t)}(C)^{-1} = SEM_{E(inv(t)}(C^{-1})$.
- Assume that the statement holds for $e_1, e_2, e \in \mathcal{B}(CFG)$. Then

1. $$\begin{aligned}
   SEM_{E(t)}(e_1 \vee e_2)^{-1} &=_{def} (SEM_{E(t)}(e_1) \cup SEM_{E(t)}(e_2))^{-1} \\
   &= SEM_{E(t)}(e_1)^{-1} \cup SEM_{E(t)}(e_2)^{-1} \\
   &=_{ind} SEM_{E(inv(t))}(e_1^{-1}) \cup SEM_{E(inv(t))}(e_2^{-1}) \\
   &=_{def} SEM_{E(inv(t))}(e_1^{-1} \vee e_2^{-1}) = SEM_{E(inv(t))}((e_1 \vee e_2)^{-1}).
   \end{aligned}$$
2. The proof of $SEM_{E(t)}(e_1 \wedge e_2)^{-1} = SEM_{E(inv(t))}((e_1 \wedge e_2)^{-1})$ is analogous to that in point 1.
3. $$\begin{aligned}
   SEM_{E(t)}(\overline{e})^{-1} &=_{def} (\mathcal{G} \times \mathcal{G} - SEM_{E(t)}(e))^{-1} = \mathcal{G} \times \mathcal{G} - SEM_{E(t)}(e)^{-1} \\
   &=_{ind} \mathcal{G} \times \mathcal{G} - SEM_{E(inv(t))}(e^{-1}) = SEM_{E(inv(t))}(\overline{e^{-1}}) \\
   &= SEM_{E(inv(t))}(\overline{e}^{-1}).
   \end{aligned}$$

□

5. Normal Forms of Transformation Units

In this section, we present three 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. In all three cases, the control conditions cause some trouble (in the same way as in Subsection 4.2). The problem is in each case that we cannot give suitable conditions explicitly in the general situation. Hence, we only assume their existence in a first step, and give some sufficient constructions in a second step.

5.1. Encapsulating Rules

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

Based on this fact, the application of a rule $r \in \mathscr{R}$ corresponds to the call of *encapsulate*(r). Hence, for each $t \in \mathscr{T}_{\mathscr{A}}$, one can construct a new transformation unit $t' \in \mathscr{T}_{\mathscr{A}}$ 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.

Observation 5.1. Let $\mathscr{A} = (\mathscr{G}, \mathscr{R}, \Rightarrow, \mathscr{E}, \mathscr{C})$ be a graph transformation approach and let $t \in \mathscr{T}_{\mathscr{A}}$. Let $disperse(t) \in \mathscr{T}_{\mathscr{A}}$ be defined as follows:

$disperse(t)$	
initial:	I_t
uses:	$\{encapsulate(r) \mid r \in R_t\} \cup \{disperse(t') \mid t' \in U_t\}$
conds:	C
terminal:	T_t

where C is some control condition with $SEM_{E(disperse(t))}(C) = SEM_{E(t)}(C_t)$. Then $SEM(disperse(t)) = SEM(t)$.

Proof. By definition of $I_{disperse(t)}$, $T_{disperse(t)}$ and $C_{disperse(t)}$, the observation holds if the relations given by the interleaving sequences of $disperse(t)$ and t are equal i.e., $RIS_{disperse(t)} = RIS_t$.

If $U_t = \{\}$ then $RIS_{disperse(t)} =_{def} (\bigcup_{r \in R_t} SEM(encapsulate(r)))^* = \Rightarrow_{R_t}^* =_{def} RIS_t$. Assume inductively that for all $t' \in U_t$, $RIS_{disperse(t')} = RIS_{t'}$ (which implies that $SEM(disperse(t')) = SEM(t')$). Then we get

$$\begin{aligned} RIS_{disperse(t)} &=_{def} (\textstyle\bigcup_{t' \in U_{disperse(t)}} SEM(t'))^* \\ &=_{def} (\textstyle\bigcup_{r \in R_t} SEM(encapsulate(r)) \cup \bigcup_{t' \in U_t} SEM(disperse(t')))^* \\ &= (\Rightarrow_{R_t} \cup \textstyle\bigcup_{t' \in U_t} SEM(disperse(t')))^* \\ &=_{ind} (\Rightarrow_{R_t} \cup \textstyle\bigcup_{t' \in U_t} SEM(t'))^* =_{def} RIS_t \end{aligned}$$

Hence, $SEM(disperse(t)) = SEM(t)$. □

If t has as control condition a boolean expression over the class $GT0$ of all grammars of type 0 over ID, we can explicitly construct a control condition for *disperse*(t) which fulfills the condition in observation 5.1.

Observation 5.2. Let $\mathscr{A} = (\mathscr{G}, \mathscr{R}, \Rightarrow, \mathscr{E}, \mathscr{C})$ be a graph transformation approach such that $GT0 \subseteq \mathscr{C}$. Let $t \in \mathscr{T}_{\mathscr{A}}$ and let $C \in \mathscr{B}(GT0)$. Then a control condition $disp(C) \in \mathscr{B}(GT0)$ can be constructed from C such that

$$SEM_{E(disperse(t))}(disp(C)) = SEM_{E(t)}(C).$$

Proof. Let $f: ID \to ID$ be defined as follows:

$$f(x) = \begin{cases} encapsulate(x) \text{ if } x \in R_t \\ disperse(x) \text{ if } x \in U_t \\ x, \text{ otherwise} \end{cases}$$

For $C \in GT0$, let $disp(C)$ be a grammar generating the language $F(L(C))$.[13] Then the following claim holds:

Claim Let $C \in GT0$ and let $t \in \mathscr{T}_{\mathscr{A}}$. Then

$$SEM_{E(t)}(C) = SEM_{E(disperse(t))}(disp(C)).$$

Proof of the claim. By induction on the length of strings we get that $\overline{E(t)}(w) = \overline{E(disperse(t))}(f^*(w))$ for all $w \in ID^*$:

$$\begin{aligned}&\overline{E(t)}(\lambda) =_{def} \Delta_{\mathscr{G}} =_{def} \overline{E(disperse(t))}(\lambda)\\ &=_{def} \overline{E(disperse(t))}(f^*(\lambda)).\end{aligned}$$

Moreover, for $a \in ID$,

$$\begin{aligned}&\overline{E(t)}(aw) =_{def} E(t)(a) \circ \overline{E(t)}(w) = E(disperse(t))(f(a)) \circ \overline{E(t)}(w)\\ &\quad =_{ind} E(disperse(t))(f(a)) \circ \overline{E(disperse(t))}(f^*(w))\\ &\quad =_{def} \overline{E(disperse(t))}(f(a)f^*(w)) =_{def} \overline{E(disperse(t))}(f^*(aw)).\end{aligned}$$

Hence,

$$\begin{aligned}&SEM_{E(t)}(C) =_{def} \widehat{E(t)}(L(C)) =_{def} \bigcup_{w \in L(C)} \overline{E(t)}(w))\\ &\quad = \bigcup_{w \in L(C)} \overline{E(disperse(t))}(f^*(w)) =_{def} \bigcup_{w \in F(L(C))} \overline{E(disperse(t))}(w)\\ &\quad =_{def} \widehat{E(disperse(t))}(F(L(C))) =_{def} SEM_{E(disperse(t))}(disp(C)).\end{aligned}$$

Let $disp(true) = true$ and for all $e, e_1, e_2 \in \mathscr{B}(GT0)$ let $disp(e_1 \vee e_2) = disp(e_1) \vee disp(e_2)$; $disp(e_1 \wedge e_2) = disp(e_1) \wedge disp(e_2)$; and $disp(\bar{e}) = \overline{disp(e)}$. Then by induction, $disp(C) \in \mathscr{B}(GT0)$ for all $C \in \mathscr{B}(GT0)$.

We now show that for each $C \in \mathscr{B}(GT0)$,

$$SEM_{E(t)}(C) = SEM_{E(disperse(t))}(disp(C)).$$

- If the $C = true$ the statement obviously holds.
- If $C \in GT0$ then by the claim $SEM_{E(t)}(C) = SEM_{E(disperse(t))}(disp(C))$.
- Assume that the statement holds for $e_1, e_2, e \in \mathscr{B}(T_0 G)$. Then

1. $$\begin{aligned}&SEM_{E(t)}(e_1 \vee e_2) =_{def} SEM_{E(t)}(e_1) \cup SEM_{E(t)}(e_2)\\ &\quad =_{ind} SEM_{E(disperse(t))}(disp(e_1)) \cup SEM_{E(disperse(t))}(disp(e_2))\\ &\quad =_{def} SEM_{E(disperse(t))}(disp(e_1) \vee disp(e_2))\\ &\quad =_{def} SEM_{E(disperse(t))}(disp(e_1 \vee e_2))\end{aligned}$$
2. The proof of $SEM_{E(t)}(e_1 \wedge e_2) = SEM_{E(disperse(t))}(disp(e_1 \wedge e_2))$ is analogous to that in point 1.
3. $$\begin{aligned}&SEM_{E(t)}(\bar{e}) =_{def} (\mathscr{G} \times \mathscr{G} - SEM_{E(t)}(e))\\ &\quad =_{ind} \mathscr{G} \times \mathscr{G} - SEM_{E(disperse(t))}(disp(e))\\ &\quad =_{def} SEM_{E(disperse(t))}(\overline{disp(e)}).\end{aligned}$$

□

5.2. Handling Graph Class Expressions as Control Conditions

As pointed out in 2.5, two graph class expressions form a control condition. Hence, the specifications of initial and terminal graphs may be handled as a

[13] Note that $disp(C)$ can always be constructed.

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 leads to the following observation.

Observation 5.3. Let t be a transformation unit over $\mathscr{A} = (\mathscr{G}, \mathscr{R}, \Rightarrow, \mathscr{E}, \mathscr{C})$ with $\mathscr{E} \times \mathscr{E} \subseteq \mathscr{C}$ (as defined in 2.5.8). Let $rem\_expr(t)$ be defined as follows:

$rem\_expr(t)$
uses: $\{rem\_expr(t') \mid t' \in U_t\}$
rules: R_t
conds: $C \wedge (I_t, T_t)$

where C is some control condition with $SEM_{E(rem\_expr(t))}(C) = SEM_{E(t)}(C_t)$. Then $SEM(t) = SEM(rem\_expr(t))$.

Proof. Since $SEM_{E(rem\_expr(t))}(C) = SEM_{E(t)}(C_t)$, we get that

$$\begin{aligned} &SEM_{E(t)}(C_t) \cap (SEM(I_t) \times SEM(T_t)) \\ &\quad = SEM_{E(rem\_expr(t))}(C) \cap SEM(I_t, T_t) \cap (\mathscr{G} \times \mathscr{G}) \\ &\quad =_{def} SEM_{E(rem\_expr(t))}(C \wedge (I_t, T_t)) \cap (SEM(all) \times SEM(all)) \end{aligned}$$

Hence, it remains to show that $RIS_t = RIS_{rem\_expr(t)}$. If $U_t = \{\}$ then by definition $RIS_t = \Rightarrow_{R_t}^* = RIS_{rem\_expr(t)}$. Assume inductively that for all $t' \in U_t$, $RIS_{t'} = RIS_{rem\_expr(t')}$. Since this implies that $SEM(t') = SEM(rem\_expr(t'))$ we get

$$\begin{aligned} RIS_t &=_{def} (\Rightarrow_{R_t} \cup \bigcup_{t' \in U_t} SEM(t'))^* \\ &=_{ind} (\Rightarrow_{R_t} \cup \bigcup_{t' \in U_t} SEM(rem\_expr(t'))^* \\ &=_{def} (\Rightarrow_{R_{rem\_expr(t)}} \cup \bigcup_{t' \in U_{rem\_expr(t)}} SEM(t'))^* =_{def} RIS_{rem\_expr(t)}. \end{aligned}$$

□

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.

If t contains a control condition in $\mathscr{B}(GT0)$ we can construct a control condition for $rem\_expr(t)$ fulfilling the condition in Observation 5.3.

Observation 5.4. Let $\mathscr{A} = (\mathscr{G}, \mathscr{R}, \Rightarrow, \mathscr{E}, \mathscr{C})$ be a graph transformation approach with $GT0 \subseteq \mathscr{C}$. Let $t \in \mathscr{T}_{\mathscr{A}}$ and let $C \in \mathscr{B}(GT0)$. Then a control condition $C' \in \mathscr{B}(GT0)$ can be constructed from C such that

$$SEM_{E(t)}(C) = SEM_{E(rem\_expr(t))}(C').$$

The proof is analogous to that of Observation 5.2 choosing $f: ID \to ID$ with

$$f(x) = \begin{cases} rem\_expr(x) \text{ if } x \in U_t \\ x, \text{ otherwise.} \end{cases}$$

5.3. 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.

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 5.5. Let $\mathscr{A} = (\mathscr{G}, \mathscr{R}, \Rightarrow, \mathscr{E}, \mathscr{C})$ be a graph transformation approach and let $t \in \mathscr{T}_{\mathscr{A}}$ such that $SEM_{E(t)}(C_t) \subseteq RIS_t$. Let $\mathit{flatten}(t) \in \mathscr{T}_{\mathscr{A}}$ be defined as follows:

$\mathit{flatten}(t)$
initial: I_t
rules: $R_t \cup (\bigcup_{t' \in FLAT(t)} R_{t'})$
conds: C
terminal: T_t

where C is some control condition with $SEM_{E(\mathit{flatten}(t))}(C) = SEM_{E(t)}(C_t)$.

Then $SEM(\mathit{flatten}(t)) = SEM(t)$.

Proof. The assumption $SEM_{E(t)}(C_t) \subseteq RIS_t$ implies that

$$\begin{aligned} SEM(t) &= (SEM(I_t) \times SEM(T_t)) \cap SEM_{E(t)}(C_t) \\ &=_{def} (SEM(I_{\mathit{flatten}(t)}) \times SEM(T_{\mathit{flattent}(t)})) \cap SEM_{E(\mathit{flattent}(t))}(C_{\mathit{flatten}(t)}) \\ &\supseteq_{def} SEM(\mathit{flatten}(t)). \end{aligned}$$

To show that $SEM(t) \subseteq SEM(\mathit{flatten}(t))$ it is sufficient to prove that $RIS_t \subseteq RIS_{\mathit{flatten}(t)}$, because then

$$\begin{aligned} SEM(\mathit{flatten}(t)) &=_{def} (SEM(I_{\mathit{flatten}(t)}) \times SEM(T_{\mathit{flattent}(t)})) \cap \\ &\quad SEM_{E(\mathit{flattent}(t))}(C_{\mathit{flatten}(t)}) \cap RIS_{\mathit{flatten}(t)} \\ &=_{def} (SEM(I_t) \times SEM(T_t)) \cap SEM_{E(t)}(C_t) \cap RIS_{\mathit{flatten}(t)} \\ &\supseteq (SEM(I_t) \times SEM(T_t)) \cap SEM_{E(t)}(C_t) \cap RIS_t =_{def} SEM(t). \end{aligned}$$

If $U_t = \{\}$, then $RIS_t \subseteq RIS_{\mathit{flatten}(t)}$. Assume that for each $t' \in U_t$, $RIS_{t'} \subseteq RIS_{\mathit{flatten}(t')}$, which implies that $SEM(t') \subseteq SEM(\mathit{flatten}(t'))$. Then

$$\begin{aligned} RIS_t &=_{def} (R_t \cup \textstyle\bigcup_{t' \in U_t} SEM(t'))^* \subseteq_{ind} (R_t \cup \bigcup_{t' \in U_t} SEM(\mathit{flatten}(t')))^* \\ &\subseteq_{def} ((R_t \cup \textstyle\bigcup_{t' \in U_t} RIS_{\mathit{flatten}(t')}))^* \\ &=_{def} (R_t \cup \textstyle\bigcup_{t' \in U_t} (R_{t'} \cup \bigcup_{t'' \in FLAT(t')} R_{t''}))^* \\ &= (R_t \cup \textstyle\bigcup_{t' \in U_t} R_{t'} \cup \bigcup_{t' \in U_t} \bigcup_{t'' \in FLAT(t')} R_{t''})^* \\ &= (R_t \cup \textstyle\bigcup_{t' \in U_t} R_{t'} \cup \bigcup_{t'' \in \bigcup_{t' \in U_t} FLAT(t')} R_{t''})^* \\ &= (R_t \cup \textstyle\bigcup_{t' \in U_t} R_{t'} \cup \bigcup_{t' \in \bigcup_{t'' \in U_t} FLAT(t'')} R_{t'})^* \\ &= (R_t \cup \textstyle\bigcup_{t' \in (U_t \cup \bigcup_{t'' \in U_t} FLAT(t''))} R_{t'})^* = (R_t \cup \bigcup_{t' \in FLAT(t)} R_{t'})^* \\ &=_{def} RIS_{\mathit{flatten}(t)}. \end{aligned}$$

□

If each control condition occurring in t or in a transformation unit of $FLAT(t)$ is a context-free grammar in CFG, and if additionally, the initial and terminal expressions of each transformation unit $t' \in FLAT(t)$ can be omitted without changing the interleaving semantics of t', we can construct a context-free grammar for $\mathit{flatten}(t)$ which fulfills the condition in Observation 5.5. Moreover, in this case, the control condition of t allows only pairs of graphs which are in the relation given by the interleaving sequences.

Observation 5.6. Let $\mathscr{A} = (\mathscr{G}, \mathscr{R}, \Rightarrow, \mathscr{E}, \mathscr{C})$ be a graph transformation approach with $CFG \subseteq \mathscr{C}$. Let $t \in \mathscr{T}_{\mathscr{A}}$ such that for each $t' \in \{t\} \cup FLAT(t)$, $C_{t'} \in CFG$, and for each $t' \in FLAT(t)$, $SEM(I_{t'}) \times SEM(T_{t'}) \supseteq SEM_{E(t')}(C_{t'})$. Then

1. $SEM_{E(t)}(C_t) \subseteq RIS_t$, and
2. a control condition $\mathit{flat}(C_t) \in CFG$ can be constructed from C_t such that $SEM_{E(t)}(C_t) = SEM_{E(\mathit{flatten}(t))}(\mathit{flat}(C_t))$.

Proof. W.l.o.g. we assume that for each $t' \in \{t\} \cup FLAT(t)$, $C_{t'}$ is a context-free grammar over $R_{t'} \cup U_{t'}$. (Such a context-free grammar $C_{t'}$ can be obtained from a grammar $C \in CFG$ with disjoint non-terminal and terminal symbols by deleting each rule containing a terminal symbol in $ID - (R_{t'} \cup U_{t'})$. Then we have $L(C_{t'}) = L(C) - \{w \in ID^* \mid w \notin (R_{t'} \cup U_{t'})^*\}$. Moreover, since for all $w \notin (R_{t'} \cup U_{t'})^*$, $\overline{E(t')}(w) = \emptyset$, $SEM_{E(t')}(C_{t'}) = SEM_{E(t')}(C)$.)

1. Since $L(C_t) \subseteq (R_t \cup U_t)^*$ point 1 holds because of Observation 2.2.
2. Let $\phi : R_t \cup U_t \to 2^{(R_t \cup U_t)^*}$ be defined as follows:

$$\phi(x) = \begin{cases} \Phi(L(C_x)) \text{ if } x \in U_t \\ \{x\}, \text{ otherwise} \end{cases}$$

where for $L \subseteq (R_t \cup U_t)^*$, $\Phi(L) = \bigcup_{w \in L} \phi^*(w)$ and $\phi^* : (R_t \cup U_t)^* \to 2^{(R_t \cup U_t)^*}$ with $\phi^*(\lambda) = \{\lambda\}$ and $\phi^*(aw) = \phi(a)\phi^*(w)$ for each $a \in R_t \cup U_t$ and $w \in (R_t \cup U_t)^*$.[14] Note that ϕ is well-defined because of the finite recursion depth of t.
Let $\mathit{flat}(C_t) \in CFG$ such that $L(\mathit{flat}(C_t)) = \Phi(L(C_t))$. (Note that context-free languages are closed under substitution. Moreover, $\mathit{flat}(C_t)$ can be constructed if for all $x \in R_t \cup U_t$, $\phi(x)$ can be constructed. Hence, by induction on the recursion depth of t we get that $\mathit{flat}(C_t) \in CFG$ can be constructed.)
We now show that

$$SEM_{E(t)}(C_t) = SEM_{E(\mathit{flatten}(t))}(\mathit{flat}(C_t)).$$

If $U_t = \{\}$ then by definition $\Phi(L(C_t)) = L(C_t)$ and $E(t) = E(\mathit{flatten}(t))$. Hence, in this case, $SEM_{E(t)}(C_t) = SEM_{E(\mathit{flatten}(t))}(\mathit{flat}(C_t))$. Assume that the statement holds for all $t' \in U_t$. Then by induction on the length of $w \in (U_t \cup R_t)^*$, the equation $\overline{E(t)}(w) = \widehat{E(\mathit{flatten}(t))}(\phi^*(w))$ can be shown as follows:

$$\begin{aligned} \overline{E(t)}(\lambda) &=_{def} \Delta\mathscr{G} =_{def} \overline{E(\mathit{flatten}(t))}(\lambda) \\ &=_{def} \widehat{E(\mathit{flatten}(t))}(\{\lambda\}) =_{def} \widehat{E(\mathit{flatten}(t))}(\phi^*(\lambda)). \end{aligned}$$

If $a \in R_t$,

$$\begin{aligned} \overline{E(t)}(aw) &=_{def} E(t)(a) \circ \overline{E(t)}(w) \\ &=_{def} E(\mathit{flatten}(t))(a) \circ \overline{E(t)}(w) \\ &= \widehat{E(\mathit{flatten}(t))}(\phi(a)) \circ \overline{E(t)}(w) \\ &=_{ind} \widehat{E(\mathit{flatten}(t))}(\phi(a)) \circ \widehat{E(\mathit{flatten}(t))}(\phi^*(w)) \\ &= \widehat{E(\mathit{flatten}(t))}(\phi(a)\phi^*(w)) \\ &=_{def} \widehat{E(\mathit{flatten}(t))}(\phi^*(aw)). \end{aligned}$$

[14] The concatenation of sets L_1 and L_2 is denoted by $L_1 L_2$.

If $a \in U_t$,

$$\begin{aligned}
\overline{E(t)}(aw) &=_{def} E(t)(a) \circ \overline{E(t)}(w) = SEM(a) \circ \overline{E(t)}(w) \\
&=_{obs.2.2} (SEM_{E(a)}(C_a) \cap (SEM(I_a) \times SEM(T_a))) \circ \overline{E(t)}(w) \\
&= SEM_{E(a)}(C_a) \circ \overline{E(t)}(w) =_{ind} SEM_{E(flatten(a))}(flat(C_a)) \circ \overline{E(t)}(w) \\
&=_{def} \widehat{E(flatten(a))}(L(flat(C_a)) \circ \overline{E(t)}(w) \\
&=_{def} \widehat{E(flatten(a))}(\Phi(L(C_a))) \circ \overline{E(t)}(w) \\
&=_{def} \widehat{E(flatten(a))}(\phi(a)) \circ \overline{E(t)}(w) =_{def} \widehat{E(flatten(t))}(\phi(a)) \circ \overline{E(t)}(w) \\
&=_{ind} \widehat{E(flatten(t))}(\phi(a)) \circ \widehat{E(flatten(t))}(\phi^*(w)) \\
&= \widehat{E(flatten(t))}(\phi(a)\phi^*(w)) =_{def} \widehat{E(flatten(t))}(\phi^*(aw)).
\end{aligned}$$

Hence, it follows that

$$\begin{aligned}
SEM_{E(t)}(C_t) &=_{def} \widehat{\overline{E(t)}}(L(C_t)) =_{def} \bigcup_{w \in L(C_t)} \overline{E(t)}(w) \\
&= \bigcup_{w \in L(C_t)} \widehat{E(flatten(t))}(\phi^*(w)) = \bigcup_{w \in \Phi(L(C_t))} \overline{E(flatten(t))}(w) \\
&=_{def} \widehat{E(flatten(t))}(\Phi(L(C_t))) =_{def} SEM_{E(flatten(t))}(flat(C_t)).
\end{aligned}$$

□

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 researchers from Berlin, Bremen, Erlangen, München, Oldenburg, and Paderborn. 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, possibly by using other binary relations. 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 [EhE96]), by Taentzer and Schürr (see [TaS95]), or the notion of a transaction introduced by Schürr and Zündorf in the framework of PROGRES (see [ScZ91]).

Further aspects of the structured development of graph transformation systems based on transformation units are studied in [KKS97, KrK99].

Acknowledgement. We are grateful to Frank Drewes for many helpful suggestions, in particular concerning the environment concept in the semantics of control conditions, and to our friends in the GRACE initiative – in particular, Ralf Betschko, Hartmut Ehrig, Gregor Engels, Annegret Habel, Reiko Heckel, Berthold Hoffmann, Peter Knirsch, Jürgen Müller, Detlef Plump, Andy Schürr, and Gabriele Taentzer – for fruitful discussions on the subject of this paper.

References

[AEH99] Marc Andries, Gregor Engels, Annegret Habel, Berthold Hoffmann, Hans-Jörg Kreowski, Sabine Kuske, Detlef Plump, Andy Schürr, and Gabriele Taentzer. Graph transformation for specification and programming. *Science of Computer Programming*, 34(1):1–54, 1999.

[Bun79] Horst Bunke. Programmed graph grammars. In Volker Claus, Hartmut Ehrig, and Grzegorz Rozenberg, editors, *Proc. Graph Grammars and Their Application to Computer Science and Biology*, volume 73 of *Lecture Notes in Computer Science*, pages 155–166, 1979.

[Cou90] Bruno Courcelle. Graph rewriting: An algebraic and logical approach. In Jan van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, pages 193–242. Elsevier, Amsterdam, 1990.

[DaP89] Jürgen Dassow and Gheorghe Păun. *Regulated Rewriting in Formal Language Theory*, volume 18 of *EATCS Monographs on Theoretical Computer Science*. Springer-Verlag, 1989.

[EhE96] Hartmut Ehrig and Gregor Engels. Pragmatic and semantic aspects of a module concept for graph transformation systems. In Janice E. Cuny, Hartmut Ehrig, Gregor Engels, and Grzegorz Rozenberg, editors, *Proc. Graph Grammars and Their Application to Computer Science*, volume 1073 of *Lecture Notes in Computer Science*, pages 137–154, 1996.

[EEK99] Hartmut Ehrig, Gregor Engels, Hans-Jörg Kreowski, and Grzegorz Rozenberg, editors. *Handbook of Graph Grammars and Computing by Graph Transformation, Vol. 2: Applications, Languages and Tools*. World Scientific, Singapore, 1999.

[EhH86] Hartmut Ehrig and Annegret Habel. Graph grammars with application conditions. In Grzegorz Rozenberg and Arto Salomaa, editors, *The Book of L*, pages 87–100. Springer-Verlag, Berlin, 1986.

[Ehr79] Hartmut Ehrig. Introduction to the algebraic theory of graph grammars. In Volker Claus, Hartmut Ehrig, and Grzegorz Rozenberg, editors, *Proc. Graph Grammars and Their Application to Computer Science and Biology*, volume 73 of *Lecture Notes in Computer Science*, pages 1–69, 1979.

[EKM99] Hartmut Ehrig, Hans-Jörg Kreowski, Ugo Montanari, and Grzegorz Rozenberg, editors. *Handbook of Graph Grammars and Computing by Graph Transformation, Vol. 3: Concurrency, Parallelism, and Distribution*. World Scientific, Singapore, 1999.

[GKS91] John R. W. Glauert, J. Richard Kennaway, and M. Ronan Sleep. DACTL: An experimental graph rewriting language. In Hartmut Ehrig, Hans-Jörg Kreowski, and Grzegorz Rozenberg, editors, *Proc. Graph Grammars and Their Application to Computer Science*, volume 532 of *Lecture Notes in Computer Science*, pages 378–395, 1991.

[Hab92] Annegret Habel. *Hyperedge Replacement: Grammars and Languages*, volume 643 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, 1992.

[HHT96] Annegret Habel, Reiko Heckel, and Gabriele Taentzer. Graph grammars with negative application conditions. *Fundamenta Informaticae*, 26(3,4):287–313, 1996.

[Him91] Michael Himsolt. Graph^{ed}: An interactive tool for developing graph grammars. In Hartmut Ehrig, Hans-Jörg Kreowski, and Grzegorz Rozenberg, editors, *Proc. Graph Grammars and Their Application to Computer Science*, volume 532 of *Lecture Notes in Computer Science*, pages 61–65, 1991.

[JaR80] Dirk Janssens and Grzegorz Rozenberg. On the structure of node-label-controlled graph languages. *Information Sciences*, 20:191–216, 1980.

[KrK96] Hans-Jörg Kreowski and Sabine Kuske. On the interleaving semantics of transformation units — a step into GRACE. In Janice E. Cuny, Hartmut Ehrig, Gregor Engels, and Grzegorz Rozenberg, editors, *Proc. Graph Grammars and Their Application to Computer Science*, volume 1073 of *Lecture Notes in Computer Science*, pages 89–108, 1996.

[KrK99] Hans-Jörg Kreowski and Sabine Kuske. Graph transformation units and modules. In Ehrig et al. [EEK99], pages 607–638.

[KKS97] Hans-Jörg Kreowski, Sabine Kuske, and Andy Schürr. Nested graph transformation units. *International Journal on Software Engineering and Knowledge Engineering*, 7(4):479–502, 1997.

[KrR90] Hans-Jörg Kreowski and Grzegorz Rozenberg. On structured graph grammars, I and II. *Information Sciences*, 52:185–210 and 221–246, 1990.

[Kre93] Hans-Jörg Kreowski. Five facets of hyperedge replacement beyond context-freeness. In Zoltán Ésik, editor, *Proc. Fundamentals of Computation Theory*, volume 710 of *Lecture Notes in Computer Science*, pages 69–86, 1993.

[LöB93] Michael Löwe and Martin Beyer. AGG — an implementation of algebraic graph rewriting. In Claude Kirchner, editor, *Proc. Rewriting Techniques and Applications*, volume 690 of *Lecture Notes in Computer Science*, pages 451–456, 1993.

[Len90] Thomas Lengauer. VLSI theory. In Jan van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume A. Elsevier Science Publishers B.V., 1990.

[LiM93] Igor Litovsky and Yves Métivier. Computing with graph rewriting systems with priorities. *Theoretical Computer Science*, 115:191–224, 1993.

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

[MaW91] Andrea Maggiolo-Schettini and Józef Winkowski. Programmed derivations of relational structures. In Hartmut Ehrig, Hans-Jörg Kreowski, and Grzegorz Rozenberg, editors, *Proc. Graph Grammars and Their Application to Computer Science*, volume 532 of *Lecture Notes in Computer Science*, pages 582–598, 1991.

[MaW96] Andrea Maggiolo-Schettini and Józef Winkowski. A kernel language for programmed rewriting of (hyper)graphs. *Acta Informatica*, 33(6):523–546, 1996.

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

[Roz97] Grzegorz Rozenberg, editor. *Handbook of Graph Grammars and Computing by Graph Transformation, Vol. 1: Foundations*. World Scientific, Singapore, 1997.

[Sch91a] Andy Schürr. *Operationales Spezifizieren mit programmierten Graphersetzungssystemen: formale Definitionen, Anwendungsbeispiele und Werkzeugunterstützung*. PhD thesis, Rheinisch-Westfälische Technische Hochschule Aachen, 1991.

[Sch91b] Andy Schürr. PROGRES: A VHL-language based on graph grammars. In Hartmut Ehrig, Hans-Jörg Kreowski, and Grzegorz Rozenberg, editors, *Proc. Graph Grammars and Their Application to Computer Science*, volume 532 of *Lecture Notes in Computer Science*, pages 641–659, 1991.

[Sch96] Andy Schürr. Programmed graph transformations and graph transformation units in GRACE. In Janice E. Cuny, Hartmut Ehrig, Gregor Engels, and Grzegorz Rozenberg, editors, *Proc. Graph Grammars and Their Application to Computer Science*, volume 1073 of *Lecture Notes in Computer Science*, pages 122–136, 1996.

[ScZ91] Andy Schürr and Albert Zündorf. Nondeterministic control structures for graph rewriting systems. In G. Schmidt and Rudolf Berghammer, editors, *Proc. Graph-Theoretic Concepts in Computer Science*, volume 570 of *Lecture Notes in Computer Science*, pages 48–62, 1991.

[TaB94] Gabriele Taentzer and Martin Beyer. Amalgamated graph transformation systems and their use for specifying AGG – an algebraic graph grammar system. In Hans-Jürgen Schneider and Hartmut Ehrig, editors, *Proc. Graph Transformations in Computer Science*, volume 776 of *Lecture Notes in Computer Science*, pages 380–394, 1994.

[TaS95] Gabriele Taentzer and Andy Schürr. DIEGO, another step towards a module concept for graph transformation systems. In A. Corradini and U. Montanari, editors, *SEGRAGRA'95, Joint COMPUGRAPH/SEMAGRAPH Workshop on Graph Rewriting and Computation*, volume 2 of *Electronic Notes in Theoretical Computer Science*. Elsevier, 1995.

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

Appendix

In this appendix, a detailed proof of the correctness theorem of Section 3 is presented. It proceeds along the structure of the transformation unit *buttterfly* and is mainly based on induction on the length of interleaving sequences.

W.l.o.g. we assume from now on that the node and edge set of each graph are disjoint, i.e. for each $G \in \mathcal{G}$, $E_G \cap V_G = \emptyset$.

Theorem A.1. Let $G, G' \in \mathcal{G}$ such that G is labelled over $\{*, b\}$. Then $(G, G') \in SEM(copy\_items)$ iff $G' \cong (V, E, l, m)$ where

$$V = V_G \uplus \{cp(v) \mid v \in V_G\} \quad [15]$$

$$E = E_G \cup \{\{cp(v_1), cp(v_2)\} \mid \{v_1, v_2\} \in E_G\} \cup \{\{v, cp(v)\} \mid v \in V_G\}$$

$$l(v) = \begin{cases} l_G(v), & v \in V_G \\ l_G(v'), & v \in \{cp(v') \mid v' \in V_G\} \end{cases} \qquad \text{for all } v \in V$$

$$m(e) = \begin{cases} c, & e \in \{\{v, cp(v)\} \mid v \in V_G\} \\ *, & \text{otherwise} \end{cases} \qquad \text{for all } e \in E.$$

Proof. The proof is based on two lemmas that are given and proved in the following.

Lemma 1. Let $G_0 \in \mathcal{G}$ be labelled over $\{*, b\}$, and let $G_0 \Rightarrow_R G_1 \Rightarrow_R \cdots \Rightarrow_R G_n$

[15] $\uplus$ denotes the disjoint union of sets.

($n \geqslant 0$) with $R = R_{copy\_items}$. Then there exist sets $\overline{V_0}, \ldots, \overline{V_n} \subseteq V_{G_0}, \overline{E_0}, \ldots, \overline{E_n} \subseteq E_{G_0}$ such that $\overline{V_0} \cup \overline{E_0} \subset \cdots \subset \overline{V_n} \cup \overline{E_n}$, and for $i = 0, \ldots, n$, $G_i \cong (V_i, E_i, l_i, m_i)$ with

$$V_i = V_{G_0} \uplus \{cp(v) \mid v \in \overline{V_i}\}$$

$$E_i = E_{G_0} \cup \{\{cp(v_1), cp(v_2)\} \mid \{v_1, v_2\} \in \overline{E_i}\} \cup \{\{v, cp(v)\} \mid v \in \overline{V_i}\}$$

$$l_i(v) = \begin{cases} l_{G_0}(v), & v \in V_{G_0} \\ l_{G_0}(v'), & v \in \{cp(v') \mid v' \in \overline{V_i}\} \end{cases} \qquad \text{for all } v \in V_i$$

$$m_i(e) = \begin{cases} c, & e \in \{\{v, cp(v)\} \mid v \in \overline{V_i}\} \\ *, & \text{otherwise} \end{cases} \qquad \text{for all } e \in E_i.$$

Proof of Lemma 1. We show Lemma 1 by induction on the length n of the derivation $G_0 \Rightarrow_R G_1 \Rightarrow_R \cdots \Rightarrow_R G_n$. If $n = 0$ the lemma obviously holds choosing $\overline{V_0} = \overline{E_0} = \emptyset$. Assume that it holds for all derivations $G_0 \Rightarrow_R G_1 \Rightarrow_R \cdots \Rightarrow_R G_n$. Let $d = (G_0 \Rightarrow_R G_1 \Rightarrow_R \cdots \Rightarrow_R G_n \Rightarrow_R G_{n+1})$. W.l.o.g. let $G_n = (V_n, E_n, l_n, m_n)$.

If a rule of r_1 is applied to G_n in d, then by definition there exists a node $v_0 \in V_n$ with $l_n(v_0) \in \{*, b\}$ such that there is no c-labelled edge $e \in E_n$ with $v_0 \in e$. Then by induction $v_0 \notin \overline{V_n} \cup \{cp(v) \mid v \in \overline{V_n}\}$, i.e. $v_0 \in V_{G_0} - \overline{V_n}$. By definition $G_{n+1} \cong (V, E, l, m)$ where $V = V_n \uplus \{cp(v_0)\}$, $E = E_n \cup \{\{v_0, cp(v_0)\}\}$, $l(v) = l_n(v)$ for all $v \in V_n$, $l(cp(v_0)) = l_n(v_0)$, $m(e) = m_n(e)$ for all $e \in E_n$, and $m(\{v_0, cp(v_0)\}) = c$. Choose $\overline{V_{n+1}} = \overline{V_n} \cup \{v_0\}$ and $\overline{E_{n+1}} = \overline{E_n}$. Then $\overline{V_n} \cup \overline{E_n} \subset \overline{V_{n+1}} \cup \overline{E_{n+1}}$. Moreover, by induction, we get that $\overline{V_{n+1}} \cup \overline{E_{n+1}} \subseteq V_{G_0} \cup E_{G_0}$ and that G_{n+1} is of the required form.

If a rule of r_2 is applied to G_n in d, there are nodes $v_1, v_2, v_3, v_4 \in V_n$ such that $\{v_1, v_2\}, \{v_1, v_3\}, \{v_2, v_4\} \in E_n$, $\{v_3, v_4\} \notin E_n$, $l_n(v_1) = l_n(v_3) \in \{*, b\}$, $l_n(v_2) = l_n(v_4) \in \{*, b\}$, $m_n(\{v_1, v_2\}) = *$, and $m_n(\{v_1, v_3\}) = m_n(\{v_2, v_4\}) = c$. Then by induction $\{v_1, v_2\} \in E_{G_0} - \overline{E_n}$ because $\{v_1, v_2\} \in \overline{E_n}$ implies $\{v_3, v_4\} \in E_n$, $\{v_1, v_2\} \in \{\{cp(v_j), cp(v_k)\} \mid \{v_j, v_k\} \in \overline{E_n}\}$ implies $\{v_i, v_j\} = \{v_3, v_4\}$ and hence $\{v_3, v_4\} \in E_n$, and $\{v_1, v_2\} \in \{\{v, cp(v)\} \mid v \in \overline{V_n}\}$ implies $m_n(\{v_1, v_2\}) = c$. Moreover, $G_{n+1} \cong (V, E, l, m)$ where $V = V_n$, $E = E_n \cup \{\{v_3, v_4\}\}$, $l = l_n$, $m(e) = m_n(e)$ for all $e \in E_n$, and $m(\{v_3, v_4\}) = *$. Choose $\overline{V_{n+1}} = \overline{V_n}$ and $\overline{E_{n+1}} = \overline{E_n} \cup \{\{v_1, v_2\}\}$. Then $\overline{V_n} \cup \overline{E_n} \subset \overline{V_{n+1}} \cup \overline{E_{n+1}}$. Moreover, by induction, we get that $\overline{V_{n+1}} \cup \overline{E_{n+1}} \subseteq V_{G_0} \cup E_{G_0}$ and that G_{n+1} is of the required form. This completes the proof of Lemma 1.

Lemma 2. Let $G_0 \in \mathcal{G}$ be a graph over $\{*, b\}$, and let $G_0 \Rightarrow_R G_1 \Rightarrow_R \cdots \Rightarrow_R G_n$ ($n \geqslant 0$) with $R = R_{copy\_items}$ and $G_n \in RED(R)$. Then $\overline{V_n} \cup \overline{E_n} = V_{G_0} \cup E_{G_0}$, where $\overline{V_n}, \overline{E_n}$ are defined as in the proof of Lemma 1.

Proof of Lemma 2. W.l.o.g. we assume that $G_n = (V_n, E_n, l_n, m_n)$ where V_n, E_n, l_n, and m_n are defined as in Lemma 1. From Lemma 1 we know that $\overline{V_n} \cup \overline{E_n} \subseteq V_{G_0} \cup E_{G_0}$. Assume that there is an $x \in (V_{G_0} \cup E_{G_0}) - (\overline{V_n} \cup \overline{E_n})$. If x is a node, then from Lemma 1 we get that there is no c-labelled edge e in G_n with $x \in e$. Hence, a rule of r_1 is applicable to G_n. Now assume $V_n = V_{G_0}$, so x is an edge, say $\{v_1, v_2\}$, and $v_1, v_2 \in \overline{V_n}$. Then from Lemma 1 we get that $m_n(\{v_1, v_2\}) = *$, and $l_n(v_1), l_n(v_2) \in \{*, b\}$. Moreover, there are c-labelled edges $\{v_1, cp(v_1)\}$, $\{v_2, cp(v_2)\}$ in G_n with $l_n(v_1) = l_n(cp(v_1))$, $l_n(v_2) = l_n(cp(v_2))$, and $cp(v_1)$ and $cp(v_2)$ are not adjacent in G_n. In this case, a rule of r_2 is applicable to G_n. Hence, $G_n \in RED(R)$ implies that $\overline{V_n} \cup \overline{E_n} = V_{G_0} \cup E_{G_0}$. This completes the proof of Lemma 2.

1. Now let $G, G' \in \mathscr{G}$ such that $(G, G') \in SEM(copy\_items)$ and G is a graph over $\{*, b\}$. Then by definition there is a derivation $G_0 \Rightarrow_R \cdots \Rightarrow_R G_n$ ($n \geqslant 0, R = R_{copy\_items}$) such that $G_0 = G$, $G_n = G'$, and $G_n \in RED(R)$. Let $\overline{V_n}, \overline{E_n}$ be defined as in the proof of Lemma 1. Then from Lemma 2 and the disjointness of V_{G_0} and E_{G_0} we get $\overline{V_n} = V_{G_0}$, $\overline{E_n} = E_{G_0}$ and by Lemma 1 G' is of the required form.
2. Conversely, let $G, G' \in \mathscr{G}$ such that G is a graph over $\{*, b\}$ and G' is defined as in the theorem. Since G is finite, we get from Lemma 1 that there is a derivation $G \Rightarrow_R^* G''$ such that $G'' \in RED(R)$. Then, as shown in point 1, $G'' \cong G'$.

This completes the proof. □

Theorem A.2. Let $G, G' \in \mathscr{G}$. Then $(G, G') \in SEM(delete)$ if and only if $G' \cong (V_G, E, l_G, m)$ where $E = E_G - \{\{v, v'\} \in E_G \mid m_G(\{v, v'\}) = c,\ l_G(v) = l_G(v') = *\}$, and $m = m_G|_E$.

Proof.

1. If $(G, G') \in SEM(delete)$ then by definition there is a derivation $G_0 \Rightarrow_r \cdots \Rightarrow_r G_n$ ($n \geqslant 0, \{r\} = R_{delete}$) such that $G_0 = G$, $G_n = G'$, and $G_n \in RED(\{r\})$. By definition of r, for $i = 1, \ldots, n$, there is a c-labelled edge $\{v_{i_1}, v_{i_2}\} \in E_{G_{i-1}}$ such that $l_{G_{i-1}}(v_{i_1}) = l_{G_{i-1}}(v_{i_2}) = *$ and $G_i \cong (V_{G_{i-1}}, E_i, l_{G_{i-1}}, m_i)$ where $E_i = E_{G_{i-1}} - \{\{v_{i_1}, v_{i_2}\}\}$, and $m_i = m_{G_{i-1}}|_{E_i}$. W.l.o.g. assume that $G_i = (V_{G_{i-1}}, E_i, l_{G_{i-1}}, m_i)$. Then by induction $G_n = (V_{G_0}, E, l_{G_0}, m)$ where $E = E_{G_0} - \{\{v_{1_1}, v_{1_2}\}, \ldots, \{v_{n_1}, v_{n_2}\}\}$, and $m = m_{G_0}|_E$. Moreover, $G_n \in RED(\{r\})$ implies that E does not contain any c-labelled ege $\{v_1, v_2\}$ such that $l_{G_n}(v_1) = l_{G_n}(v_2) = *$. Hence, $E = E_{G_0} - \{\{v, v'\} \in E_{G_0} \mid m_{G_0}(\{v, v'\}) = c, l_{G_0}(v) = l_{G_0}(v') = *\}$. It follows that G' is of the required form.
2. Conversely, let $G, G' \in \mathscr{G}$ such that G' is defined as in the theorem. Since to each $G \in \mathscr{G}$ with a c-labelled edge e connecting unlabelled nodes, r is applicable, we get from the finiteness of E_G and the definition of r that there is a derivation $G \Rightarrow_r^* G''$ such that $G'' \in RED(\{r\})$ (because r reduces the number of c-labelled edges by 1 if it is applied once). Then, as shown in point 1, $G'' \cong G'$.

□

Theorem A.3. Let $G, G' \in \mathscr{G}$ such that G is labelled over $\{*, b\}$. Then $(G, G') \in SEM(copy)$ iff $G' \cong (V, E, l, m)$ where

$$V = V_G \uplus \{cp(v) \mid v \in V_G\}$$

$$E = E_G \cup \{\{cp(v_1), cp(v_2)\} \mid \{v_1, v_2\} \in E_G\} \cup \{\{v, cp(v)\} \mid v \in V_G^b\}$$ [16]

$$l(v) = \begin{cases} l_G(v), & v \in V_G \\ l_G(v'), & v \in \{cp(v'),\ v' \in V_G\} \end{cases} \quad \textit{for all } v \in V$$

$$m(e) = \begin{cases} c, & e \in \{\{v, cp(v)\} \mid v \in V_G^b\} \\ *, & \textit{otherwise} \end{cases} \quad \textit{for all } e \in E.$$

[16] For $x \in C_V$ ($x \in C_E$) the set of all x-labelled nodes (edges) in G is denoted by V_G^x (E_G^x).

Proof. By Observation 4.1 we get that

$$SEM(copy) = SEM(copy\_items) \circ SEM(delete).$$

Hence, by Theorems 1 and 2, $(G, G') \in SEM(copy)$ iff G' is defined as in the theorem. □

Theorem A.4. Let $G, G' \in \mathscr{G}$. Let $C(G) = \{\{v_1, v_2\} \in E_G^c \mid v_1, v_2 \in V_G^b\}$. Then $(G, G') \in SEM(next\_rank)$ iff $G' \cong (V, E, l, m)$ where

$$V = V_G \uplus \{cp_j(e) \mid e \in C(G), j \in \{1,2\}\}$$

$$E = E_G - C(G) \cup \{\{v_j, cp_k(\{v_1, v_2\})\} \mid \{v_1, v_2\} \in C(G), j,k \in \{1,2\}\}$$

$$l(v) = \begin{cases} b, & v \in \{cp_j(e) \mid e \in C(G), j \in \{1,2\}\} \\ *, & v \in \bigcup_{e \in C(G)} e \\ l_G(v), & otherwise \end{cases} \quad for\ all\ v \in V$$

$$m(e) = \begin{cases} m_G(e), & e \in E_G \\ *, & otherwise \end{cases} \quad for\ all\ e \in E.$$

Proof. The proof is based on two lemmas that are given and proved in the following.

Lemma 3. Let $G_0 \in \mathscr{G}$. Then for each derivation $G_0 \Rightarrow_r \cdots \Rightarrow_r G_n$ with $\{r\} = R_{next\_rank}$, there are sets $C_0, \ldots, C_n$ such that $C_0 \subset C_1 \subset \cdots \subset C_n \subseteq C(G_0)$ and for $i = 0, \ldots, n$, $G_i \cong (V_i, E_i, l_i, m_i)$ where

$$V_i = V_{G_0} \uplus \{cp_j(e) \mid e \in C_i, j \in \{1,2\}\}$$

$$E_i = (E_{G_0} - C_i) \cup \{\{v_j, cp_k(\{v_1, v_2\})\} \mid \{v_1, v_2\} \in C_i, j,k \in \{1,2\}\}$$

$$l_i(v) = \begin{cases} b, & v \in \{cp_j(e) \mid e \in C_i, j \in \{1,2\}\} \\ *, & v \in \bigcup_{e \in C_i} e \\ l_{G_0}(v), & otherwise \end{cases} \quad for\ all\ v \in V_i$$

$$m_i(e) = \begin{cases} m_{G_0}(e), & e \in E_{G_0} \\ *, & otherwise \end{cases} \quad for\ all\ e \in E_i.$$

Proof of Lemma 3. If $n = 0$ the lemma obviously holds choosing $C_0 = \emptyset$. Assume that it holds for some $n \in \mathbb{N}$. Let $G_0 \Rightarrow_r \cdots \Rightarrow_r G_n \Rightarrow_r G_{n+1}$ such that w.l.o.g. $G_n = (V_n, E_n, l_n, m_n)$. Then by definition of r there exists an edge $e' = \{v_1, v_2\} \in E_{G_n}^c$ with $v_1, v_2 \in V_{G_n}^b$. By induction hypothesis, $m_n(e') = c$ implies $e' \in C(G_0) - C_n$. Moreover, $G_{n+1} \cong (V, E, l, m)$ where

$$V = V_n \uplus \{cp_1(e'), cp_2(e')\}$$

$$E = (E_n - \{e'\}) \cup \{\{v_j, cp_k(e')\} \mid j,k \in \{1,2\}\}\}$$

$$l(v) = \begin{cases} b, & v \in \{cp_1(e'), cp_2(e')\} \\ *, & v \in e' \\ l_n(v), & otherwise \end{cases} \quad for\ all\ v \in V$$

$$m_i(e) = \begin{cases} m_n(e), & e \in E_{G_n} \\ *, & otherwise \end{cases} \quad for\ all\ e \in E.$$

Choose $C_{n+1} = C_n \cup \{e'\}$. Then $C_n \subset C_{n+1}$. Moreover, by induction, $C_{n+1} \subseteq C(G_0)$ and G_{n+1} is of the required form.

Lemma 4. Let $G_0 \in \mathscr{G}$ and let $G_0 \Rightarrow_r G_1 \Rightarrow_r \cdots \Rightarrow_r G_n$ such that $\{r\} = R_{next\_rank}$ and $G_n \in RED(\{r\})$. Then $C_n = C(G_0)$, where C_n is defined as in the proof of Lemma 3.

Proof of Lemma 4. W.l.o.g. let $G_n = (V_n, E_n, l_n, m_n)$ as defined in Lemma 3. From Lemma 1 we know that $C_n \subseteq C(G_0)$. Let $e \in C(G_0) - C_n$. Then by Lemma 3, $e \in C(G_n)$, i.e. r is applicable to G_n. This completes the proof of Lemma 4.

1. Now let $(G, G') \in SEM(next\_rank)$. Then by definition there is a derivation $G_0 \Rightarrow_r^* G_n$ such that $G_0 = G$, $G_n = G'$, and $G' \in RED(\{r\})$. Let C_n be defined as in the proof of Lemma 3. Then by Lemma 4, $C_n = C(G)$ and by Lemma 3, G_n is of the required form.
2. Conversely, let $G, G' \in \mathscr{G}$ such that G' is defined as in the theorem. Since $C(G)$ is finite, we get from Lemma 3 that there is a derivation $G \Rightarrow_r G''$ such that $G'' \in RED(\{r\})$. Then, as shown in point 1, $G'' \cong G'$.

This completes the proof. □

The following theorem states that the transformation unit *butterfly* generates the set of all butterfly networks.

Theorem A.5. $SEM(butterfly) = \{(B_0, B_k) \mid k \in \mathbb{N}\}$.

Proof. For the semantics of *butterfly* the following holds:

$$\begin{aligned}
SEM(butterfly) &=_{obs.2.2} SEM_{E(buttterfly)}(L((copy\ ;\ next_rank)^*)) \cap (\{B_0\} \times \mathscr{G}) \\
&=_{def} SEM_{E(buttterfly)}((copy\ ;\ next_rank)^*) \cap (\{B_0\} \times \mathscr{G}) \\
&=_{obs.2.1} SEM_{E(buttterfly)}(copy\ ;\ next_rank)^* \cap (\{B_0\} \times \mathscr{G}) \\
&=_{obs.2.1} (SEM_{E(buttterfly)}(copy) \circ SEM_{E(buttterfly)}(next_rank))^* \cap \\
&\quad (\{B_0\} \times \mathscr{G}) \\
&=_{def} (SEM(copy) \circ SEM(next_rank))^* \cap (\{B_0\} \times \mathscr{G}).
\end{aligned}$$

Hence, $(G, G') \in SEM(butterfly)$ iff there are graphs $G_0, M_1, G_1, \ldots, M_n, G_n$ $(n \geqslant 0)$ such that $B_0 \cong G_0 = G$, $G_n = G'$, and for $i = 1, \ldots, n$, $(G_{i-1}, M_i) \in SEM(copy)$, and $(M_i, G_i) \in SEM(next\_rank)$.

1. We show by induction on n that $G_n \cong B_n$ if graphs $G_0, M_1, G_1, \ldots, M_n, G_n$ with this property exist. For $n = 0$, this obviously holds. Assume that $G_n \cong B_n$. Let $(G_n, M_{n+1}) \in SEM(copy)$, and $(M_{n+1}, G_{n+1}) \in SEM(next\_rank)$.
   By Theorem 3, $M_{n+1} \cong (V_{n+1}, E_{n+1}, l_{n+1}, m_{n+1})$ where

$$\begin{aligned}
V_{n+1} &= V_{G_n} \uplus \{cp(v) \mid v \in V_{G_n}\} \\
E_{n+1} &= E_{G_n} \cup \{\{cp(v_1), cp(v_2)\} \mid \{v_1, v_2\} \in E_{G_n}\} \cup \\
&\quad \{\{v, cp(v)\} \mid v \in V_{G_n}^b\}
\end{aligned}$$

$$l_{n+1}(v) = \begin{cases} l_{G_n}(v), & v \in V_{G_n} \\ l_{G_n}(v'), & v \in \{cp(v'),\ v' \in V_{G_n}\} \end{cases} \quad \text{for all } v \in V_{n+1}$$

$$m_{n+1}(e) = \begin{cases} c, & e \in \{\{v, cp(v)\}, v \in V_{G_n}^b\} \\ *, & \text{otherwise} \end{cases} \quad \text{for all } e \in E_{n+1}.$$

Then by induction, $M_{n+1} \cong (V', E', l', m')$ where

$$V' = \{v_{ir}, cp(v_{ir}) \mid i = 0,\dots,2^n-1, r = 0,\dots,n\}$$ [17]

$$\begin{aligned} E' = \; & \{\{v_{ir}, v_{j(r-1)}\}, \{cp(v_{ir}), cp(v_{j(r-1)})\} \mid \\ & i,j = 0,\dots,2^n-1, r = 1,\dots,n, \\ & i = j \text{ or } b_i(r) \neq b_j(r) \\ & \text{and } b_i(l) = b_j(l) \text{ for all } l \in \{1,\dots,n\} - \{r\}\} \cup \\ & \{\{v_{i0}, cp(v_{i0})\} \mid i = 0,\dots,2^n-1\} \end{aligned}$$

(here, as well as below, $b_i(1)\cdots b_i(n)$ is the binary representation of i $(0 \leqslant i < 2^n)$)

$$l'(v) = \begin{cases} *, & v \in \{v_{ir}, cp(v_{ir}) \mid \\ & \quad i = 0,\dots,2^n-1, r = 0,\dots,n\} \\ b, & v \in \{v_{i0}, cp(v_{i0}) \mid \\ & \quad i = 0,\dots,2^n-1\} \end{cases} \qquad \text{for all } v \in V'$$

$$m'(e) = \begin{cases} *, \; e \in \\ \quad \{\{v_{ir}, v_{j(r-1)}\}, \{cp(v_{ir}), cp(v_{j(r-1)})\} \mid \\ \quad i,j = 0,\dots,2^n-1, \\ \quad r = 1,\dots,n, \; i = j \text{ or } b_i(r) \neq b_j(r) \\ \quad \text{and } b_i(l) = b_j(l) \\ \quad \text{for all } l \in \{1,\dots,n\} - \{r\}\} \\ c, \; e \in \\ \quad \{\{v_{i0}, cp(v_{i0})\} \mid i = 0,\dots,2^n-1\} \end{cases} \quad \text{for all } e \in E'.$$

Let $C(M_{n+1})$ be defined as in Theorem 4. Then

$$C(M_{n+1}) = \{\{v_{i0}, cp(v_{i0})\} \mid i = 0,\dots,2^n-1\}$$

and we get by Theorem 4 that $G_{n+1} \cong (V, E, l, m)$ where

$$\begin{aligned} V = \; & \{v_{ir}, cp(v_{ir}) \mid i = 0,\dots,2^n-1, r = 0,\dots,n\} \cup \\ & \{cp_j(\{v_{i0}, cp(v_{i0})\}) \mid i = 0,\dots,2^n-1, j \in \{1,2\}\} \end{aligned}$$

$$\begin{aligned} E = \; & \{\{v_{ir}, v_{j(r-1)}\}, \{cp(v_{ir}), cp(v_{j(r-1)})\} \mid \\ & i,j = 0,\dots,2^n-1, r = 1,\dots,n, i = j \text{ or } b_i(r) \neq b_j(r) \\ & \text{and } b_i(l) = b_j(l) \text{ for all } l \in \{1,\dots,n\} - \{r\}\} \cup \\ & \{\{v_{i0}, cp_j(\{v_{i0}, cp(v_{i0})\})\} \mid i = 0,\dots,2^n-1, j \in \{1,2\}\} \cup \\ & \{\{cp(v_{i0}), cp_j(\{v_{i0}, cp(v_{i0})\})\} \mid i = 0,\dots,2^n-1, j \in \{1,2\}\} \end{aligned}$$

$$l(v) = \begin{cases} b, & v \in \{cp_j(\{v_{i0}, cp(v_{i0}\}) \mid \\ & \quad i = 0,\dots,2^n-1, j \in \{1,2\}\} \\ *, & \text{otherwise} \end{cases} \qquad \text{for all } v \in V$$

$$m(e) = *, \qquad \text{for all } e \in E.$$

Let $f_V : V \to V_{B_{n+1}}$ such that $f_V(v_{ir}) = v_{i(r+1)}$, $f_V(cp(v_{ir})) = v_{(i+2^n)r+1}$, $f_V(cp_1(\{v_{i0}, cp(v_{i0})\}) = v_{i0}$, and $f_V(cp_2(\{v_{i0}, cp(v_{i0})\}) = v_{(i+2^n)0}$, for $i = 0,\dots,2^n-1$, $r = 0,\dots,n$. Then

[17] In the following, set definitions like $\{v_{ir}, cp(v_{ir}) \mid i = 0,\dots,2^n-1, r = 0,\dots,n\}$ are used as abbreviations for $\{v_{ir} \mid i = 0,\dots,2^n-1, r = 0,\dots,n\} \cup \{cp(v_{ir}) \mid i = 0,\dots,2^n-1, r = 0,\dots,n\}$.

$$\begin{aligned} f_V(V) &= \{f_V(v_{ir}), f_V(cp(v_{ir})) \mid i = 0,\ldots 2^n - 1,\ r = 0,\ldots,n\} \cup \\ &\quad \{f_V(cp_j(\{v_{i0}, cp(v_{i0})\})) \mid i = 0\ldots,2^n - 1,\ j \in \{1,2\}\} \\ &= \{v_{ir} \mid i = 0,\ldots 2^{n+1} - 1,\ r = 1,\ldots,n+1\} \\ &\quad \{v_{i0} \mid i = 0,\ldots 2^{n+1} - 1\} \\ &= \{v_{ir} \mid \ = 0,\ldots 2^{n+1} - 1,\ r = 0,\ldots,n+1\} \\ &=_{def} V_{B_{n+1}} \end{aligned}$$

Moreover, $l = l_{B_{n+1}} \circ f_V$. Let $f_E : E \to E_{B_{n+1}}$ such that for all $\{v_1, v_2\} \in E$, $f_E(\{v_1, v_2\}) = \{f_V(v_1), f_V(v_2)\}$. Then

$$\begin{aligned} f_E(E) &= \{\{f_V(v_{ir}), f_V(v_{j(r-1)})\}, \{f_V(cp(v_{ir})), f_V(cp(v_{j(r-1)}))\} \mid \\ &\quad i,j = 0,\ldots,2^n - 1, r = 1,\ldots,n,\ i = j \ or\ b_i(r) \neq b_j(r) \\ &\quad and\ b_i(l) = b_j(l)\ for\ all\ l \in \{1,\ldots,n\} - \{r\}\} \cup \\ &\quad \{\{f_V(v_{i0}), f_V(cp_k(\{v_{i0}, cp(v_{i0})\}))\} \mid i = 0,\ldots,2^n - 1, \\ &\quad k \in \{1,2\}\} \cup \{\{f_V(cp(v_{i0})), f_V(cp_k(\{v_{i0}, cp(v_{i0})\}))\} \mid \\ &\quad i = 0,\ldots,2^n - 1,\ k \in \{1,2\}\} \\ &= \{\{v_{ir}, v_{j(r-1)}\} \mid i,j = 0,\ldots,2^{n+1} - 1, r = 2,\ldots,n+1, i = j \\ &\quad or\ b_i(r) \neq b_j(r) and\ b_i(l) = b_j(l) \\ &\quad for\ all\ l \in \{1,\ldots,n+1\} - \{r\}\} \cup \\ &\quad \{\{v_{i1}, v_{j0}\} \mid i,j = 0,\ldots,2^{n+1} - 1,\ i = j\ or\ b_i(1) \neq b_j(1) \\ &\quad and\ b_i(l) = b_j(l)\ for\ all\ l \in \{2,\ldots,n+1\}\}. \end{aligned}$$

Moreover, $m = m_{B_{n+1}} \circ f_E$. Hence, $(f_V, f_E) : G_{n+1} \to B_{n+1}$ is a graph isomorphism, i.e. G_{n+1} is a butterfly network.

2. We now show by induction on n that $(B_0, B_n) \in SEM(butterfly)$. Obviously, $(B_0, B_0) \in SEM(butterfly)$. Assume that for some $n \in \mathbb{N}$, $(B_0, B_n) \in SEM(butterfly)$. Then there is an interleaving sequence in *butterfly* from B_0 to B_n. From Theorems 3 and 4 we get that there are graphs M_{n+1}, G_{n+1} such that $(B_n, M_{n+1}) \in SEM(copy)$, and $(M_{n+1}, G_{n+1}) \in SEM(next\_rank)$. As shown in point 1, $G_{n+1} \cong B_{n+1}$. Hence, $(B_0, B_{n+1}) \in SEM(butterfly)$.

This completes the proof. □

Received July 1997
Accepted in revised form February 2000 by O Herzog