More About Control Conditions for Transformation Units*

Sabine Kuske

Universität Bremen, Fachbereich 3
Postfach 33 04 40, D-28334 Bremen, Germany
kuske@informatik.uni-bremen.de

Abstract. A transformation unit is a structuring principle for composing graph transformation systems from small units. One of the basic components of a transformation unit is its control condition which allows to restrict the non-determinism of graph transformation. The concept of transformation units is generic in the sense that each formalism which specifies a binary relation on graphs can be used as a control condition. This paper discusses a selection of concrete classes of control conditions which seem to provide reasonable expressive power for specifying and programming with transformation units. These include regular expressions, once, as-long-as-possible, priorities, and conditionals; some of them were already used in an ad hoc manner in earlier papers. It is shown which classes of control conditions can be replaced by others without changing the semantics of the corresponding transformation unit. Moreover, three properties of control conditions are studied: minimality, invertibility and continuity.

1 Introduction

Real applications of graph transformation may often consist of hundreds of rules which can only be managed in a reasonable and transparent way with the help of a structuring principle. Therefore, several modularization concepts for graph transformation systems are currently under development (cf. also [HEET99]). One of them is the transformation unit, a structuring principle which allows to decompose large graph transformation systems into small reusable units that can import each other (see [KK96,KKS97,AEH+99,KK99]). It transforms graphs by interleaving rule applications with calls of imported transformation units. One fundamental feature of the transformation unit is its independence of a particular graph transformation approach, i.e. of a selected graph data model, a specific type of rules, etc. (see [Roz97] for an overview of the main graph transformation approaches).

Each transformation unit contains a control condition that allows to regulate its graph transformation process. This is meaningful because formalisms which

* This work was partially supported by the ESPRIT Working Group Applications of Graph Transformation (APPLIGRAPH) and the EC TMR Network GETGRATS (General Theory of Graph Transformation Systems).

H. Ehrig et al. (Eds.): Graph Transformation, LNCS 1764, pp. 323–337, 2000.

allow to specify transformations of graphs in a rule-based way are usually nondeterministic for two reasons. First, there may be more than one rule which can be applied to a certain graph. Second, a rule can be applied to various parts of a graph. Hence, in order to program or specify with graph transformation, it is often desirable to regulate the graph transformation process, for example by choosing rules according to a priority, or by prescribing a certain sequence of steps (cf. e.g. [Sch75,Bun79,Nag79,MW96,Sch97,TER99], see also [DP89] for regulation concepts in string transformation).

The concept of transformation units is generic in the sense that each formalism which specifies a binary relation on graphs can be used as a control condition. In order to use transformation units for specifying and programming with graph transformation, concrete classes of control conditions must be employed.

Based on the variety of control mechanisms for graph transformation proposed in the literature, we discuss in this paper a selection of concrete classes of control conditions for transformation units which seem to provide reasonable expressive power for the practical use of transformation units. These include regular expressions, priorities, conditionals, once, and as-long-as-possible, where some of them were already used in an ad hoc manner in earlier papers [KK96,AEH+99]. We define their semantics in the context of transformation units and show how some classes of control conditions can be expressed by others without changing the interleaving semantics of the corresponding transformation unit. Moreover, three properties of control conditions are studied: minimality, invertibility, and continuity. The first two are required to perform certain operations on transformation units like flattening and inversion. The third leads to a fixed point semantics in the case of recursive transformation units. Please note that, due to space limitations, proofs are only sketched in this paper.

2 Transformation Units

A transformation unit comprises descriptions of initial and terminal graphs, a set of rules, a control condition, and a set of imported transformation units. It transforms initial graphs to terminal ones by interleaving rule applications with calls to imported transformation units such that the control condition is obeyed. Transformation units are a fundamental concept of the graph- and rule-centered language GRACE, currently under development by researchers from various sites. In the following we briefly recall the definition of transformation units together with its interleaving semantics.

Since transformation units are independent of a particular graph data model or a specific type of graph transformation rules, we assume that the following are given: a class $\mathcal{G}$ of *graphs*, a class $\mathcal{R}$ of *rules*, and a *rule application operator* $\Rightarrow$ specifying a binary relation $\Rightarrow_r \subseteq \mathcal{G} \times \mathcal{G}$ for each $r \in \mathcal{R}$. Intuitively, $\Rightarrow_r$ yields all pairs (G, G') of graphs where G' is obtained from G by applying r. In order to specify the initial and terminal graphs of a transformation unit, we assume in addition that a class $\mathcal{E}$ of *graph class expressions* is given where each $X \in \mathcal{E}$ specifies a set $SEM(X) \subseteq \mathcal{G}$. The control condition of a transformation unit

is taken from a class $\mathcal{C}$ of *control conditions* each of which specifies a binary relation on graphs. Since control conditions may contain identifiers (usually for imported transformation units or local rules), their semantics depends on their *environment*, a mapping which associates each identifier with a binary relation on graphs. This means more precisely that each $C \in \mathcal{C}$ specifies a binary relation $SEM_E(C) \subseteq \mathcal{G} \times \mathcal{G}$ for every mapping $E\colon ID \to 2^{\mathcal{G}\times\mathcal{G}}$ where ID is a set of names. All these components form a *graph transformation approach* $\mathcal{A} = (\mathcal{G}, \mathcal{R}, \Rightarrow, \mathcal{E}, \mathcal{C})$.

A *transformation unit* over $\mathcal{A}$ is a system $t = (I, U, R, C, T)$ where $I, T \in \mathcal{E}$, $R \subseteq \mathcal{R}$, $C \in \mathcal{C}$, and U is a set of (already defined) transformation units over $\mathcal{A}$. This should be taken as a recursive definition of the set $\mathcal{T}_\mathcal{A}$ of transformation units over $\mathcal{A}$. Hence, initially U is the empty set yielding unstructured transformation units which may be used in the next iteration, and so on. Note that, for reasons of space limitations, in this paper we consider only transformation units with an acyclic import structure. But the presented results (apart from Observation 2) can easily be transferred to the more general case of transformation units having a cyclic import structure which were studied in [KKS97].

The interleaving semantics of a transformation unit contains a pair (G, G') of graphs if G is an initial graph and G' is a terminal graph, G can be transformed into G' using the rules and the imported transformation units, and (G, G') is allowed by the control condition. Formally, let $t = (I, U, R, C, T) \in \mathcal{T}_\mathcal{A}$ and assume that every $t' \in U$ already defines a binary relation $SEM(t') \subseteq \mathcal{G} \times \mathcal{G}$ on graphs. Then the *environment of* t is the mapping $E(t)\colon ID \to 2^{\mathcal{G}\times\mathcal{G}}$ defined by $E(t)(id) = SEM(id)$ if $id \in U$, $E(t)(id) = \Rightarrow_{id}$ if $id \in R$, and $E(t)(id) = \emptyset$, otherwise.[1] The *interleaving semantics* of t is the relation $SEM(t) = (SEM(I) \times SEM(T)) \cap RIS(t) \cap SEM_{E(t)}(C)$ where $RIS(t) = (\bigcup_{id\in U\cup R} E(t)(id))^*$.[2] The name $RIS(t)$ stands for *relation induced by the interleaving sequences of* t because it consists of all pairs $(G, G') \in \mathcal{G} \times \mathcal{G}$ such that there is an interleaving sequence $G_0, \ldots, G_n$ of t with $G_0 = G$, $G_n = G'$. A sequence $G_0, \ldots, G_n$ $(n \geq 0)$ is called an *interleaving sequence of* t if, 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 Control Conditions

With control conditions, the non-determinism of the graph transformation process can be restricted so that a more functional behaviour is achieved. A typical example is to allow only iterated rule applications where the sequences of applied rules belong to a particular control language. In general, every description of a binary relation on graphs may be used as a control condition. In the following, we discuss a selection of control conditions suitable to specify and program with

[1] For technical simplicity, we assume here that ID contains U and R as disjoint sets. Nevertheless, if transformation units are used in a specification or programming language, ID should provide a set of predefined identifiers out of which the elements of U and R are named. Note that such an explicit naming mechanism can be added to the presented concepts in a straighforward way.

[2] For a binary relation ρ its reflexive and transitive closure is denoted by ρ^*.

graph transformation. These conditions are regular expressions, once, as-long-as-possible, priorities, and conditionals.

3.1 Regular Expressions

Regular expressions – well known from the area of formal languages – are useful to prescribe the order in which rules or imported transformation units should be applied. Here, we consider the class REG of regular expressions over ID which is recursively given by $\epsilon, \emptyset \in REG$, $ID \subseteq REG$, and $(C_1\,;\,C_2)$, $(C_1 \,|\, C_2)$, $(C^*) \in REG$ if $C, C_1, C_2 \in REG$. In order to omit parentheses, we assume that * has a stronger binding than $;$ which in its turn has a stronger binding than $|$. Intuitively, the expression ϵ requires that no rule or imported unit is applied, $\emptyset$ specifies the empty set, and $id \in ID$ applies the rule or transformation unit id exactly once. Moreover, $C_1\,;\,C_2$ applies first C_1 and then C_2, $C_1 \,|\, C_2$ chooses non-deterministically between C_1 and C_2, and C^* iterates C arbitrarily often. Formally, this means that for each environment E we have $SEM_E(\epsilon) = \Delta\mathcal{G}$, $SEM_E(\emptyset) = \emptyset$, $SEM_E(id) = E(id)$ for each $id \in ID$, $SEM_E(C_1\,;\,C_2) = SEM_E(C_1) \circ SEM_E(C_2)$, $SEM_E(C_1 \,|\, C_2) = SEM_E(C_1) \cup SEM_E(C_2)$, and $SEM_E(C^*) = SEM_E(C)^*$.[3]

The following transformation unit *Eulerian_test* has a regular expression as its control condition. It imports the four units $relabel\_all\_nodes(*, even)$, *check_degree*, $relabel\_all\_edges(ok, *)$, and $relabel\_all\_nodes(even, *)$, and applies them in this order exactly once. The initial graphs are all (undirected) unlabelled and connected graphs, whereas the terminal graphs are unlabelled.

Eulerian_test	
initial:	*unlabelled, connected*
uses:	$relabel\_all\_nodes(*, even)$, $relabel\_all\_nodes(even, *)$, $relabel\_all\_edges(ok, *)$, *check_degree*,
conds:	$relabel\_all\_nodes(*, even)\,;$ *check_degree* $;$ $relabel\_all\_edges(ok, *)\,;$ $relabel\_all\_nodes(even, *)$
terminal:	*unlabelled*

Given an appropriate interleaving semantics for the imported units, *Eulerian_test* can be used to check for each unlabelled and connected graph G whether it is Eulerian, i.e. $(G, G) \in SEM(Eulerian\_test)$ iff G is Eulerian. For this, the unit $relabel\_all\_nodes(*, even)$ labels each unlabelled node of its input graph with *even*; the unit *check_degree* keeps the label of each node with an even degree, changes the label of each node with an odd degree into *odd*, and labels each edge with *ok*; $relabel\_all\_edges(ok, *)$ unlabels each *ok*-labelled edge; and the unit $relabel\_all\_nodes(even, *)$ unlabels each *even*-labelled node. (Note that the symbol $*$ stands for unlabelled.) The unit *check_degree*, which does the main part of the algorithm, is presented in Sect. 3.3.

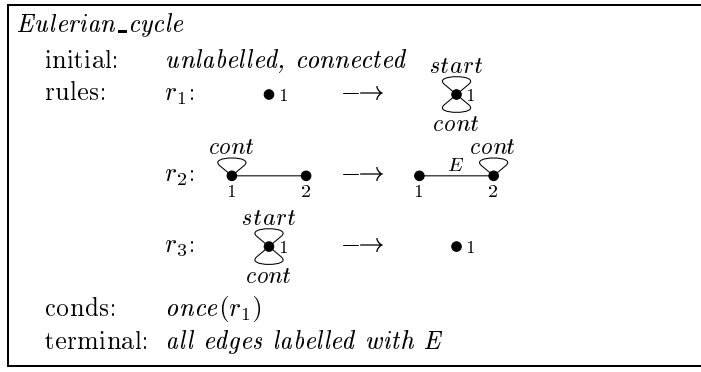
[3] $\Delta\mathcal{G}$ denotes the identity relation on $\mathcal{G}$ and $SEM_E(C_1) \circ SEM_E(C_2)$ the sequential composition of the relations $SEM_E(C_1)$ followed by $SEM_E(C_2)$.

Regular expressions can be extended in a straightforward way from the set ID to an arbitrary set M of control conditions of some other type. Hence, each regular expression over M also serves as a control condition the semantics of which is defined analogously to the one above. This "extended class" of regular expressions is useful for iterating the execution of or choosing non-deterministically between various control conditions in M.

3.2 Once

The control condition $once(id)$, where id is (the name of) a rule or an imported transformation unit, allows only interleaving sequences in which id is applied exactly once. All other rules or imported units can be applied arbitrarily often. Formally, for each environment E, $(G, G') \in SEM_E(once(id))$ if there exist $G_0, \ldots, G_n \in \mathcal{G}$ and $id_1 \cdots id_n \in ID^*$ such that (1) $G_0 = G$ and $G_n = G'$, (2) $(G_{i-1}, G_i) \in E(id_i)$ for $i = 1, \ldots, n$, and (3) there exists exactly one $i \in \{1, \ldots, n\}$ with $id_i = id$.

An example of a transformation unit using the control condition once and double pushout rules [CEH+97] is *Eulerian_cycle*, which searches for a Eulerian cycle in a connected unlabelled non-empty graph. The rule r_1 is applied exactly once and marks a node with a *start*-loop and a *cont*-loop, where *cont* stands for *continue*. With rule r_1 it is determined at which node the unit begins to traverse its input graph with r_2. The rule r_2 passes the *cont*-loop of a node v to a node v' provided that v and v' are connected via an unlabelled edge e, and labels e with E. After each application of r_2 the E-labelled edges form a path in the current graph from the node with the *start*-loop to the node with the *cont*-loop. The rule r_3 of *Eulerian_cycle* deletes the *cont*-loop and the *start*-loop provided that they are attached to the same node. The terminal component requires that all edges are labelled with E. The semantics of *Eulerian_cycle* consists of all pairs (G, G') where G is a non-empty Eulerian graph and G' is obtained from G by labelling all edges with E.



The next observation states that once can be replaced by a regular expression. This follows from the definitions of the semantics of regular expressions, once, and transformation units.

Observation 1 Let $t = (I, U, R, C, T)$ with $C = once(id)$ and let $C' = c\,;\, id\,;\, c$ where $c = (id_1 \mid \cdots \mid id_n)^*$ and $\{id_1, \ldots, id_n\} = (U \cup R) \backslash \{id\}$. Then $SEM_{E(t)}(C) = SEM_{E(t)}(C')$.

If the underlying approach fulfills some weak assumptions, each transformation unit with a regular expression as its control condition can be transformed into a semantically equivalent one with the control condition $once(r)$ where r is a rule. We illustrate the construction for the case where the underlying rules are double pushout rules. (But note that in general it can be done for many other rule classes.) For this purpose, we consider an arbitrary but fixed graph transformation approach $\mathcal{A} = (\mathcal{G}, \mathcal{R}, \Rightarrow, \mathcal{E}, \mathcal{C})$ where $\mathcal{G}$, $\mathcal{R}$, and $\Rightarrow$ are defined as in the double pushout approach, and $REG \subseteq \mathcal{C}$.

Observation 2 Let $t = (I, U, R, C, T) \in \mathcal{T}_{\mathcal{A}}$ with $C \in REG$. Then we can construct $t' = (I, U', R', once(r), T)$ such that $r \in R$ and $SEM(t) = SEM(t')$.

Proof. (Sketch) The unit t' is obtained as follows: (1) Construct a finite deterministic automaton aut with state set Q, alphabet ID, transition function $\delta\colon Q \times ID \to Q$, initial state $s \in Q$, and final state $f \in Q$ such that aut recognizes the language specified by C, and the states of Q do not occur as node labels in $\mathcal{G}$. (Note that aut can always be constructed.) (2) For each $\bar{t} = (\bar{I}, \bar{U}, \bar{R}, \bar{C}, \bar{T}) \in U$ and $q, q' \in Q$ let $(\bar{t}, q, q')$ be the unit

$(\bar{t}, q, q')$	
initial:	$\{G \uplus \bullet_q \mid G \in SEM(\bar{I})\}$
uses:	$\bar{U}$
rules:	$\bar{R} \cup \{rem(q), ins(q')\}$
conds:	$rem(q)\,;\, C\,;\, ins(q')$
terminal:	$\{G \uplus \bullet'_q \mid G \in SEM(\bar{T})\}$

where for each $q \in Q$ and each $G \in \mathcal{G}$, $G \uplus \bullet_q$ denotes the disjoint union of G and a q-labelled node, $rem(q)$ removes a q-labelled node, and $ins(q)$ inserts a q-labelled node (i.e. $rem(q) = (\bullet_q, \emptyset, \emptyset)$ and $ins(q) = (\emptyset, \emptyset, \bullet_q)$). For all $r = (L, K, R) \in R$ and $q, q' \in Q$ let (r, q, q') be the double pushout rule $(L \uplus \bullet_q, K, R \uplus \bullet_{q'})$. Then define $U' = \bigcup_{t \in U}\{(t, q, q') \mid q, q' \in Q, \delta(q, t) = q')\}$, $R' = \bigcup_{r \in R}\{(r, q, q') \mid q, q' \in Q, \delta(q, r) = q')\} \cup \{ins(s), rem(f)\}$, and $r = ins(s)$.

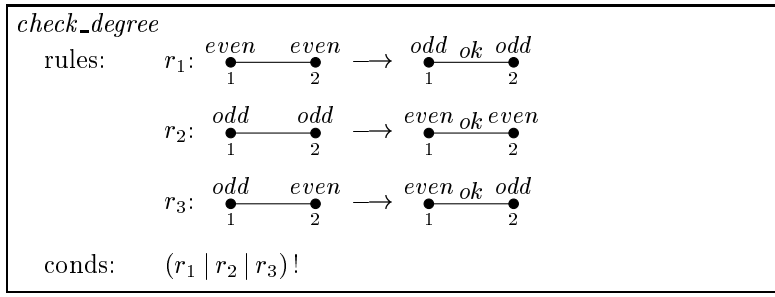
As shown in [KK96], $(G, G') \in SEM(t)$ iff there are $x_1 \cdots x_n \in (U \cup R)^*$ and $G_0, \ldots, G_n \in \mathcal{G}$ with $G_0 = G$ and $G_n = G'$ such that (a) $x_1 \cdots x_n$ is specified by C and (b) for $i = 1, \ldots, n$, $(G_{i-1}, G_i) \in E(t)(x_i)$. By definition (a) is equivalent to the fact that there are $q_0, \ldots, q_n \in Q$ with $q_0 = s$, $q_n = f$ and $\delta(q_{i-1}, x_i) = q_i$ (i.e. $x_1 \cdots x_n$ is recognized by aut). We also have by definition that (b) is equivalent to $(G_{i-1} \uplus \bullet_{q_{i-1}}, G_i \uplus \bullet_{q_i}) \in E(t')(x_i, q_{i-1}, q_i)$ $(i = 1, \ldots, n)$. Moreover, $(G_0, G_0 \uplus \bullet_s) \in E(t')(ins(s))$ and $(G_n \uplus \bullet_f, G_n) \in E(t')(rem(f))$.

Hence, we get on the one hand that $SEM(t) \subseteq SEM(t')$. Conversely, by definition of t' and aut the unit t' always applies $ins(s)$ in the first step and $rem(f)$ in the last (exactly once). Hence, on the other hand $SEM(t') \subseteq SEM(t)$. □

3.3 As-Long-As-Possible

In general, a control condition C specifies a binary relation on graphs. The control condition $C!$ iterates this process as long as possible. For example, if C is a rule, the condition $C!$ applies this rule to the current graph as long as possible. Formally, for each environment E the condition $C!$ specifies the set of all pairs $(G, G') \in SEM_E(C)^*$ such that there is no $G'' \in \mathcal{G}$ with $(G', G'') \in SEM_E(C)$.

The control condition of the following transformation unit *check_degree* requires that the rules r_1, r_2, and r_3 are applied as long as possible. The rules are relabelling rules, i.e. they do not change the underlying structure of a graph but only its labelling. For example, r_1 relabels two *even*-labelled nodes into *odd*-labelled ones and an unlabelled edge connecting these nodes into an *ok*-labelled one.



When used within the unit *Eulerian_test* of Sect. 3.1 the input graph of *check_degree* is some non-empty connected graph where all edges are unlabelled and all nodes are labelled with *even*. Remember that for such a graph, the unit *check_degree* keeps the label of each node with an even degree, changes the label of each node with an odd degree into *odd*, and labels each edge with *ok*.

The next observation states that each condition of the form $(r_1|\cdots|r_n)!$ where $r_1, \ldots, r_n$ are rules can be replaced by the expression $(r_1 \,|\cdots|\, r_n)^*$ if the underlying class $\mathcal{E}$ of graph class expressions fulfills the following requirements: (1) For each rule set P, the expression $reduced(P)$ is in $\mathcal{E}$ which specifies the set of all graphs that are reduced w.r.t. P, i.e. $G \in SEM(reduced(P))$ if and only if no rule in P is applicable to G; (2) $\mathcal{E}$ is closed under intersection, i.e. for all $X_1, X_2 \in \mathcal{E}$, $X_1 \wedge X_2 \in \mathcal{E}$ with $SEM(X_1 \wedge X_2) = SEM(X_1) \cap SEM(X_2)$. The proof is obtained in a straightforward way.

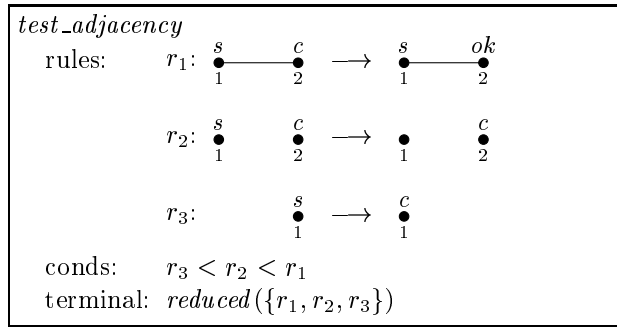
Observation 3 Let $t = (I, U, R, (r_1|\cdots|r_n)!, T)$ with $r_i \in R$ $(i = 1, \ldots, n)$. Let $t' = (I, U, R, (r_1 \,|\cdots|\, r_n)^*, reduced(\{r_1, \ldots, r_n\}) \wedge T)$. Then $SEM(t) = SEM(t')$.

3.4 Priorities

A *priority* consists of a set $N \subseteq ID$ and a non-reflexive partial order $<$ on N. Intuitively, such a condition allows to apply a rule or a transformation unit $id \in N$ to the current graph G only if no rule or transformation unit $id' \in N$

of higher priority is applicable to G. Formally, let $C = (N, <)$ be a priority and let for each $id \in N$, $HP(id, C) = \{id' \mid id < id'\}$. Then for each environment E, $(G, G') \in SEM_E((N, <))$ if there exist $G_0, \ldots, G_n \in \mathcal{G}$ such that (1) $G_0 = G$ and $G_n = G'$ and (2) for $i = 1, \ldots, n$, $(G_{i-1}, G_i) \in E(id_i)$ for some $id_i \in N$ and for all $id \in HP(id_i, C)$ there is no $G \in \mathcal{G}$ with $(G_{i-1}, G) \in E(id)$.

The following transformation unit *test_adjacency* has a priority control condition. If the input graph is an undirected node-labelled graph with exactly one s-labelled node v, it tests whether v is adjacent to each c-labelled node. In this case v is labelled with c and all c-labelled nodes with ok. Otherwise v becomes unlabelled. The rules are again relabelling ones. The condition $r_2 < r_1$ guarantees that the label s is only deleted if there exists a c-labelled node not adjacent to v. Because of $r_3 < r_2$ the label of v can only be transformed into c if v is adjacent to all c-labelled nodes in the input graph. The terminal expression requires that no rule of $\{r_1, r_2, r_3\}$ is applicable to the output graph. (Note that the unit *test_adjacency* can be used for example to find a clique in an undirected unlabelled graph.)



Each transformation unit t with a priority control condition can be transformed into a semantically equivalent unit t' the control condition of which is composed of regular expressions and as-long-as-possible. Roughly speaking, the construction is based on the fact that the control condition $(\{id_1, id_2\}, <)$, where $id_2 < id_1$ can be replaced by $(id_1 \mid (id_1\,!)\,;\, id_2)^*$.

Observation 4 Let $t = (I, U, R, C, T)$ with $C = (N, <)$ and $N = \{id_1, \ldots, id_n\}$. Let $C' = (p(id_1) \mid \cdots \mid p(id_n))^*$, where for each $id \in N$, $p(id) = id$ if $HP(id, C) = \emptyset$, and $p(id) = ((p(id'_1) \mid \cdots \mid p(id'_m))\,!)\,;\, id$ if $m > 0$ and $\{id'_1, \ldots, id'_m\} = HP(id, C)$. Then $SEM(t) = SEM((I, U, R, C', T))$.

Proof. (Sketch) For each $id \in N$ let the *recursion depth* $rd(id)$ of id be defined by $rd(id) = 0$ if $HP(id, C) = \emptyset$, and $rd(id) = \max\{rd(id'_1), \ldots, rd(id'_m)\} + 1$ if $\{id'_1, \ldots, id'_m\} = HP(id, C)$ and $m > 0$. (Note that $rd(id)$ is well defined because $<$ is acyclic.) By induction on $rd(id_i)$ $(i = 1, \ldots, n)$ we get $SEM_{E(t)}(p(id_i)) \subseteq SEM_{E(t)}((N, <))$ which implies that $SEM_{E(t)}(C') \subseteq SEM_{E(t)}((N, <))$. Conversely, consider for $(G, G') \in SEM_{E(t)}((N, <))$ a sequence $G_0, \ldots, G_n$ of graphs

satisfying the properties according to the definition of priorities. By induction on n we get $SEM_{E(t)}((N,<)) \subseteq SEM_{E(t)}(prior(id_1) \mid \cdots \mid p(id_n))^*)$. Hence, $SEM(t) = SEM((I,U,R,C',T))$. □

Let $t = (I,U,R,C,T)$ be a transformation unit where C is a *rule priority* $(N,<)$, i.e. $<$ is defined only on R. Then under some assumptions t can be transformed into a semantically equivalent transformation unit with the default control condition *true* specifying all pairs of graphs.

The basic idea of the construction is to encode priorities into negative contexts of rules which forbid the application of the rules to a graph G if certain subgraphs – specified in the negative contexts – occur in G. Rules with negative contexts are a restricted version of the rules with negative application conditions studied within the single pushout approach in [HHT96].

In order to transform the rule priority of t into *true*, we assume that each rule p of the underlying approach $\mathcal{A}$ has a *left-hand side* $lhs(p)$ and is applicable to a graph G whenever there exists a graph morphism from $lhs(p)$ to G. (Note that for example the single pushout approach [EHK+97] as well as the hyperedge replacement approach [DHK97] and the node replacement approach [ER97] fulfill these requirements.) A rule with negative contexts is of the form $r = (\mathcal{N},p)$ where $\mathcal{N}$ is a set of graphs, p is a rule, and for each $G \in \mathcal{N}$ the left-hand side of p is a proper subgraph of G, i.e. $lhs(p) \subset G$. Such a rule r is applied to a graph G according to the following steps: (1) CHOOSE a graph morphism g from $lhs(p)$ to G. (2) CHECK the negative context condition: for each $G' \in \mathcal{N}$ there is no graph morphism h from G' to G such that h restricted to $lhs(p)$ is equal to g. (3) APPLY p to G in the same way as in $\mathcal{A}$.

Observation 5 Let $t = (I,U,R,C,T) \in \mathcal{T}_{\mathcal{A}}$ such that $C = (N,<)$ is a rule priority with $N = R \cup U$. For each $p \in \mathcal{R}$ let $neg(p,C) = (\mathcal{N},p)$ such that $\mathcal{N} = \{lhs(p) \uplus lhs(p') \mid p' \in HP(p,C)\}$. Let $t' = (I,U,\{neg(p,C) \mid p \in R\},true,T)$. Then $SEM(t) = SEM(t')$.

Proof. By definition we have $G \Rightarrow_{neg(p,C)} G'$ iff $G \Rightarrow_p G'$ and for each $p' \in HP(p,C)$ there is no graph G'' such that $G \Rightarrow_{p'} G''$. Based on this, we get by definition that $SEM_{E(t)}(C) = RIS(t')$. It follows that $SEM(t') =_{def} SEM(I) \times SEM(T) \cap RIS(t') = (SEM(I) \times SEM(T)) \cap SEM_{E(t)}(C)$. Moreover, by definition $SEM_{E(t)}(C) \subseteq RIS(t)$. Hence, $SEM(t) = SEM(t')$. □

Remark. Note that the requirement $N = R \cup U$ can be made without loss of generality because $SEM_{E(t)}(C) \subseteq RIS(t)$. More precisely, t can be transformed into a semantically equivalent transformation unit $t' = (I,U',R',(N' <'),T) \in \mathcal{T}_{\mathcal{A}}$ where $U' = U \cap N$ and $R' = R \cap N$, $N' = U' \cup R'$, and $<'$ is the restriction of $<$ to N'.

3.5 Conditionals

A *conditional* is of the form IF X THEN C where X is a graph class expression and C a control condition (which is already defined). For each environment E

it allows all pairs (G, G') of graphs such that $G \in SEM(X)$ and $(G, G') \in SEM_E(C)$.

For each graph class expression X let $\neg X$ specify all graphs which are not in $SEM(X)$ (i.e. $SEM(\neg X) = \mathcal{G} - SEM(X)$). Then we can construct control conditions which iterate conditionals and branch in the control flow, if we combine conditionals with regular expressions (over control conditions). Some examples are:

- (IF X THEN C) | (IF $\neg X$ THEN C') executes C if X is fulfilled and C' otherwise.
- (IF X THEN C)! executes C as long as the current graph belongs to $SEM(X)$.
- (IF X THEN C)* executes C if X is fulfilled and iterates this arbitrarily often.

Obviously, each control condition C of the previous paragraphs can be expressed by the conditional IF *all* THEN C where *all* denotes a graph class expression specifying the set of all graphs. Moreover, conditionals do not increase the semantic power of transformation units, i.e. $(I, U, R,$ IF X THEN $C, T)$ can be replaced by the semantically equivalent unit $(I \wedge X, U, R, C, T)$.

3.6 Further Control Conditions

There are some further control conditions worth to be mentioned. First, each transformation unit t 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 t is given by $SEM(t)$. Second, each pair $(X_1, X_2) \in \mathcal{E} \times \mathcal{E}$ defines a binary relation on graphs by $SEM((X_1, X_2)) = SEM(X_1) \times SEM(X_2)$ and, therefore, it can be used as a control condition which is independent of the choice of an environment, i.e. $SEM_E((X_1, X_2)) = SEM((X_1, X_2))$ for all environments E. Third, another kind of rule priorities is based on graph transformation approaches where a rule r is applied to a graph G at a so-called *occurrence* which is a subgraph of G. (This is the case in all major graph transformation approaches.) Such a rule priority allows to apply a rule at an occurrence *occ* only if no occurrence of a rule with higher priority overlaps with *occ* (cf. [LM93]). Fourth, BCF expressions [Sch97] can be used as control conditions. Since they provide a certain kind of implicit import which in general may be cyclic, they should be used for transformation units with non-hierarchical import structure. In the case of acyclic BCF expressions, it can be shown that they can be directly translated into semantically equivalent transformation units with non-hierarchical import structure using only control conditions of the types presented in the previous subsections.

4 Properties of Control Conditions

In the following, we present some features of control conditions. First, we introduce minimal control conditions which are used for the flattening of transformation units in [KK96] and show that for each transformation unit there is

a semantically equivalent one whose control condition is minimal. The semantics of a transformation unit with a minimal control condition can be computed without constructing the interleaving sequences. It is just obtained from the control condition and the initial and terminal expressions. Second, invertible control conditions are considered. Invertibility of operations is a useful property for systems where previous states should be reconstructible. In this sense, an interesting question is under which conditions a transformation unit t can be transformed into a transformation unit t' which specifies the inverted relation of t. It turns out that this is only possible if the control condition of t is invertible (cf. [KK96]). Finally, we consider continuity of control conditions, a property which leads to a fixed point semantics in the case of cyclic import structures (cf. [KKS97]).

4.1 Minimality

A control condition C of a transformation unit $t = (I, U, R, C, T)$ is *minimal* (with respect to t) if C only specifies pairs of graphs which occur in the relation induced by the interleaving sequences of t, i.e. $SEM_{E(t)}(C) \subseteq RIS(t)$. In a transformation unit t with a minimal control condition the explicit computation of $RIS(t)$ is no longer necessary, i.e. $SEM(t) = (SEM(I) \times SEM(T)) \cap SEM_{E(t)}(C)$. As mentioned above, minimality of control conditions is one of the features required for flattening a transformation unit, i.e. for constructing a semantically equivalent one with empty import.

It can be shown by induction on the structure of regular expressions that each $C \in REG$ is minimal. Moreover, by definition priorities are minimal and the control conditions of the form $C!$ are minimal if C is minimal.

Moreover, for each transformation unit there is a semantically equivalent one with a minimal control condition if the underlying approach provides regular expressions and intersection of control conditions. The proof is straightforward.

Observation 6 Let $t = (I, U, R, C, T)$ and let $C' = C \wedge (id_1 \,|\, \cdots \,|\, id_n)^*$ where $\{id_1, \ldots, id_n\} = U \cup R$ and $SEM_E(C') = SEM_E(C) \cap SEM_E((id_1 \,|\, \cdots \,|\, id_n)^*)$ for each environment E. Then C' is minimal with respect to t, and $SEM(t) = SEM((I, U, R, C', T))$.

4.2 Invertibility

An *invertible* class $\mathcal{C}$ of control conditions contains for each $C \in \mathcal{C}$ a condition C^{-1} which, roughly speaking, specifies the inverse of the relation specified by C. More precisely, if C specifies in some environment E the relation ρ, the condition C^{-1} specifies the inverse of ρ in the inverted environment by associating with each identifier id the inverse of $E(id)$, i.e. $SEM_{inv(E)}(C^{-1}) = \rho^{-1}$ where $inv(E)(id) = E(id)^{-1}$ for each $id \in ID$. Invertibility of control conditions is required for the inversion of transformation units, a useful operation for reconstructing previous states of graph transformation systems when programming with transformation units.

It can be shown that the class *REG* and the class of all boolean expressions over *REG* are invertible (cf. [KK96]). Hence, $once(id)$ is also invertible (see Observation 1). Moreover, under some assumptions we can construct an inverse control condition for each condition of the types introduced in Sects. 3.3–3.5.

Concretely, for each conditional IF X THEN C we can construct its inverse control condition if C can be inverted. The proof follows directly from the definitions.

Observation 7 Let $\mathcal{C}$ be an invertible class of control conditions and let $C \in \mathcal{C}$. Then $SEM_E(\text{IF } X \text{ THEN } C)^{-1} = SEM_{inv(E)}(C^{-1}\,;\, \text{IF } X \text{ THEN } \epsilon)$ for each environment E and each graph class expression X.

To invert as-long-as-possible and priorities, we require that for certain control conditions C and each environment E there exists a graph class expression (C, E) specifying all graphs G such that there is no G' with $(G, G') \in SEM_E(C)$. The proof of Observation 8 follows directly from the definitions. For Observation 9 let $p(id)$ be defined as in Observation 4.

Observation 8 Let $\mathcal{C}$ be an invertible class of control conditions and let $C \in \mathcal{C}$. Then $SEM_E(C!)^{-1} = SEM_{inv(E)}(\text{IF } (C, E) \text{ THEN } (C^{-1})^*)$ for each environment E.

Observation 9 Let $C = (N, <)$ be a priority with $N = \{id_1, \ldots, id_n\}$. For each environment E and each $id \in N$ let $p(id, E) = id$ if $HP(id, C) = \emptyset$, and $p(id, E) = id\,;\, \text{IF } ((p(id'_1) \,|\, \cdots \,|\, p(id'_m)), E) \text{ THEN } (p(id'_1, E) \,|\, \cdots \,|\, p(id'_m, E))^*$ if $HP(id, C) = \{id'_1, \ldots, id'_m\}$ and $m > 0$. Then $SEM_E((N, <))^{-1} = SEM_{inv(E)}((p(id_1, E) \,|\, \cdots \,|\, p(id_n, E))^*)$.

Proof. (Sketch) By Observation 4 $SEM_E((N, <))^{-1} = SEM_E((p(id_1) \,|\, \cdots \,|\, p(id_n))^*)^{-1}$, which is equal to $(SEM_E(p(id_1))^{-1} \cup \cdots \cup SEM_E(p(id_n))^{-1})^*$. By induction on the recursion depth $rd(id)$ of $id \in N$, we get $SEM_E(p(id))^{-1} = SEM_{inv(E)}(p(id, E))$. Hence, $SEM_E((N, <))^{-1} = (SEM_{inv(E)}(p(id_1, E)) \cup \cdots \cup SEM_{inv(E)}(p(id_n, E)))^* = SEM_{inv(E)}((p(id_1, E) \,|\, \cdots \,|\, p(id_n, E))^*)$. □

Remark. The constructions for inverting priorities can be simplified by distributing them over several transformation units. The idea is that for each environment E each control condition of the form $id\,;\, \text{IF } ((p(id_1)| \,\cdots\, | p(id_n)), E) \text{ THEN } (p(id_1, E) \,|\, \cdots \,|\, p(id_m, E))^*$ can be expressed for example by $id\,;\, id'$ where id' is an imported unit with control condition $\text{IF } ((p(id_1)| \,\cdots\, | p(id_n)), E) \text{ THEN } (p(id_1, E) \,|\, \cdots \,|\, p(id_m, E))^*$. Analogously, the control condition of id' can be simplified into $\text{IF } (id'', E) \text{ THEN } id'''$, etc.

4.3 Continuity

A control condition C is *continuous* if for each chain of environments $E_1 \subseteq E_2 \subseteq \cdots$ the following holds: Computing first the semantics of the control condition with respect to each environment in the chain and then taking the

union of the resulting relations yields the same result as computing the semantics of the control condition with respect to the union of the environments in the chain, i.e. $\bigcup_{i\in\mathbb{N}} SEM_{E_i}(C) = SEM_{\cup_{i\in\mathbb{N}} E_i}(C)$.[4]

Observation 10 Regular expressions are continuous. Priorities and as-long-as-possible are non-continuous.

Proof. (Sketch) The first statement follows by induction on the structure of regular expressions. Now, let $p = (\{id_1, id_2\}, <)$ be the priority control condition with $id_2 < id_1$. Let $G_0, \ldots, G_4$ be non-isomorphic graphs. Let $E_0 \subseteq E_1 \subseteq \cdots$ be such that for each $i > 1$, $E_i = E_{i+1}$, and $E_0(id_1) = \{(G_1, G_2)\}$, $E_0(id_2) = \{(G_2, G_3)\}$, $E_1(id_1) = \{(G_1, G_2), (G_2, G_4)\}$, and $E_1(id_2) = E_0(id_2)$. Then by definition $(G_1, G_3) \in SEM_{E_0}(p)$, but $(G_1, G_3) \notin SEM_{E_1}(p)$. Hence, $(G_1, G_3) \in \bigcup_{i\in\mathbb{N}} SEM_{E_i}(p)$, but $(G_1, G_3) \notin SEM_{\cup_{i\in\mathbb{N}} E_i}(p)$. This implies that priorities are not continuous. The proof of the non-continuity of the condition as-long-as-possible is very similar. □

Remark. In the case of a transformation unit t with a possibly cyclic import structure one gets a fixed point semantics if the control conditions are continuous w.r.t. imported transformation units (cf. [KKS97]). This means that a fixed point also exists in the cases of $(N, <)$ and $C!$ provided that $<$ and C only refer to local rules.

5 Conclusion

We have proposed classes of control conditions for transformation units and shown interrelations between them. Moreover, some properties of control conditions were given. There remains some further work to be done:

- The practical usability of the proposed classes of control conditions should be examined with the help of realistic case studies.
- As mentioned before, for each BCF expression with acyclic import structure there is a semantically equivalent transformation unit with the presented classes of control conditions. We conjecture that the set of all BCF expressions can be translated into transformation units with arbitrary import structure and the control conditions discussed here. This should be thoroughly worked out, since a large number of regulation mechanisms proposed for transformation systems can be defined with BCF expressions.
- In the area of term graph rewriting, reduction strategies like *innermost*, *leftmost*, etc. are used that cannot be expressed with the conditions presented in Sect. 3 because their semantics depend on the specific structure of the intermediate term graphs in interleaving sequences. In general, there may

[4] For two environments E_1 and E_2, $E_1 \subseteq E_2$ if for each $id \in ID$, $E_1(id) \subseteq E_2(id)$; and $E_1 \cup E_2$ is the environment defined by $E_1 \cup E_2(id) = E_1(id) \cup E_2(id)$ for all $id \in ID$.

be various application areas of graph transformation which demand such a particular type of control conditions. Those classes of conditions should be formulated and studied within the framework of transformation units.
- In [EH86], a general notion of application conditions for graph transformation rules is introduced. Special classes of this notion are studied in [LM93] or [HHT96], for example. The relations between such application conditions and control conditions of transformation units should be studied systematically. Note that Observation 5 is a first step in this direction, because it relates rule priorities with negative application conditions.
- The presented control conditions should be compared with those employed in rule-based systems which are not based on graph transformation (like, for example, expert systems).
- In Sect. 4, some properties of control conditions were presented. Probably, there are further interesting properties in the context of regulating the graph transformation process in a modular way. This should be worked out.

Acknowledgement

I thank Hans-Jörg Kreowski, Frank Drewes, Renate Klempien-Hinrichs, and the anonymous referees for their helpful comments.

References

AEH+99. 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. 323, 324

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. 324

CEH+97. Andrea Corradini, Hartmut Ehrig, Reiko Heckel, Michael Löwe, Ugo Montanari, and Francesca Rossi. Algebraic approaches to graph transformation part I: Basic concepts and double pushout approach. In Rozenberg [Roz97]. 327

DHK97. Frank Drewes, Annegret Habel, and Hans-Jörg Kreowski. Hyperedge replacement graph grammars. In Rozenberg [Roz97], pages 95–162. 331

DP89. Jürgen Dassow and Gheorghe Păun. *Regulated Rewriting in Formal Language Theory*, volume 18 of *EATCS Monographs on Theoretical Computer Science*. Springer-Verlag, 1989. 324

EEKR99. Hartmut Ehrig, Gregor Engels, Hans-Jörg Kreowski, and Grzegorz Rozenberg, editors. *Handbook of Graph Grammars and Computing by Graph Transformation, Vol. II: Applications, Languages and Tools*. World Scientific, Singapore, 1999. To appear. 337

EH86. 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. 336

EHK+97. Hartmut Ehrig, Reiko Heckel, Martin Korff, Michael Löwe, Leila Ribeiro, Annika Wagner, and Andrea Corradini. Algebraic approaches to graph transformation II: Single pushout approach and comparison with double pushout approach. In Rozenberg [Roz97], pages 247–312. 331

ER97. Joost Engelfriet and Grzegorz Rozenberg. Node replacement graph grammars. In Rozenberg [Roz97], pages 1–94. 331

HEET99. Reiko Heckel, Gregor Engels, Hartmut Ehrig, and Gabriele Taentzer. Classification and comparison of modul concepts for graph transformation systems. In Ehrig et al. [EEKR99]. To appear. 323

HHT96. Annegret Habel, Reiko Heckel, and Gabriele Taentzer. Graph grammars with negative application conditions. *Fundamenta Informaticae*, XXVI(3,4):287–313, 1996. 331, 336

KK96. 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. 323, 324, 328, 332, 333, 334

KK99. Hans-Jörg Kreowski and Sabine Kuske. Graph transformation units and modules. In Ehrig et al. [EEKR99]. To appear. 323

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. 323, 325, 333, 335

LM93. Igor Litovsky and Yves Métivier. Computing with graph rewriting systems with priorities. *Theoretical Computer Science*, 115:191–224, 1993. 332, 336

MW96. Andrea Maggiolo-Schettini and Józef Winkowski. A kernel language for programmed rewriting of (hyper)graphs. *Acta Informatica*, 33(6):523–546, 1996. 324

Nag79. Manfred Nagl. *Graph-Grammatiken: Theorie, Anwendungen, Implementierungen*. Vieweg, Braunschweig, 1979. 324

Roz97. Grzegorz Rozenberg, editor. *Handbook of Graph Grammars and Computing by Graph Transformation, Vol. I: Foundations*. World Scientific, Singapore, 1997. 323, 336, 337

Sch75. Hans-Jürgen Schneider. Syntax-directed description of incremental compilers. In D. Siefkes, editor, *GI — 4. Jahrestagung*, volume 26 of *Lecture Notes in Computer Science*, pages 192–201, 1975. 324

Sch97. Andy Schürr. Programmed graph replacement systems. In Rozenberg [Roz97], pages 479–546. 324, 332

TER99. Gabriele Taentzer, C. Ermel, and Michael Rudolf. The AGG-approach: Language and tool environment. In Ehrig et al. [EEKR99]. To appear. 324