Confluence of Typed Attributed Graph Transformation Systems

Reiko Heckel, Jochen Malte Küster, and Gabriele Taentzer

University of Paderborn, Paderborn, Germany
Technical University of Berlin, Berlin, Germany
reiko|jkuester@upb.de, gabi@cs.tu-berlin.de

Abstract. The issue of confluence is of major importance for the successful application of attributed graph transformation, such as automated translation of UML models into semantic domains. Whereas termination is undecidable in general and must be established by carefully designing the rules, local confluence can be shown for term rewriting and graph rewriting using the concept of critical pairs. In this paper, we discuss typed attributed graph transformation using a new simplified notion of attribution. For this kind of attributed graph transformation systems we establish a definition of critical pairs and prove a critical pair lemma, stating that local confluence follows from confluence of all critical pairs.

1 Introduction

Graph transformation is increasingly popular as a meta-language to specify and implement visual modelling techniques, like the UML. It may be used for parsing visual languages [1] and for automated translation of visual models into code or semantic domains [6, 19], or as a semantic domain itself [13, 9]. Often, it is important to know whether the graph transformation system shows a functional behavior (is terminating and confluent) or if there are conflicts between rule applications that lead to true non-determinism. For example, functional behavior avoids the overhead of backtracking in the case of parsing and semantic ambiguity in the mapping of models to semantic domains.

For term rewrite systems confluence can be shown using the concept of critical pairs. Critical pairs which can be detected and analyzed statically, represent potential conflicts in a minimal context. If the rewrite system is terminating, confluence follows if all critical pairs can be joined [15].

This theory of critical pairs and confluence has been transferred to transformation systems on term graphs and hyper graphs [17, 18]. However, in most applications of graph transformation to visual modelling techniques, *attributed* graphs are used to represent diagrams with textual, numerical, or layout information, semantic annotations, etc. To develop the theory of critical pairs and confluence in this case is the aim of this paper.

In particular, we shall be motivated by the problem of translating diagrams into a formal specification language for automatic verification. Such a mapping

can be described by attributed graph transformation rules based on the graphical presentation of the abstract syntax of the diagrams extended by semantic attributes that contain the results of the translation [10]. The translation has to be functional, i.e., terminating and confluent in order to ensure the existence of a unique result. As an example we present a translation of simple UML statecharts into CSP [11] for automated verification by means of a CSP model checker [8].

For this purpose, we introduce typed, vertex-attributed graph transformation systems. A critical pair lemma is established which states that a graph transformation system is locally confluent if all critical pairs are confluent. Thus, confluence can be shown by computing all critical pairs and demonstrating their confluence.

In the following, we first introduce typed attributed graph transformation and present, as a running example, rules for translating UML statecharts to CSP. Thereafter, critical pairs are defined and the critical pair lemma is stated. Lastly, the critical pairs in our example are discussed and the available tool support is described.

2 Typed attributed graph transformation

Next, we present the *algebraic double-pushout (DPO) approach* [5] to the transformation of typed attributed graphs [2]. The two basic ingredients are graphs, representing object structures, and algebras representing pre-defined abstract data types. Attributed graphs occur at two levels: the type level (modelling a schema or class diagram) and the instance level (modelling an individual system snapshot).

Attributed graphs. By a *graph* we mean a directed unlabelled graph $G = \langle G_V, G_E, src^G, tar^G \rangle$ with a set of vertices G_V, a set of edges G_E, and functions $src^G : G_E \to G_V$ and $tar^G : G_E \to G_V$ associating to each edge its source and target vertex. A graph homomorphism $f : G \to H$ is a pair of functions $\langle f_V : G_V \to H_V, f_E : G_E \to H_E \rangle$ preserving source and target.

To speak about algebras throughout the paper, we assume a many-sorted signature $\Sigma = \langle S, OP \rangle$ consisting of a set of sort symbols $s \in S$ and a family of sets of operation symbols $op : s_1 \dots s_n \to s \in OP$ indexed by their arities.

Definition 1 (attributed graphs and morphisms). *An attributed graph (over Σ) is a pair $AG = \langle G, A \rangle$ of a graph G and a Σ-algebra A such that $\mid A \mid \subseteq G_V$, where $\mid A \mid$ is the disjoint union of the carrier sets A_s of A, for all $s \in S$, and such that $\forall\, e \in G_E : \ src(e) \notin \mid A \mid$. Let $Attr(AG) = \{e \in G_E \mid tar(e) \in \mid A \mid\}$, $Graph(AG) = G \setminus (\mid A \mid + Attr(AG))$ and $Alg(AG) = A$.*

An attributed graph morphism $f : \langle G_1, A_1 \rangle \to \langle G_2, A_2 \rangle$ *is a pair of a Σ-homomorphism $f_A = (f_s)_{s \in S} : A_1 \to A_2$ and a graph homomorphism $f_G = \langle f_V, f_E \rangle : G_1 \to G_2$ such that $\mid f_A \mid \subseteq f_V$, where $\mid f_A \mid = \bigcup_{s \in S} f_s$.*

Attributed graphs and graph morphisms form a category of Σ-attributed graphs $\mathbf{AGraph}(\Sigma)$. Often, we will fix the data algebra A in advance—in this case we also speak of a graphs and graph morphisms attributed over A.

Summarizing, data values are represented as vertices of graphs, henceforth called *data vertices* $d \in\mid A \mid$ to distinguish them from *object vertices* $v \in G_V \setminus \mid A \mid$. Object vertices are linked to data vertices by *attributes*, i.e., edges $a \in G_E$ with $src(a) = v$ and $tar(a) = d$. Edges between object vertices are called *links*. We have assumed that there are no edges from data vertices.

Compared with other notions of attributed graphs, like [14], where special attribute carriers are used to relate graph elements and data values, our presentation is simpler because attributed graphs are regarded as a special case of ordinary graphs. However, this limits us to attributed vertices.

Typed graphs. The concept of typed graphs [2] captures the well-known dichotomy between classes and objects, or between database schema and instance, in the case of graphs. Below, it is extended to attributed graphs.

Definition 2 (typed attributed graphs). *An* attributed type graph *over Σ is an attributed graph $ATG = \langle TG, Z \rangle$ over Σ where Z is the final Σ-algebra Z having $Z_s = \{s\}$ for all $s \in S$.*

An attributed instance graph $\langle AG, ag \rangle$ over ATG is an attributed graph AG (over the same signature) equipped with an attributed graph morphism $ag : AG \to ATG$.

A morphism of typed attributed graphs $h : \langle AG_1, ag_1 \rangle \to \langle AG_2, ag_2 \rangle$ is a morphism of attributed graphs which preserves the typing, that is, $ag_2 \circ h = ag_1$.

Thus, elements of Z represent the sorts of the signature which are included in TG as types for data vertices. In general, vertices and edges of TG represent vertex and edge types, while attributes in ATG are, in fact, attribute declarations. Given an attribute declaration $a \in ATG$ and an object vertex $v \in AG$ such that $ag(v) = src(a)$ we write $a(v)$ to denote the set of v's a-values $\{d \in\mid A \mid \mid \exists\, e \in AG_E.\ src(e) = v \wedge tar(e) = d\}$.

Instance graphs will be usually infinite, e.g. if the data type $\mathbb{N}$ of natural numbers is present, each $n \in \mathbb{N}$ will be a separate vertex. However, since the data type part will be kept constant during transformation, there is no need to represent this infinite set of vertices as part of the current state. The examples shown contain only those data vertices connected to some object vertex.

Sample type and instance graphs. For the translation of statecharts to CSP, the UML metamodel has to be flattened and inheritance must be removed by simulating it through additional attributes. In Fig. 2, an abridged metamodel for statecharts is shown as attributed type graph where types of model elements are depicted by rectangles while data types are shown as ellipses. Note that the inscriptions of the nodes and edges are the node and edge identities and no labels. The inscriptions of the data types such as String refers to the sort symbols in the signature given in Fig. 1.

SORTS

String, CSPEq, CSPExpr, Char

OPNS

empty: ->String
concat: String, String -> String
concat: String, Char -> String
empty: -> CSPExpr
stop: -> CSPExpr
[]: -> CSPExpr
->: string, CSPExpr -> CSPExpr
State: string -> CSPExpr
=: CSPExpr, CSPExpr -> CSPEq

Fig. 1. Signature for CSP translation

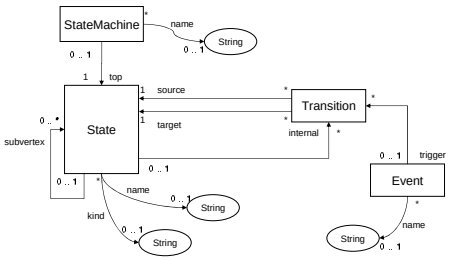

Fig. 2. Type graph for statecharts, derived from the UML metamodel

The type graph contains cardinality constraints in UML-like notation, which restrict the number of in and outgoing edges of vertices. The formal treatment of such constraints, however, is beyond the scope of this paper.

In Fig. 3, a simple instance graph for a statechart is shown. On the left, the formal representation is depicted with attribute values being modelled as vertices of the graph whereas on the right the UML-like syntax is given with the short-hand for attributes modelled inside a compartment of a class vertex. In the following, we will use this short-hand for space reasons and conformity with UML syntax.

Graph transformation. The DPO approach to graph transformation has originally been developed for vertex- and edge-labeled graphs [5]. Here, we present the typed version [2] extended to attributed graphs.

According to the DPO approach, graph transformation rules (also called graph productions), are specified by pairs of injective graph morphisms ($L \xleftarrow{l} K \xrightarrow{r} R$), called rule spans. The left-hand side L contains the items that must be present for an application of the rule, the right-hand side R those that are present afterwards, and the gluing graph K specifies the "gluing items", i.e., the objects which are read during application, but are not consumed. The transformation of graphs is defined by a pair of pushout diagrams, a so-called double pushout.

Definition 3 (DPO graph transformation). *Given an attributed type graph ATG and fixing a sort-indexed family of sets of variables $X = (X_s)_{s \in S}$, an ATG-typed graph transformation rule over X is a span of injective graph morphisms $p = (L \xleftarrow{l} K \xrightarrow{r} R)$ over ATG such that L, K, R are attributed over $T_\Sigma(X)$ and l, r are identities on $T_\Sigma(X)$. If we are not interested in the gluing graph K we write $p : L \curvearrowright R$.*

A double-pushout (DPO) diagram *o is a diagram like below where (1), (2) are pushouts and top and bottom are rule spans. Given a rule p a* (direct DPO)

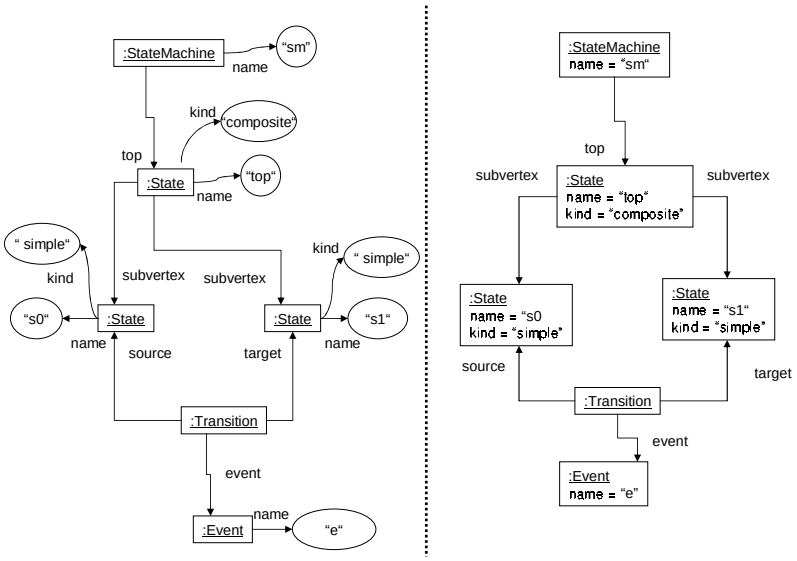

Fig. 3. Instance graph: formal vs UML-like syntax

transformation *from G to H, denoted by $G \overset{p(o)}{\Longrightarrow} H$, is given by a DPO diagram where g, h are identities on $Alg(D)$.*

$$\begin{array}{ccccc} L & \xleftarrow{l} & K & \xrightarrow{r} & R \\ {\scriptstyle o_L}\downarrow & (1) & \downarrow{\scriptstyle o_K} & (2) & \downarrow{\scriptstyle o_R} \\ G & \xleftarrow[g]{} & D & \xrightarrow[h]{} & H \end{array}$$

The DPO diagram o is a categorical way of representing the occurrence of a rule in a bigger context. Operationally, it formalizes the replacement of a subgraph in a graph by two gluing diagrams, called pushouts. The left-hand side pushout (1) is responsible for removing the occurrence of $L \setminus l(K)$ in G, resulting in graph D. The right-hand side pushout (2) adds a copy of $R \setminus r(K)$ to D leading to the derived graph H.

The construction of pushout (1) requires that only objects in the image of K may be merged or (in the case of vertices) connected to edges in the context. This is reflected, respectively, in the *identification* and the *dangling condition* of the DPO approach, i.e. the *gluing condition*. Given a rule $p = (L \xleftarrow{l} K \xrightarrow{r} R)$ and an occurrence $o_L : L \to G$ of the left-hand side, the existence of the pushout complement (1), and hence of a direct derivation[1] $G \overset{p(o)}{\Longrightarrow} H$ is characterized by the satisfaction of the gluing condition. The *identification condition* states that objects from the left-hand side may only be identified by the match if they also

[1] Pushout (2) always exists, since category $\frown Graph_{TG}$ is cocomplete due to the cocompleteness of category $\frown Graph$.

belong to the interface (and are thus preserved). The *dangling condition* ensures that graph D obtained by removing all objects that are to be deleted from G, is indeed a graph, i.e. no edges are left "dangling" without source or target node.

Definition 4 (graph transformation system). *A graph transformation system $GTS = (\Sigma, ATG, X, R)$ consists of a data type signature Σ, an attributed type graph ATG over Σ, a family of variables X over Σ, and a set of attributed graph transformation rules R over ATG and X.*

A transformation sequence $G_0 \overset{*}{\Longrightarrow} G_n = G_0 \overset{p_1(o_1)}{\Longrightarrow} \cdots \overset{p_n(o_n)}{\Longrightarrow} G_n$ in GTS is a sequences of consecutive transformation steps such that G_0 is typed over ATG and all rules p_i come from R.

Rules for mapping statecharts to CSP. Mapping rules for statecharts to the process algebra of Communicating Sequential Processes (CSP [11]) can be described by attributed graph transformation rules [6]. Such a rule consists of a UML metamodel instance extended by semantic attributes and control attributes. Within the semantic attributes, the actual computation of CSP expressions for the statechart is performed. Control attributes drive the order of rule applications. If an attribute is changed applying a rule, we use an assignment notation.

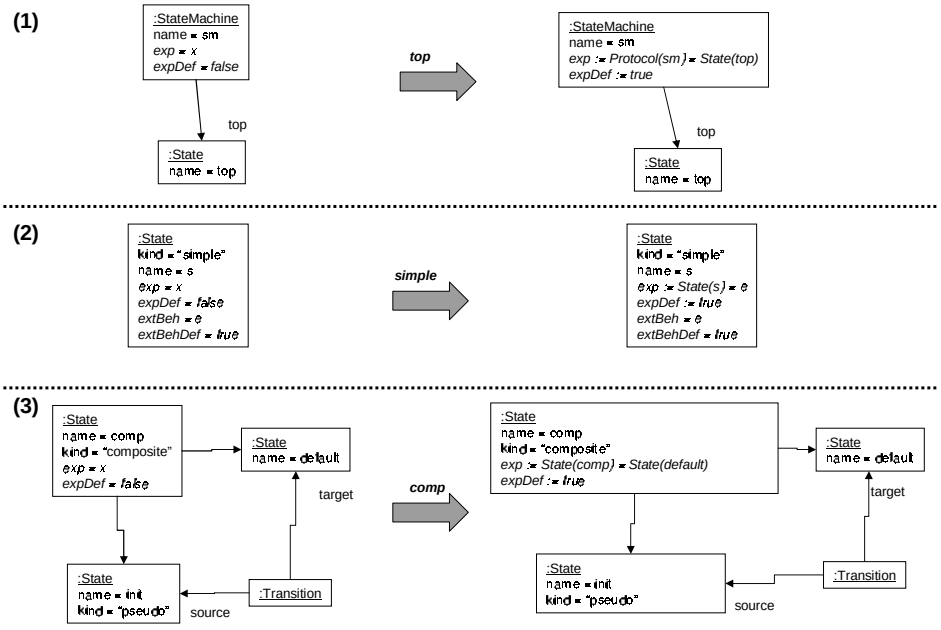

Fig. 4. Mapping rules for states

Consider, for example, rule (3) in Fig. 4 which defines the semantics of a composite (OR) state in terms of the semantics of its default state. The original attributes from the metamodel are represented in plain font whereas the

semantic and control attributes are printed in italics. In this case, the composite state is annotated with a semantic attribute exp and a control attribute expDef. Application of this rule leads to an exp set to a CSP expression defining the behavior of the composite state to be the behavior of the default state, and a change in the control attribute expDef from false to true, thereby hindering another application of this rule to the same composite state.

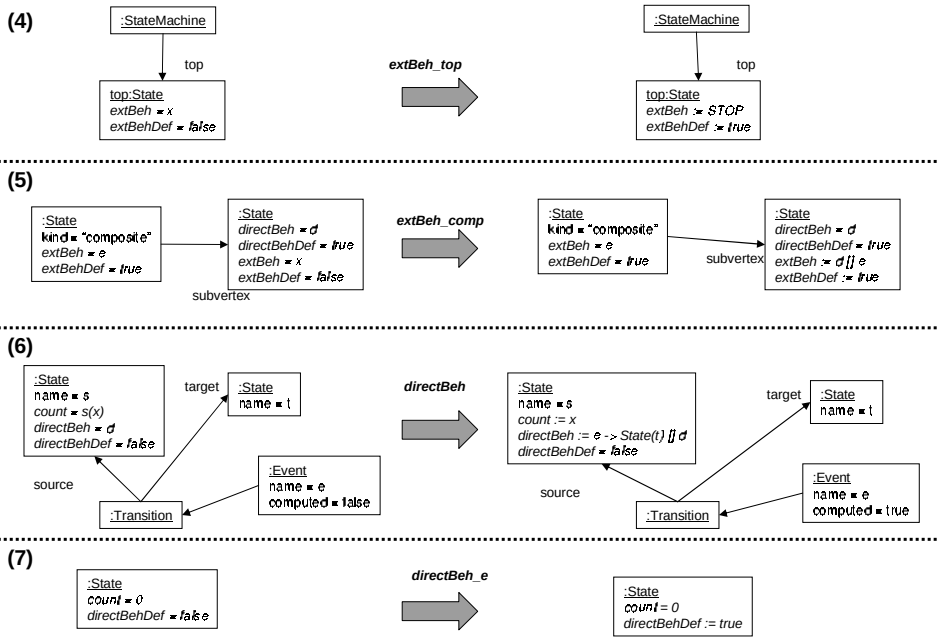

Fig. 5. Mapping rules for the behavior

In general, termination is undecidable for graph transformation systems. However, concerning our mapping rules, termination results from the following argument: Each rule is annotated with a finite number of control attributes. These control attributes take on a number of finite values (mostly true or false or a natural number). As each rule application either decreases a control attribute or changes its value from false to true, after a finite number of rule applications no rule will be applicable anymore, leading to termination of the mapping process.

3 Critical Pairs and Confluence

Independent transformations. The definition of parallel independence demands that the occurrences of two independent transformations do only share items which are preserved by both steps. Under this assumption, the local Church-Rosser theorem states that the two steps can be executed in any order with the same overall result [4].

Given two transformations $G \overset{p_1(o_1)}{\Longrightarrow} H_1$ and $G \overset{p_2(o_2)}{\Longrightarrow} H_2$, $G \overset{p_1(o_1)}{\Longrightarrow} H_1$ is *(weakly) parallel independent* of $G \overset{p_2(o_2)}{\Longrightarrow} H_2$ if the occurrence $o_1(L_1)$ of the left-hand side of p_1 is preserved by the application of p_2. This is the case if $o_1(L_1) \cap o_2(L_2 \setminus g_2(D_2)) = \emptyset$, that is, $o_1(L_1)$ does not overlap with objects that are deleted by p_2. If the two transformations are mutually independent, they can be applied in any order yielding the same result. In this case we speak of *parallel independence*. Otherwise, if one of two alternative transformations is not independent of the second, the second will disable the first. In this case, the two steps are *in conflict*.

Proposition 1 (local Church-Rosser theorem). *Given two parallel independent transformations* $H_1 \overset{p_1(o_1)}{\Longleftarrow} G \overset{p_2(o_2)}{\Longrightarrow} H_2$ *with* $H_1 \xleftarrow{h_1} C_1 \xrightarrow{g_1} G \xleftarrow{g_2} C_2 \xrightarrow{h_2} H_2$*, there are transformations* $H_1 \overset{p_2(o_2')}{\Longrightarrow} X$ *and* $H_2 \overset{p_1(o_1')}{\Longrightarrow} X$ *with* $o_1' = h_2 \circ g_2^{-1} \circ o_1$ *and* $o_2' = h_1 \circ g_1^{-1} \circ o_2$.

The local Church-Rosser theorem has been shown for colored graphs in [4]. Simply rephrasing its proof for typed graphs would lead to the proof of proposition 1.

In the case of attributed graph transformation, where attribute values are modelled as vertices and attribute links are edges, any modification of an attribute corresponds to a deletion of an attribute link and the generation of a new one. Therefore, two steps which modify the same attribute, are in conflict.

Critical pairs. A system is locally confluent if all conflicting pairs are *confluent*, that is, they are extendible by transformation sequences leading to a common successor graph. In order to check this in finite time, the potentially infinite set of conflicting pairs has to be reduced to a finite set of representatives. This is the aim of the construction of *critical pairs*, which produces all conflicting pairs of steps. A critical pair is *minimal*, i.e., it does not contain unnecessary context. It is also *syntactic*, meaning that it is attributed over a term algebra $T_\Sigma(X)$ or a quotient term algebra $T_\Sigma(X)/_\equiv$ for congruence relation $\equiv$ specified by a set of equational axioms or implemented by an equality predicate of an abstract data type.

Definition 5 (critical pairs). *Assume a graph transformation system* $GTS = (\Sigma, ATG, X, R)$*, a congruence* $\equiv \subseteq T_\Sigma(X) \times T_\Sigma(X)$*, and two rules* $p_1 : L_1 \curvearrowright R_1$ *and* $p_2 : L_2 \curvearrowright R_2$ *of* R *using disjoint subsets of variables of* X *in their attribute terms.*

A critical pair candidate for p_1 and p_2 wrt. $\equiv$ *is a pair of non-parallel independent transformations* $CP(p_1, k_1, K, p_2, k_2) = P_1 \overset{p_1(k_1)}{\Longleftarrow} K \overset{p_2(k_2)}{\Longrightarrow} P_2$*, with* K *attributed over the quotient term algebra* $T_\Sigma(X)/_\equiv$.

A critical pair for p_1 and p_2 wrt. $\equiv$ *is a minimal element among the candidates w.r.t. the partial order* $\sqsubseteq$ *defined by* $CP(p_1, k_1, K, p_2, k_2) \sqsubseteq CP(p_1, k_1', K', p_2, k_2')$ *iff there exists a morphism* $k : K \to K'$*, injective on* $Graph(K)$ *and* $Attr(K)$ *(but not necessarily on* $Alg(K)$*), such that* $k_1' = k \circ k_1$ *and* $k_2' = k \circ k_2$.

For the graph structure, minimality means that no unnecessary context is present, i.e. k_1 and k_2 are jointly surjective. For the algebra part, the minimality condition generalizes the idea of a most general unifier, that is, a substitution $\sigma : X \to T_\Sigma(X)$ with as little instantiation of variables as needed to equate two terms. Thus, $\sqsubseteq$ always has a set of minimal elements.

Next we consider three cases where the set of critical pairs for two given rules is finite (up to isomorphic copies) and can be effectively computed. In each case, we assume as given an overlapping $\mathit{Graph}(K)$ of the graph structures $\mathit{Graph}(L_1)$ and $\mathit{Graph}(L_2)$ of L_1 and L_2 with two graph morphisms $k_{i\,G} : \mathit{Graph}(L_i) \to \mathit{Graph}(K)$. The set of these overlappings is finite (up to isomorphism) if the graphical parts of L_1 and L_2 are both finite. The problem consists in checking if such an overlapping can be extended to the data type part, i.e., if a $T_\Sigma(X)/_\equiv$-attributed graph K exists with Σ-homomorphisms $k_{i\,A} : T_\Sigma(X) \to T_\Sigma(X)/_\equiv$ such that $\langle k_{i\,G}, k_{i\,A}\rangle : L_i \to K$ form attributed graph morphisms. We restrict our considerations to the case of single-valued (rather than multi-valued) attributes.

Case 1. First, we assume that the congruence $\equiv$ is trivial, i.e., it contains only the syntactic identities. In this case, $T_\Sigma(X)/_\equiv = T_\Sigma(X)$, that is, K is attributed over terms with variables of X. The attribute term for a vertex v in K is obtained by computing the most general unifier of all pre-images of v under k_1 and k_2.

More precisely, call $U(a, v) \subseteq T_\Sigma(X)$ the *unification set* for an attribute a of object vertex $v \in K$, given by $U(a, v) = \{t \mid (v = k_1(v_1) \wedge a(v_1) = t) \vee (v = k_2(v_2) \wedge a(v_2) = t)\}$, and enumerate the unification sets for all vertices v and all relevant attributes a as $U_1, \ldots, U_n$.

Now, a candidate $CP(p_1, k_1, K, p_2, k_2)$ is *attribute unifiable* if there exist substitutions σ_i such that $\sigma_1 = \mathit{mgu}(U_1)$ and, for all $j \in \{2, \ldots, n\}$, $\sigma_j = \mathit{mgu}(\sigma_{j-1}(U_j))$ where $\mathit{mgu}(U_i)$ computes the most general unifier of the set of terms U_i, if it exists.

If $CP(p_1, k_1, K, p_2, k_2)$ is attribute unifiable, the value of an attribute a for a vertex v in K is $\sigma_n(t)$ for any $t \in U(a, v)$. The algebra homomorphism part of both k_1 and k_2 is the free homomorphic extension to $T_\Sigma(X)$ of the same $\sigma_n : X \to T_\Sigma(X)$. If $CP(p_1, k_1, K, p_2, k_2)$ is not attribute unifiable, there is no critical pair based on this gluing of graphs.

Case 2. A second, more general variant allows a congruence $\equiv$ specified by a set of equational axioms, represented computationally by a confluent and terminating term rewrite system. In this case, the normal forms of this rewrite system can be used as unique representatives of their equivalence classes so that, effectively, a graph attributed over $T_\Sigma(X)/_\equiv$ can be represented as a $T_\Sigma(X)$-attributed graph. Since for normal forms, equivalence coincides with syntactic equality, we can reuse the construction of Case 1 by transforming the attributes terms in L_1 and L_2 to their normal forms, performing the unification, and attributing the graph K with the representatives of the equivalence classes of the resulting terms.

Case 3. Finally, we may allow any congruence which can be decided on ground terms, like the equivalence on CSP processes which is checked by the FDR tool [8]. In this case, attributes in the left-hand sides of the rules have to be restricted to ground terms and variables, and to merge two attribute values we may either check their equivalence, if both are ground terms, or apply a substitution, if one is a variable.

The three cases can occur in combinations. In general we may use a different implementation for every sort of the algebra, and each of these implementations determines certain restrictions for the terms in the left-hand sides of rules. It shall be noted that these restrictions are not only relevant to the effective construction of critical pairs, but also to the transformation of graphs attributed over equivalence classes of terms. Here, unification is replaced by pattern matching of terms in the rules with (equivalence classes of) terms in the graphs to be transformed and, depending on the implementation of this equivalence, this pattern matching may be limited to purely syntactic matching in case 1, up to checking for equivalence of ground terms in case 3.

A sample critical pair. In Fig. 6, two conflicting transformations on an overlapping graph of rule directBeh with itself are shown. Note that there are further overlapping graphs of rule directBeh with itself. In the present case, the gluing

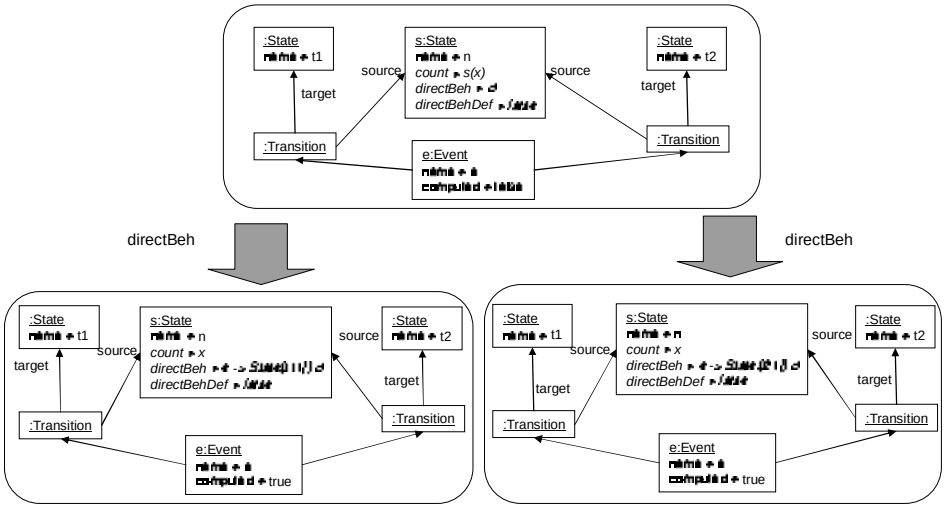

Fig. 6. A critical pair

condition is satisfied as only attribute values are changed by the application of these rules. Furthermore, the transformations are not parallel independent, because the two rules overlap in state s and event e. Attribute *count* is changed (i. e. deleted and created), and therefore this is clearly a critical pair as far as

the graphical structure is concerned. In order to decide whether this extends to the attribute part one has to consider the underlying equivalence relation on terms. In our case, for sort CSPEq, we assume an equivalence of CSP processes like *failures equivalence* of processes, which can be checked by the FDR tool [8], thereby leading to case 3. We further assume for sort Bool that the equivalence relation is trivial (case 1) and for sort Nat a terminating and confluent term rewrite system (case 2). By construction of the overlapping graph, the critical pair candidate is minimal with respect to object vertices. We now compute the attributes of K. Concerning attribute computed of event e and directBehDef of state s, this is clearly the term false. Concerning attribute count, $s(x)$ and $s(y)$ are unified to $s(x)$. Finally, with respect to attribute directBeh, the two terms d and f are unified to d.

Embedding and completeness. Critical pairs cover all possible conflicting situations, which can be obtained by embedding the critical pairs into a larger context and instantiating their attributes. That means, to apply the same rules at essentially the same occurrence in a bigger graph.

Proposition 2 (completeness of critical pairs). *Assume a graph transformation system $GTS = (\Sigma, ATG, X, R)$, a congruence $\equiv \subseteq T_\Sigma(X) \times T_\Sigma(X)$, and two conflicting transformation steps $H_1 \overset{p_1(o_1)}{\Longleftarrow} G \overset{p_2(o_2)}{\Longrightarrow} H_2$ in GTS with $Alg(H_i) = Alg(G) = A$ such that there exists a Σ-homomorphism $m : T_\Sigma(X)/_\equiv \to A$. In this case, there exists a critical pair $P_1 \overset{p_1(k_1)}{\Longleftarrow} K \overset{p_2(k_2)}{\Longrightarrow} P_2$ over $\equiv$ which embeds into the conflicting steps $H_1 \overset{p_1(o_1)}{\Longleftarrow} G \overset{p_2(o_2)}{\Longrightarrow} H_2$.*

Proof. Sketch: Transformations $H_1 \overset{p_1(o_1)}{\Longleftarrow} G \overset{p_2(o_2)}{\Longrightarrow} H_2$ can be replayed on terms of $T_\Sigma(X)/_\equiv$ according to Σ-homomorphism m, since morphism $c : T_\Sigma(X) \to T_\Sigma(X)/_\equiv$ is unique and $m \circ c$ is equal on the algebra part of all occurrence morphisms in the given conflicting steps. Furthermore, composition and decomposition properties for pushouts in category $\frown Graph$ have to be used [4]. The conflicting transformations can be further reduced by cutting off unneeded context such that k_1 and k_2 are jointly surjective. That leads to $P_1 \overset{p_1(k_1)}{\Longleftarrow} K \overset{p_2(k_2)}{\Longrightarrow} P_2$ where $Alg(P_i) = Alg(K) = T_\Sigma(X)/_\equiv$ and $P_i \subseteq H_i$, $K \subseteq G$. $P_1 \overset{p_1(k_1)}{\Longleftarrow} K \overset{p_2(k_2)}{\Longrightarrow} P_2$ is a critical pair due to the minimality of graph and algebra parts.

An embedding theorem [3, 12] answers the question, under which conditions a given transformation sequence $K_0 \overset{*}{\Longrightarrow} K_n$ can be replayed in a bigger context. In the double-pushout approach it is well-known that this is the case whenever the morphism $m : K_0 \to G_0$ satisfies the gluing condition wrt. the *derived rule* $K_0 \curvearrowright K_n$ summarizing the effect of the overall transformation sequence.

Definition 6 (derived production). *Given two spans $s = (G \overset{g}{\longleftarrow} D \overset{h_1}{\longrightarrow} H)$ and $t = (H \overset{h_2}{\longleftarrow} E \overset{i}{\longrightarrow} I)$, their composition $s;\ t = (G \overset{g \circ h_1'}{\longleftarrow} C \overset{i \circ h_2'}{\longrightarrow} I)$ is defined up to isomorphism by the pullback $D \overset{h_1'}{\longleftarrow} C \overset{h_2'}{\longrightarrow} E$ of $D \overset{h_1}{\longrightarrow} H \overset{h_2}{\longleftarrow} E$.*

For a direct transformation $g = G \overset{p(o)}{\Longrightarrow} H$ as in Def. 3, its derived production *$der(g)$ is defined by the bottom span of the DPO diagram $(G \xleftarrow{g} D \xrightarrow{h} H)$. Given a transformation sequence $h = (G_0 \overset{p_1(o_1)}{\Longrightarrow} \ldots \overset{p_n(o_n)}{\Longrightarrow} G_n)$ with $der(h_i)$ the derived production of $h_i = (G_{i-1} \overset{p_1(o_1)}{\Longrightarrow} G_i)$. The derived production of the sequence h is defined as $der(h) = der(h_1); \ldots; der(h_n)$.*

It is obvious that the derived production is properly typed over ATG, if all original productions and graph G_0 are. Note that both morphisms of $der(g)$ are identities on the data algebras and that the derived production is unique up to isomorphism due to the pullback construction. The definition of derived productions can easily be generalized to transformation sequences of length greater than 2.

Proposition 3 (embedding of transformations). *Given a transformation sequence $k = (K_0 \overset{p_1(k_1)}{\Longrightarrow} \cdots \overset{p_n(k_n)}{\Longrightarrow} K_n)$ and a graph morphism $m : K_0 \to G_0$, then there is a transformation sequence $g = (G_0 \overset{p_1(o_1)}{\Longrightarrow} \cdots \overset{p_n(o_n)}{\Longrightarrow} G_n)$ with $o_i = m \circ k_i$, if and only if, $der(k)$ is applicable at m, i.e. there is a transformation $G_0 \overset{der(k)(m)}{\Longrightarrow} G_n$.*

The embedding theorem above has been shown in [16] for $n = 2$, but can be generalized to $n > 2$ straight forward. Moreover, it has been shown for colored graphs. The proof of proposition 3 would be a simple rephrasing for typed graphs.

Confluence. Embedding is relevant in the proof of the critical pair lemma below where it is shown that a graph transformation system GTS is locally confluent if all its critical pairs showing conflicting situations can be joined.

Definition 7 (confluence). *Two transformation sequences $H_1 \overset{*}{\Longleftarrow} G \overset{*}{\Longrightarrow} H_2$ are* confluent *if there are transformation sequences $H_1 \overset{*}{\Longrightarrow} X$ and $H_2 \overset{*}{\Longrightarrow} X$.*

Two transformations $H_1 \overset{p_1}{\Longleftarrow} G \overset{p_2}{\Longrightarrow} H_2$ with $der(G \overset{p_i}{\Longrightarrow} H_i) = (G \xleftarrow{g_i} D_i \xrightarrow{h_i} H_i)$ are strongly confluent *if for $(D_1 \xleftarrow{d_1} D \xrightarrow{d_2} D_2)$ being the pullback of $(D_1 \xrightarrow{g_1} G \xleftarrow{g_2} D_2)$ and $der(G \overset{p_1}{\Longrightarrow} H_i \overset{*}{\Longrightarrow} X) = (G \xleftarrow{c_i} C_i \xrightarrow{x_i} X)$ there are morphisms $e_i : D \to C_i$ with $g_i \circ d_i = e_i \circ c_i$ for $i = 1, 2$.*

A graph transformation system $GTS = (\Sigma, ATG, X, R)$ is confluent w.r.t. a congruence $\equiv \subseteq T_\Sigma(X) \times T_\Sigma(X)$ *if for any Σ-algebra A satisfying this congruence,[2] all pairs of transformations $H_1 \overset{*}{\Longleftarrow} G \overset{*}{\Longrightarrow} H_2$ in GTS attributed over A are confluent. GTS is* locally confluent w.r.t. *A if the same holds for all pairs of the form $H_1 \overset{p_1}{\Longleftarrow} G \overset{p_2}{\Longrightarrow} H_2$.*

Strong confluence means that those graph objects preserved by transformations $G \Longrightarrow H_i$, are not deleted by transformations $H_i \overset{*}{\Longrightarrow} X$ for $i = 1, 2$. The congruence $\equiv$ plays the role of an equational specification for the data algebra A. However, it also covers cases where the congruence is not given in terms of equations, or where it is not equationally axiomatizable.

[2] An algebra A satisfies a congruence over terms with variables in X if for all assignments $\alpha : X \to A$, the free extension to $\bar{\alpha} : T_\Sigma(X)/_\equiv \to A$ is a homomorphism.

Confluence of a sample critical pair. The critical pair in Fig. 6 is not confluent: Assume that first the left hand rule application of directBeh is followed, setting attribute computed of the event to true. Thereby, any further application of rule directBeh to this event is not possible and as directBeh is the only rule adding term $e \to State(t2)$, there is no possibility of joining the two rules. Hence, we have found that our rule set is not confluent. This critical pair can be made confluent if we move attribute computed to the transitions.

Proposition 4 (critical pair lemma). *A graph transformation system $GTS = (\Sigma, ATG, X, R)$ is locally confluent w.r.t. a congruence $\equiv \subseteq T_\Sigma(X) \times T_\Sigma(X)$ if, for all pairs of rules p_1, p_2 in R, each critical pair $P_1 \overset{p_1(k_1)}{\Longleftarrow} K \overset{p_2(k_2)}{\Longrightarrow} P_2$ over $\equiv$ is strongly confluent.*

Proof. Consider two direct transformations $H_1 \overset{p_1(o_1)}{\Longleftarrow} G \overset{p_2(o_2)}{\Longrightarrow} H_2$ with rules $p_i = L_i \curvearrowright R_i$ for $i = 1, 2$. There are the following cases:

1. $H_1 \overset{p_1(o_1)}{\Longleftarrow} G \overset{p_2(o_2)}{\Longrightarrow} H_2$ is parallel independent. Thus, there are transformations $H_1 \overset{p_1}{\Longrightarrow} X$ and $H_2 \overset{p_2}{\Longrightarrow} X$ due to Proposition 1.
2. $H_1 \overset{p_1}{\Longleftarrow} G \overset{p_2}{\Longrightarrow} H_2$ is not parallel independent. According to Proposition 2 there is a critical pair $P_1 \overset{p_1(k_1)}{\Longleftarrow} K \overset{p_2(k_2)}{\Longrightarrow} P_2$ which embeds into $H_1 \overset{p_1}{\Longleftarrow} G \overset{p_2}{\Longrightarrow} H_2$ with morphisms $g : K \to G$ and $h_i : P_i \to H_i$ for $i = 1, 2$. Assuming that all critical pairs are confluent, there are two transformation sequences $K \Longrightarrow P_1 \Longrightarrow X$ and $K \Longrightarrow P_2 \Longrightarrow X$.
   Let $der(K \Longrightarrow P_i) = (K \overset{d_i}{\longleftarrow} D_i \overset{p_i}{\longrightarrow} P_i)$, $der(K \Longrightarrow P_i \Longrightarrow X) = (K \overset{c_i}{\longleftarrow} C_i \overset{x_i}{\longrightarrow} X)$, and $(D_1 \overset{dd_1}{\longleftarrow} D \overset{dd_2}{\longrightarrow} D_2)$ being the pullback of $(D_1 \overset{d_1}{\longrightarrow} K \overset{d_2}{\longleftarrow} D_2)$. Due to strong confluence and the construction of derived productions, there are morphisms $e_i : D \to C_i$ with $c_i \circ e_i = d_i \circ dd_i$ and $f_i : C_i \to D_i$. Furthermore, let *Boundary* be all nodes of K being in touch with edges of $G - K$. $der(G \Longrightarrow H_i)$ and $der(K \Longrightarrow P_i)$ contain[3] *Boundary*, since the gluing condition is satisfied for these transformations. Thus, there is a morphism *Boundary* $\to D$ due to pullback properties and $der(K \Longrightarrow P_i \Longrightarrow X)$ contain *Boundary*, i.e. it satisfies the gluing condition. By Proposition 3 there are transformations $G \overset{p_1(o_i')}{\Longrightarrow} H_i' \Longrightarrow Y$ with $o_i' = g \circ k_i$, thus $o_i' = o_i$ and hence H_i' is isomorphic to H_i.

As local confluence and termination imply confluence according to Newman's lemma [15], confluence of a terminating consistent graph transformation system can be shown by proving for all critical pairs the property of being confluent.

Restricting to vertex-preserving transformations and morphisms which are injective up to data vertices, the gluing condition is always satisfied for any transformation. Of course, this provides us with unrestricted embedding. Moreover in this case, confluence means always strong confluence. This restricted kind of transformations is the only one used in the running example. Thus, the critical pair lemma can easily be used.

[3] A production contains a graph G if there is a morphism $m : G \to K$.

Confluence of the example GTS. Computing critical pair candidates can be done by constructing all overlapping graphs of left hand sides of all combinations of rule pairs. In the following, we will check the example rule set for critical pairs.

Note that in this case the graph transformation system is vertex-preserving.

We first recall that so far the cardinality constraints specified in Fig. 2 have not been part of the formal treatment. However, such cardinality constraints can be seen as *negative constraints* as they require a certain structure not to be existent (i. e. that an object has more than one link to another object). The given graph transformation system is preserving these constraints because it only changes attribute values.

As a consequence, we do not have to show confluence for those critical pairs that do not have an overlapping graph fulfilling all negative constraints because those critical pairs will never be subgraphs of G. This simplifies tremendously the following discussion of critical pairs. However, it is important to note that in the case of positive constraints the set of critical pairs cannot be reduced to those where the overlapping graph fulfills the constraints. In order to show confluence of the complete rule set, we have to compute all critical pairs and show that each critical pair is confluent.

We first note that critical pairs only occur if two rules change the same attributes. Rule top changes only attributes of the state machine which are not changed or read by any other rule. Due to the constraints there is only one top state and therefore top cannot be overlapped with itself in the top state. Rules simple and comp change attributes exp and expDef, which are only changed by comp as well. However, there is no overlapping graph because a state cannot be a composite and simple state at the same time. Both rules extBehtop and extBehcomp change attributes extBeh and extBehDef. Due to the constraint that the top vertex is not a subvertex of any other state, there exists no overlapping graph. Rules directBeh and directBehe do not overlap because $s(x)$ cannot be unified with 0. Overlapping directBeh with itself at their transitions or target states is confluent (under the assumption that the attribute computed is shifted to the Transition class).

Tool support The complexity of computing critical pairs and proving their joinability arises the need for tool support. Currently, AGG [7] supports the computation of critical pairs for attributed graphs: All possible overlapping graphs are constructed and two rules are critical if they change the same attribute.

The attributed graph transformation implemented in AGG allows variables and constants in left-hand rules sides only. But this is not a restriction, since attribute conditions can be stated separately. However, unification on attribute values is not supported.

Furthermore, proving confluence by showing that all critical pairs are confluent also requires support. Here, an interactive approach that enables stepwise rule applications to a common successor graph could be followed, thereby avoiding the complexity of automated derivations.

Currently, AGG does not support type graphs and constraints. An additional possibility to specify graph constraints which can be used to check graphs and

to provide rules with post conditions such that consistent transformations are performed only, is under development. Having graph constraints available, the critical pair analysis can be made more efficient in the sense, that only those overlapping graphs are computed which satisfy the negative constraints. For most application we can expect that the set of critical pairs will become considerably smaller.

4 Conclusion

Confluence and termination of attributed graph transformation are important issues whenever attributed graph transformation is to be used in an automated way. In this paper, we have shown how confluence can be ensured for typed attributed graph transformation systems. Motivated by the translation of UML statecharts to CSP, typed attributed graph transformation systems have been introduced in order to represent transformations of diagrams based on a meta model-like representation. Then, the theory of critical pairs has been extended to typed attributed graph transformation systems. We have shown that an attributed graph transformation system is locally confluent if all its critical pairs are confluent. The concept of critical pairs has been applied to a concrete set of translation rules, thereby discovering an error leading to a non-confluent critical pair. Moreover, the issue of tool support based on AGG has been sketched.

Having now the critical pairs analysis technique for attributed graph transformation at hand, also the other applications mentioned in the introduction already, can be checked for functional behavior in future. In [9], functional requirements are described by UML use cases refined by activity and collaboration diagrams. Using graph transformation as semantic domain here, graph rules formalism the functional requirements to a system. The critical pair analysis can be used to find out conflicts and dependencies between different use cases.

Another application of graph transformation where functional behavior is of importance, is parsing of visual diagrams. Allowing free editing of visual diagrams, they have to be parsed to be sure that they belong to some visual language. Analyzing the critical pairs of parsing rules, and applying conflict-free rules first, increases the efficiency of graph parsing [1].

References

1. P. Bottoni, A. Schürr, and G. Taentzer. Efficient Parsing of Visual Languages based on Critical Pair Analysis and Contextual Layered Graph Transformation. In *Proc. IEEE Symposium on Visual Languages*, September 2000. Long version available as technical report SI-2000-06, University of Rom.
2. A. Corradini, U. Montanari, and F. Rossi. Graph processes. In *Fundamenta Informaticae*, volume 26 (3,4), pages 241–266, 1996.
3. H. Ehrig. Embedding theorems in the algebraic theory of graph grammars. In *LNCS 56*, pages 245–255. Springer, 1977.

4. H. Ehrig. Introduction to the Algebraic Theory of Graph Grammars (A Survey). In *Graph Grammars and their Application to Computer Science and Biology*. Springer LNCS 73, 1979.
5. H. Ehrig, M. Pfender, and H.J. Schneider. Graph grammars: an algebraic approach. In *14th Annual IEEE Symposium on Switching and Automata Theory*, pages 167–180. IEEE, 1973.
6. G. Engels, R. Heckel, and J. M. Küster. Rule-based specification of behavioral consistency based on the UML meta-model. In M. Gogolla and C. Kobryn, editors, *Proc. 4th Intl. Conference on The Unified Modeling Language (UML '02), Toronto, Canada, October, 2001*, volume 2185 of *LNCS*, pages 272–287. Springer, 2001.
7. C. Ermel, M. Rudolf, and G. Taentzer. The AGG-Approach: Language and Tool Environment. In H. Ehrig, G. Engels, H.-J. Kreowski, and G. Rozenberg, editors, *Handbook of Graph Grammars and Computing by Graph Transformation, volume 2: Applications, Languages and Tools*, pages 551–603. World Scientific, 1999. See also `http://tfs.cs.tu/berlin.de/agg`.
8. Formal Systems Europe (Ltd). *Failures-Divergence-Refinement: FDR2 User Manual*, 1997.
9. J. H. Hausmann, R. Heckel, and G. Taentzer. Detection of Conflicting Functional Requirements in a Use Case-Driven Approach. In *Proc. 24th Intl. Conference on Software Engineering*, Orlando, FL, 2002. ACM/IEEE Computer Society.
10. R. Heckel, J.M. Küster, and G. Taentzer. Towards automatic translation of UML models into semantic domains. In H.-J. Kreowski, editor, *Proc. ETAPS'02 Workshop on Application of Graph Transformation (AGT'02), Grenoble, France*, April 2002.
11. C. A. R. Hoare. *Communicating Sequential Processes*. Prentice Hall, 1985.
12. H.-J. Kreowski. *Manipulation von Graphmanipulationen*. PhD thesis, FB13, 1978.
13. S. Kuske. A formal semantics of UML state machines based on structured graph transformation. In M. Gogolla and C. Kobryn, editors, *Proc. UML 2001, Toronto, Kanada*, volume 2185 of *LNCS*. Springer-Verlag, 2001.
14. M. Löwe, M. Korff, and A. Wagner. An algebraic framework for the transformation of attributed graphs. In *Term Graph Rewriting: Theory and Practice*, pages 185–199. John Wiley & Sons Ltd, 1993.
15. M. H. A. Newman. On theories with a combinatorial definition of 'equivalence'. In *Annals of Mathematics*, 43 (2), pages 223–243, 1942.
16. J. Padberg and G. Taentzer. Embedding of derivations in high-level replacement systems. Technical Report 93/9, Technical University of Berlin, Computer Science Department, 1993.
17. D. Plump. Hypergraph Rewriting: Critical Pairs and Undecidability of Confluence. In M.R Sleep, M.J. Plasmeijer, and M. C.J.D. van Eekelen, editors, *Term Graph Rewriting*, pages 201–214. Wiley, 1993.
18. D. Plump. Term graph rewriting. In G. Engels, H.-J. Kreowski, and G. Rozenberg, editors, *Handbook of Graph Grammars and Computing by Graph Transformation, Volume 2: Applications, Languages, and Tools*, pages 3 – 62. World Scientific, 1999.
19. D. Varro, G. Varro, and A. Pataricza. Designing the Automatic Transformation of Visual Languages. *Science of Computer Programming*, 44(2), 2002.