Available online at www.sciencedirect.com
SCIENCE DIRECT®
ELSEVIER
Electronic Notes in Theoretical Computer Science 152 (2006) 97–109
Electronic Notes in Theoretical Computer Science
www.elsevier.com/locate/entcs

Efficient Detection of Conflicts in Graph-based Model Transformation

Leen Lambers , Hartmut Ehrig [2,1]

Institut für Softwaretechnik und Theoretische Informatik
Technische Universität Berlin
Germany

Fernando Orejas [3]

Dept. L.S.I.
Tech. Univ. Catalonia
Barcelona, Spain

Abstract

Using graph transformation as a formalism to specify model transformation, termination and confluence of the graph transformation system are often required properties. Only under these two conditions, existence and uniqueness of the outcoming model is ensured. Verifying confluence of a graph transformation system can be reduced to checking both local confluence and termination.
A graph transformation system is locally confluent, if all its conflicts in a minimal context can be resolved. Formally, this means, that all critical pairs of the graph transformation system should be strictly confluent. Thus, answering the question of local confluence of a graph transformation system, requires the following two steps. At first the computation of all critical pairs is necessary. Secondly this set of critical pairs has to be tested for strict confluence. This paper concentrates on the first step, proposing an efficient method to compute the set of all critical pairs of a given graph transformation system. Efficiency is obtained because of the following two main reasons. At first all pairs of rules are analyzed to check if they can actually cause a conflict. Then for each conflict inducing pair of rules, the set of critical pairs is computed in a constructive way, avoiding needless computations.
The overall goal of this paper is to encourage tool development and enhancement concerning the detection of conflicts in a graph transformation system. It should be an aid to translate the main theoretical results concerning confluence of a graph transformation system into practical methods for the analysis of model transformations specified with graph transformation.

Keywords: conflict, confluence, critical pair, model transformation, graph transformation

1571-0661/$ – see front matter
doi:10.1016/j.entcs.2006.01.017

1 Introduction

A graph transformation system (gts) exhibits functional behavior if it is terminating and confluent. This is an important feature if graph transformation is used, to specify model transformation [3]. Imagine, that we use a confluent and terminating gts $\mathcal{S}$, to translate one model into another one. In praxis, some graph transformation tool executes the model transformation. Despite the nondeterministic rule application, the transformation has the following two characteristics. On the one hand, we can be sure, that the outcoming model exists, since the gts $\mathcal{S}$ is terminating. On the other hand, the outcoming model is unique, because of confluency of $\mathcal{S}$. Thus, each run of $\mathcal{S}$ to an input model yields one specific output model. Formally, this means, that $\mathcal{S}$ operates as a function $\mathcal{S}$: $M_1 \to M_2$ on the set M_1 of input models, generating a unique element of M_2, the set of output models of the model transformation. How to ensure termination of a gts is described in [4]. An established result in term rewriting [8] is, that if a rewrite system is locally confluent and terminating, it is also confluent. So the question is, how to ensure local confluence of a gts. In [6], the so called critical pair lemma states, that a gts is locally confluent if all its critical pairs are strictly confluent. Critical pairs are conflict situations of the graph transformation system in a minimal context. This paper is concerned with the efficient detection of these critical pairs. Once detected, they should be checked for strict confluence to conclude local confluence of the gts.

Efficient conflict detection is obtained, because of the following two reasons. At first, rules are analyzed, to check if they can actually cause a conflict. Secondly, the minimal context of each conflict is computed in a constructive way. Therefore needless computations, checking for a conflict in each possible minimal context, are avoided. AGG [10] is a graph transformation tool, which allows an automatic detection of the set of critical pairs of a gts. Implementing the optimizations, explained in this paper, should improve its efficiency. Moreover an algorithm is recommended for other graph transformation tool developers, with the intention to implement critical pair detection.

We begin this paper with some definitions for the used graph transformation approach [2]. Then the concept of critical pair and conflict is rehearsed. The main part of this paper is concerned with explaining how to compute in an efficient way the set of all critical pairs of a gts.

[1] Email: ehrig@cs.tu-berlin.de
[2] Email: leen@cs.tu-berlin.de
[3] Email: orejas@lsi.upc.edu

2 Graph Transformation

The theory of confluence and critical pairs has been worked out for different graph transformation approaches [9]. This paper explains how to apply the theory of confluence and critical pairs, developed for graph transformation in the double pushout approach [6]. Thus in this paragraph we shortly repeat the main definitions.

Definition 2.1 [graph and graph morphism] A *graph* $G = (G_E, G_V, s, t)$ consists of a set G_E of edges, a set G_V of vertices and two mappings $s, t : G_E \to G_V$, assigning to each edge $e \in G_E$ a source $q = s(e) \in G_V$ and target $z = t(e) \in G_V$. A *graph morphism* $f : G_1 \to G_2$ between two graphs $G_i = (G_{i,E}, G_{i,V}, s_i, t_i)$, $(i = 1, 2)$ is a pair $f = (f_E : G_{E,1} \to G_{E,2}, f_V : G_{V,1} \to G_{V,2})$ of mappings, such that $f_V \circ s_1 = s_2 \circ f_E$ and $f_V \circ t_1 = t_2 \circ f_E$. A graph morphism $f : G_1 \to G_2$ is injective (resp.surjective) if f_V and f_E are injective (resp. surjective) mappings. A *graph isomorphism* is an injective and surjective graph morphism. A *graph inclusion* $i : H \to G$ is a graph morphism, with $i_V : v \mapsto v$ and $i_E : e \mapsto e$. H is a subgraph of G if there exists a graph inclusion $i : H \to G$. Two graph morphisms $m_1 : L_1 \to G$ and $m_2 : L_2 \to G$ are *jointly surjective* if $m_{1,V}(L_{1,V}) \cup m_{2,V}(L_{2,V}) = G_V$ and $m_{1,E}(L_{1,E}) \cup m_{2,E}(L_{2,E}) = G_E$. A pair of jointly surjective morphisms (m_1, m_2) is also called an *overlapping* of L_1 and L_2. A *graph projection* p is a graph morphism from a subgraph G of $G_1 \times G_2$ into G_1 (resp. G_2) with $p_V : G_V \to G_{1,V}$ (resp. $G_{2,V}$) and $p_E : G_E \to G_{1,E}$ (resp. $G_{2,E}$) projections. **Remark:** Throughout this paper often f is used, instead of explicitly writing f_V and/or f_E.

Definition 2.2 [category **Graph**] The category having graphs as objects and graph morphisms as arrows is called **Graph**.

Definition 2.3 [rule] A graph transformation rule $p : L \stackrel{l}{\leftarrow} K \stackrel{r}{\rightarrow} R$ consists of a rule name p and a pair of injective graph morphisms $l : K \to L$ and $r : K \to R$. The graphs L, K and R are called the left-hand side (lhs), the interface, and the right-hand side (rhs) of p, respectively. The rule p is non-deleting if $l : K \to L$ is an isomorphism. **Remark:** If l is a graph inclusion, then a rule is non-deleting if $L = K$.

Definition 2.4 [graph transformation system] A graph transformation system consists of a set of rules $(p : L \stackrel{l}{\leftarrow} K \stackrel{r}{\rightarrow} R)_{p \in \mathcal{P}}$ with $\mathcal{P}$ the set of rule names.

Definition 2.5 [match] Given a rule $p : L \stackrel{l}{\leftarrow} K \stackrel{r}{\rightarrow} R$ and a graph G, one can try to apply p to G if there is an occurence of L in G i.e. an injective graph

morphism, called match $m : L \to G$. **Remark:** In general a match doesn't have to be injective. Here we restrict to injective matches.

Definition 2.6 [direct graph transformation] Given a graph G, a rule $p : L \stackrel{l}{\leftarrow} K \stackrel{r}{\rightarrow} R$ and a match $m : L \to G$, a direct graph transformation from G to H using p exists if and only if the double pushout (DPO) diagram

$$\begin{array}{ccccc} L & \stackrel{l}{\longleftarrow} & K & \stackrel{r}{\longrightarrow} & R \\ {\scriptstyle m}\downarrow & & \downarrow & & \downarrow \\ G & \longleftarrow & D & \longrightarrow & H \end{array}$$

can be constructed. In this case we write $G \stackrel{p,m}{\Rightarrow} H$. Since pushouts in **Graph** always exist, the DPO can be constructed if the pushout complement of $K \to L \to G$ exists. If so, we say that, the match m satisfies the gluing condition of rule p. Note, that since a match in this paper is injective, the identification condition is always fullfilled.

Definition 2.7 [graph transformation] A graph transformation over a graph transformation system $\mathcal{G}$ is either a graph G, or a sequence of direct graph transformations $G_{i-1} \stackrel{p_i}{\Rightarrow} G_i$, with p_i a rule in $\mathcal{G}$.

3 Conflicts and Critical Pairs

Given a graph G, we may have several rules that can be applied to G. However, this situation is not necessarily a conflictive one. In particular if we have two rules $p_1 : L_1 \stackrel{l_1}{\leftarrow} K_1 \stackrel{r_1}{\rightarrow} R_1$ and $p_2 : L_2 \stackrel{l_2}{\leftarrow} K_2 \stackrel{r_2}{\rightarrow} R_2$ such that they can both be applied to G via the matches m_1 and m_2, the situation is not a conflict if, after applying any of the rules, we can still apply the other one, i.e. if the transformation defined by the former does not destroy the application of the latter. The following definitions characterize this situation:

Definition 3.1 [parallel independence] Two direct transformations $G \stackrel{(p_1,m_1)}{\Longrightarrow} H_1$ and $G \stackrel{(p_2,m_2)}{\Longrightarrow} H_2$ are parallel independent if

$$m_1(L_1) \cap m_2(L_2) \subseteq m_1(l_1(K_1)) \cap m_2(l_2(K_2))$$

This condition can be expressed categorically in the following way:

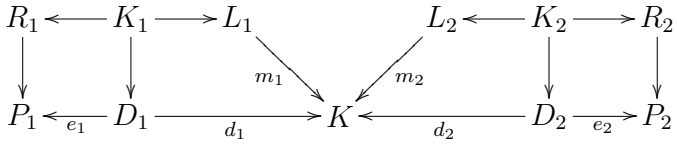
$$\exists h_1 : L_1 \to D_2 : d_2 \circ h_1 = m_1 \wedge \exists h_2 : L_2 \to D_1 : d_1 \circ h_2 = m_2$$

$$\begin{array}{ccccccccccc} R_1 & \longleftarrow & K_1 & \longrightarrow & L_1 & & & & L_2 & \longleftarrow & K_2 & \longrightarrow & R_2 \\ \downarrow & & \downarrow & & & {\scriptstyle h_2}\searrow & & \swarrow{\scriptstyle h_1} & & & \downarrow & & \downarrow \\ & & & & & & {\scriptstyle m_1}\searrow & \swarrow{\scriptstyle m_2} & & & & & \\ H_1 & \stackrel{e_1}{\longleftarrow} & D_1 & & & & \stackrel{d_1}{\longrightarrow} & & G & \stackrel{d_2}{\longleftarrow} & D_2 & \stackrel{e_2}{\longrightarrow} & H_2 \end{array}$$

Definition 3.2 [conflict] Two direct transformations $G \overset{(p_1,m_1)}{\Rightarrow} H_1$ and $G \overset{(p_2,m_2)}{\Rightarrow} H_2$ are in conflict if they are not parallel independent. **Remark:** This type of conflict is also called delete-use-conflict. In particular rule p_2 deletes something, what p_1 uses if $m_1(L_1) \cap m_2(L_2) \not\subseteq m_2(l_2(K_2))$ and/or p_1 deletes something, what p_2 uses if $m_1(L_1) \cap m_2(L_2) \not\subseteq m_1(l_1(K_1))$.

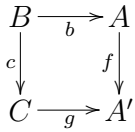
A minimal (in the sense that the graphs G considered are as small as possible) conflict situation can be characterized by the notion of critical pair:

Definition 3.3 [critical pair] A critical pair is a pair of direct transformations $K \overset{(p_1,m_1)}{\Rightarrow} P_1$ and $K \overset{(p_2,m_2)}{\Rightarrow} P_2$ in conflict, s.t. m_1 and m_2 are jointly surjective morphisms.



Two notions that are important for the rest of the paper are the concepts of boundary and context introduced in [5]:

Definition 3.4 [boundary - context] The boundary B of an injective graph morphism $f : A \to A'$ consists of all nodes $a \in A$ such that $f(a)$ is adjacent to an edge in $A' \setminus f(A)$. The context $C = A' \setminus f(A) \cup f(b(B))$ can be glued to A over the boundary B obtaining the pushout object A'. This situation is expressed by the following pushout, called boundary pushout with b, c and g graph inclusions.



Remark: As described in [5] the boundary pushout is an inital pushout.

4 Efficient Conflict Detection

Looking at the critical pair definition, a straightforward way of computing the set of critical pairs of two rules $p_1 : L_1 \leftarrow K_1 \rightarrow R_1$ and $p_2 : L_2 \leftarrow K_2 \rightarrow R_2$ is expressed by the following algorithm:

```
CP = empty;
compute all jointly surjective m1:L1->K and m2:L2->K;
for each pair (m1,m2)
```

```
  if (m1 and m2 fullfill gluing condition)
      compute (T1,T2) induced by (p1,m1) and (p2,m2);
      if (T1 and T2 are in conflict)
        CP = CP U {(T1,T2)};
return CP;
```

This way of computing can be made more efficiently because of the following reasons.

(i) It is not necessary to compute *any* overlapping (m_1, m_2) of L_1 and L_2 if both rules are non-deleting.

(ii) If one of the rules is non-deleting and the other one is deleting, it is not necessary to compute *all* overlappings (m_1, m_2) of L_1 and L_2. It is enough to compute directly those overlappings which will lead to a critical pair.

The next paragraphs explain these two optimization steps in more detail and show their correctness.

4.1 1st Optimization

In the first optimization step we need to check if rules are deleting or non-deleting. In that way we can avoid computing overlappings of two non-deleting rules. This would be redundant since the set of critical pairs of two non-deleting rules is empty, as shown in the following lemma.

Lemma 4.1 *Given two non-deleting rules* $p_1 : L_1 \overset{l_1}{\leftarrow} K_1 \overset{r_1}{\rightarrow} R_1$ *and* $p_2 : L_2 \overset{l_2}{\leftarrow} K_2 \overset{r_2}{\rightarrow} R_2$*, each pair of direct graph transformations* $H_1 \overset{p_1,m_1}{\Leftarrow} G \overset{p_2,m_2}{\Rightarrow} H_2$ *is parallel independent.*

Proof. We have to prove, that $m_1(L_1) \cap m_2(L_2) \subseteq m_1(l_1(K_1)) \cap m_2(l_2(K_2))$. Since for non-deleting rules $l_1 : K_1 \rightarrow L_1$ and $l_2 : K_2 \rightarrow L_2$ are isomorphisms, because of surjectivity $l_1(K_1) = L_1$ and $l_2(K_2) = L_2$. Thus, $m_1(L_1) \cap m_2(L_2) \subseteq m_1(l_1(K_1)) \cap m_2(l_2(K_2)) = m_1(L_1) \cap m_2(L_2)$ holds. □

A rule $p : L \overset{l}{\leftarrow} K \overset{r}{\rightarrow} R$ is non-deleting if l is an isomorphism. There are several ways to check this, depending on how graphs and graph morphisms are stored in the relative system. The following lemma proves, that checking if l is an isomorphism actually corresponds to checking if the context C of l is equal to the empty graph. This method is preferred here, since the context graph C can be reused in the algorithm for detecting the critical pairs. The 2nd optimization in the following paragraph explains why. If graphs are stored by their adjaceny matrix, the computation of the context graph C of l is of quadratic complexity in the number of nodes and edges in L.

Lemma 4.2 *A rule $L \stackrel{l}{\leftarrow} K \stackrel{r}{\rightarrow} R$ is a non-deleting rule if and only if the context C of $l : K \rightarrow L$ is equal to the empty graph.*

$$\begin{array}{ccccc} C & \stackrel{c}{\longleftarrow} & B & & \\ {\scriptstyle g}\downarrow & & \downarrow{\scriptstyle b} & & \\ L & \stackrel{l}{\longleftarrow} & K & \longrightarrow & R \end{array}$$

Proof.

- Since $C = L \setminus l(K) \cup l(b(B))$ is equal to the empty graph, $L \setminus l(K)$ is also equal to the empty graph. Therefore $L = l(K)$ and since l is also an injective graph morphism, we can conclude, that l is an isomorphism.
- If l is an isomorphism, we know, that $L = l(K)$ or $L \setminus l(K)$ is equal to the empty graph. Therefore, also the boundary B is equal to the empty graph. Conclusion: $C = L \setminus l(K) \cup l(b(B))$ is equal to the empty graph.

□

4.2 2nd Optimization

After detecting the deleting rules we keep only those pairs of rules, with one non-deleting and one deleting rule. For such a pair of rules, it is possible to compute only those overlappings, which lead to a critical pair. This can be achieved by identifying in the overlapping at least one element, which is deleted by the deleting rule, with an element of the lhs of the other rule. The construction of such a special overlapping is formulated in detail in the following definition and theorem. Moreover in the theorem it is shown, that this optimization is correct. This means, that computing only this kind of special overlappings, leads us anyhow to the set of all critical pairs.

Definition 4.3 [conflict conditions - compatibility conditions] Given a non-deleting rule $p_1 : L_1 \stackrel{l_1}{\leftarrow} K_1 \stackrel{r_1}{\rightarrow} R_1$ and a rule $p_2 : L_2 \stackrel{l_2}{\leftarrow} K_2 \stackrel{r_2}{\rightarrow} R_2$, with B_2 the boundary, C_2 the context of $l_2 : L_2 \leftarrow K_2$

- a graph S and two morphisms $o_1 : S \rightarrow L_1, o_2 : S \rightarrow C_2$ satisfy the *conflict conditions* if and only if S is a minimal (i.e. there doesn't exist a graph S', satisfying the following conditions and being a real subgraph of S) graph such that the following holds:

(i) S is a subgraph of $L_1 \times C_2$, with injective projections $o_1 : S \rightarrow L_1$ and $o_2 : S \rightarrow C_2$

(ii) $o_2(S)$ is not a subgraph of B_2

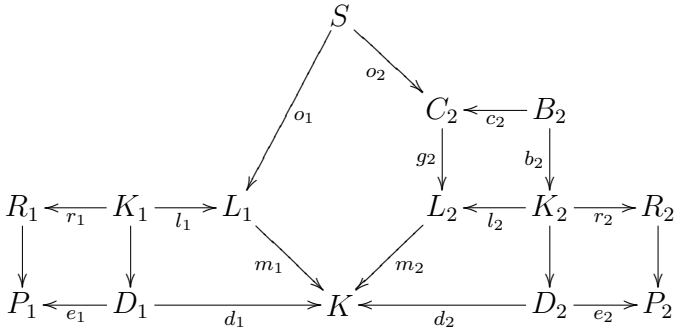
- two morphisms $m_1 : L_1 \rightarrow K$ and $m_2 : L_2 \rightarrow K$ satisfy the *compatibility conditions* with respect to $(S, o_1 : S \rightarrow L_1, o_2 : S \rightarrow C_2)$ if and only if

(i) m_1 and m_2 are jointly surjective

(ii) $m_1 \circ o_1 = m_2 \circ g_2 \circ o_2$
(iii) m_1 (resp. m_2) satisfies the gluing condition of p_1 (resp. p_2)

Theorem 4.4 *Given a non-deleting rule $p_1 : L_1 \overset{l_1}{\leftarrow} K_1 \overset{r_1}{\rightarrow} R_1$ and a rule $p_2 : L_2 \overset{l_2}{\leftarrow} K_2 \overset{r_2}{\rightarrow} R_2$, the following holds:*

- *Each (S, o_1, o_2), satisfying the conflict conditions and each pair of graph morphisms $m_1 : L_1 \to K$ and $m_2 : L_2 \to K$, satisfying the compatibility conditions with respect to (S, o_1, o_2) gives rise to a critical pair $P_1 \overset{p_1,m_1}{\Leftarrow} K \overset{p_2,m_2}{\Rightarrow} P_2$.*

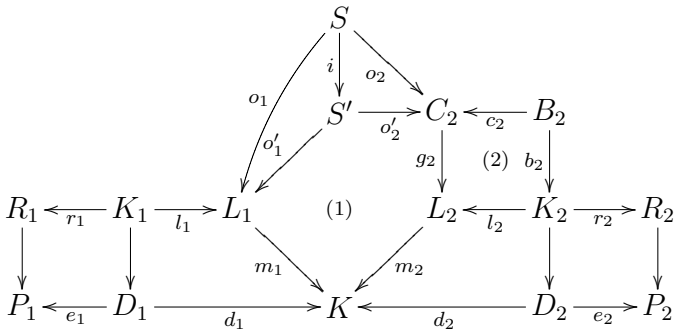


- *For each critical pair $P_1 \overset{p_1,m_1}{\Leftarrow} K \overset{p_2,m_2}{\Rightarrow} P_2$ there exists a graph S and morphisms $o_1 : S \to L_1, o_2 : S \to C_2$ such that (S, o_1, o_2) satisfies the conflict conditions and the matches $m_1 : L_1 \to K$ and $m_2 : L_2 \to K$ satisfy the compatibility conditions with respect to (S, o_1, o_2).*

Proof.

- Because of the 3rd compatibility condition, the pair of direct transformations $L_1 \overset{p_1,m_1}{\Rightarrow} P_1$ and $L_2 \overset{p_2,m_2}{\Rightarrow} P_2$ exists. The 1st compatibility condition ensures, that m_1 and m_2 are jointly surjective, thus it suffices to show that $m_1(L_1) \cap m_2(L_2) \not\subseteq m_2(l_2(K_2))$ to prove that the direct transformations $L_1 \overset{p_1,m_1}{\Rightarrow} P_1$ and $L_2 \overset{p_2,m_2}{\Rightarrow} P_2$ are in conflict and to conclude that they form a critical pair. So we should find an element $y \in m_1(L_1) \cap m_2(L_2)$ s.t. $y \notin m_2(l_2(K_2))$. Because of the conflict conditions of S, there exists an x in S_V or S_E such that $o_2(x) \in C_2 \setminus c_2(B_2) = C_2 \setminus B_2$. This is, because otherwise $o_2(S)$ would be a subgraph of B_2. Because of $o_2(x) \in C_2 \setminus B_2$ and the fact that (2) is a pushout, $g_2(o_2(x)) \notin l_2(K_2)$. Since m_2 is injective, also $m_2(g_2(o_2(x))) \notin m_2(l_2(K_2))$. The 2nd compatibility condition says, that $\forall x \in S : m_1(o_1(x)) = m_2(g_2(o_2(x)))$. Thus, we have found an $y = m_2(g_2(o_2(x))) \in m_1(L_1) \cap m_2(L_2)$ s.t. $y \notin m_2(l_2(K_2))$.
- At first we can build the pullback of m_1 and $m_2 \circ g_2$, obtaining a pullback object S' subgraph of $L_1 \times C_2$ and induced injective graph projections o'_1 :

$S' \to L_1$ and $o'_2 : S' \to C_2$.



Since the critical pair consists of two direct transformations in conflict and p_1 is a non-deleting rule, we know that, there exists an $y \in m_1(L_1) \cap m_2(L_2)$, such that $y \notin m_2(l_2(K_2))$. This is, because since $m_1(L_1) \cap m_2(L_2) \not\subseteq m_1(l_1(K_1)) \cap m_2(l_2(K_2)))$ and since l_1 is an isomorphism each $y \in m_1(L_1)$ holds a preimage x in L_1 and each x in L_1 holds a preimage z in K_1. This means, that $m_1(L_1) \cap m_2(L_2) \subseteq m_1(l_1(K_1))$. Then, it follows that $m_1(L_1) \cap m_2(L_2) \not\subseteq m_2(l_2(K_2))$. Thus, there exist $x_1 \in L_1$ and $x_2 \in L_2$ s.t. $m_2(x_2) = y = m_1(x_1)$, but there exists no $k \in K_2$ s.t. $l_2(k) = x_2$. Because of this and the fact that (2) is a pushout, there exists $c \in C_2 \setminus B_2$ s.t. $g_2(c) = x_2$. Thus, we have that $m_1(x_1) = m_2(x_2) = m_2(g_2(c))$. Since (1) is a pullback, this means, that S' is not equal to the empty graph. In fact, there exists an $s \in S'$ s.t. $o'_2(s) = c$ and $o'_1(s) = x_1$. Since $o'_2(s) = c$ and $c \in C_2 \setminus B_2$, $o'_2(S')$ can't be a subgraph of B_2. Thus (S', o'_1, o'_2) fullfills the conflict conditions if S' in addition would be minimal. If it is not, then we take a minimal subgraph S of S', with projections $o_1 : S \to L_1$ and $o_2 : S \to C_2$ s.t. $o_2(S)$ is not a subgraph of B_2. Then (S, o_1, o_2) satisfies the conflict conditions. The matches (m_1, m_2) of the critical pair are by definition jointly surjective and they fullfill the gluing condition of p_1 resp. p_2. Since (1) is a pullback and S is a subgraph of S', $m_1 \circ o_1 = m_2 \circ g_2 \circ o_2$. Thus (m_1, m_2) fullfill the compatibility conditions with respect to (S, o_1, o_2). □

5 Critical Pair Algorithm

The optimization steps in the last paragraph help us to compute in an efficient way the set of critical pairs of two non-deleting rules or of one deleting and one non-deleting rule. The following algorithm computes the set of critical pairs $CP = CP_1 \cup CP_2$, induced by the rules $p_1 : L_1 \overset{l_1}{\leftarrow} K_1 \overset{r_1}{\rightarrow} R_1$ and $p_2 : L_2 \overset{l_2}{\leftarrow} K_2 \overset{r_2}{\rightarrow} R_2$. Note, that CP_1 (resp. CP_2) is the set of critical pairs, expressing the delete-use-conflicts in which p_2 (resp. p_1) uses something, what

is deleted by p_1 (resp. p_2), `comp.` is an abbreviation of `compatibility` and T_1 (resp. T_2) is a direct transformation induced by a match m_1 (resp. m_2), satisfying the gluing condition and the rule p_1 (resp. p_2).

```
CP_1 = empty;
CP_2 = empty;
compute context C1 of l1;
compute context C2 of l2;
if (C1 != empty and C2 = empty)
  compute all S of L2 and C1, satisfying conflict conditions;
  for each S
    compute all m1:L1->K and m2:L2->K, satisfying comp. conditions;
    if (m1 and m2 fullfill gluing condition)
      compute (T1,T2) induced by (p1,m1) and (p2,m2);
      CP_1 = CP_1 U {(T1,T2)};
if (C2 != empty and C1 = empty)
  compute all S of L1 and C2, satisfying conflict conditions;
  for each S
    compute all m1:L1->K and m2:L2->K, satisfying comp. conditions;
    if (m1 and m2 fullfill gluing condition)
      compute (T1,T2) induced by (p1,m1) and (p2,m2);
      CP_2 = CP_2 U {(T1,T2)};
return (CP_1, CP_2);
```

Note, that this algorithm avoids computing overlappings of a pair of non-deleting rules. The first optimization tells us, that this is the case if the context graphs $C1$ and $C2$ are both empty. Secondly, not all possible overlappings m_1 and m_2 are computed, but only those, satisfying the compatibility conditions over a graphs S, satisfying the conflict conditions. The second optimization tells us, that if m_1 and m_2 fullfill the gluing condition, the pair of induced direct transformations T_1 and T_2 is a critical pair. It is not necessary anymore to check if T_1 and T_2 are in conflict, because the 1st part of Theorem 4.4 tells us, that they automatically are. Moreover following this algorithm we definitely get *all* critical pairs, because this is proven in the 2nd part of Theorem 4.4.
The computation of critical pairs of a pair of rules, as proposed in the beginning of this paragraph before making the optimizations is exponential in the number of nodes and edges of L_1 and L_2 because of the computation of all overlappings (m_1, m_2). For example in Figure 1 we have two lhs's of two rules and all their possible overlappings. For the case of pairs of rules, consisting of at least one non-deleting rule we can apply the recommended two optimizations. For example in Figure 2 we have one deleting and one non-deleting rule and can apply the second optimization, leading us directly to the only

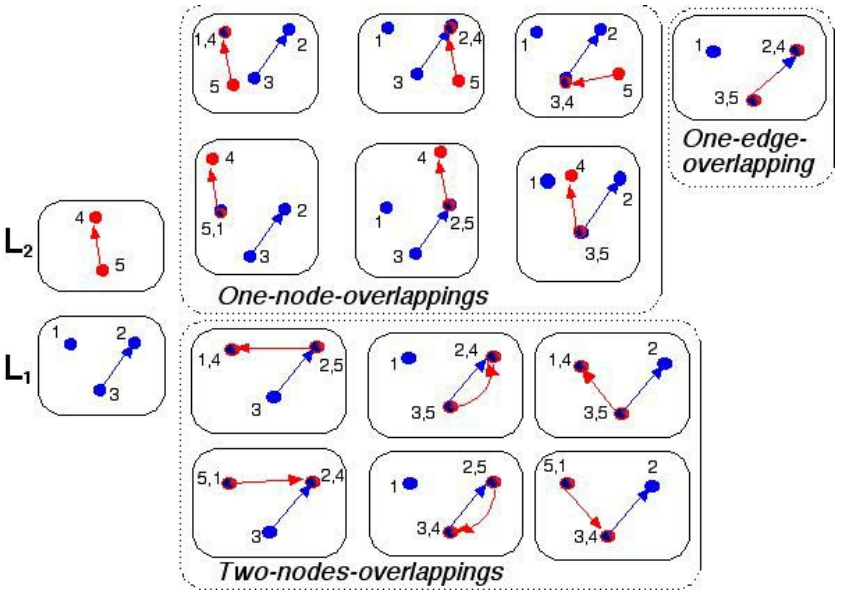


Fig. 1. Example: all possible overlappings of L_1 and L_2

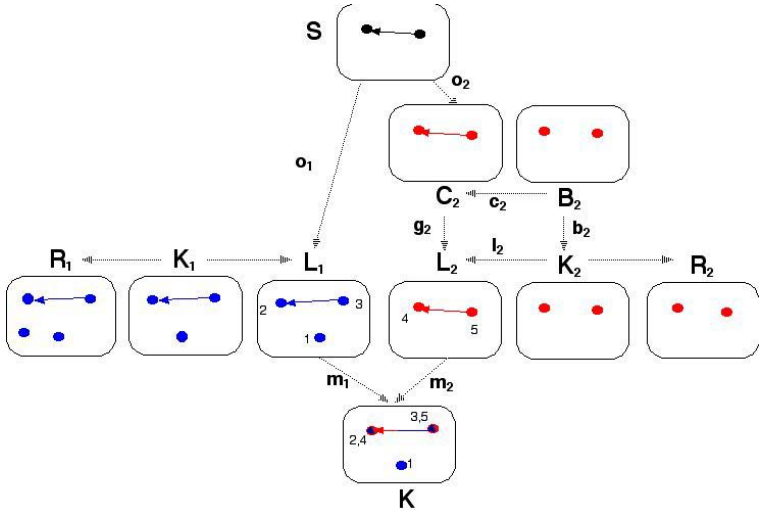


Fig. 2. Example: construction of critical overlapping of L_1 and L_2

critical pair for this pair of rules. This is because only one graph S satisfies the conflict conditions of these rules and only one pair of overlappings satisfies the compatibility conditions with respect to this graph S. In the case of two non-deleting rules the algorithm is of quadratic complexity in the number of nodes and edges of L_1 and L_2, because in this case only the computation of the context graphs is carried out.

6 Summary and Outlook

Conflict detection is a necessary step in checking functional behavior of a graph-based model transformation. A critical pair describes in a formal way a conflict situation occuring in a minimal context. The set of critical pairs, i.e. all minimal conflicts, can be computed in an efficient and constructive way. At first, each pair of rules is analyzed, filtering out only the conflict inducing pair of rules. Secondly, the minimal context of each conflict, described by the critical pair definition, is computed in a constructive way. Therefore needless computations, checking for a conflict in each possible minimal context, are avoided. Further necessary steps in checking functional behavior are analysis of all minimal conflicts for strict confluence and checking for termination of the graph transformation system.

Note, that in this paper an efficient conflict detection is proposed for a pair of non-deleting rules or a pair of one deleting and one non-deleting rule. The case of two deleting rules and the case of having transformations with non-injective matches still have to be investigated. Moreover we should point out the possibility of further improvement of the proposed algorithm. Therefore we would need an extension of the theory which ensures the fact that each critical pair possesses a unique graph S, satisfying the conflict conditions, over which the critical pair can be constructed. Then we could rule out the possibility of computing the same critical pair more than once. This kind of extensions and the theoretical results already available in this paper can be formulated as well in the context of adhesive HLR categories [6], which will be done in a following paper.

Further future work consists of extending conflict detection to other kinds of conflicts, adding negative application conditions (NAC's) [1], typing and attributes [7] to the graph transformation formalism. Therefore, at first the results in [6] about local confluence and critical pairs should be extended to gts with NAC's. In addition we are working on finding a necessary and sufficient condition for a graph transformation system to be locally confluent. In [6] only a sufficient condition is formulated. This means, that it is not known yet if a graph transformation system can be locally confluent if not all of its critical pairs are strictly confluent. Then for these extended graph transformation formalism again the mechanism of at first analysing all pairs of rules and afterwards efficient computation of the minimal context of each conflict should be worked out.

Acknowledgement

This work in collaboration partly arose during a research stay of the first author with a SEGRAVIS (HPRN-CT-2002-00275) grant in february and march '05 at the UPC in Barcelona. The work of Fernando Orejas has been partially supported by the Spanish project GRAMMARS (TIN2004-07925-C03-01). Further on we are grateful to Enrico Biermann, Olga Runge and Gabi Taentzer for the suggestive discussions on the implementation of critical pair detection in AGG [10].

References

[1] Annegret Habel, R. H. and G. Taentzer, *Graph grammars with negative application conditions* (1996).

[2] Corradini, A., U. Montanari, F. Rossi, H. Ehrig, R. Heckel and M. Löwe, *Algebraic Approaches to Graph Transformation I: Basic Concepts and Double Pushout Approach*, in: G. Rozenberg, editor, *Handbook of Graph Grammars and Computing by Graph Transformation, Volume 1: Foundations*, World Scientific, 1997 pp. 163–245.

[3] de Lara, J. and G. Taentzer, *Automated Model Transformation and its Validation using ATOM^3 and AGG*, in: A. Blackwell, K. Marriott and A. Shimojima, editors, *Diagrammatic Representation and Inference* (2004).

[4] Ehrig, H., K. Ehrig, J. de Lara, G. Taentzer, D. Varró and S. Varró-Gyapay, *Termination criteria for model transformation*, in: *Proc. Fundamental Approaches to Software Engineering (FASE)*, Lecture Notes in Computer Science (2005), pp. 00–00, to appear.
URL http://tfs.cs.tu-berlin.de/~ehrig/public/EEL+05.pdf

[5] Ehrig, H., K. Ehrig, A. Habel and K.-H. Pennemann, *Constraints and application conditions: From graphs to high-level structures*, in: F. Parisi-Presicce, P. Bottoni and G. Engels, editors, *Proc. 2nd Int. Conference on Graph Transformation (ICGT'04)*, LNCS 3256 (2004), pp. 287–303.
URL http://www.cs.tu-berlin.de/~ehrig/publications/ICGT04paper3.pdf

[6] Ehrig, H., A. Habel, J. Padberg and U. Prange, *Adhesive high-level replacement categories and systems*, in: F. Parisi-Presicce, P. Bottoni and G. Engels, editors, *Proc. 2nd Int. Conference on Graph Transformation (ICGT'04)*, LNCS 3256 (2004), pp. 144–160.
URL http://www.cs.tu-berlin.de/~ehrig/publications/ICGT04paper1.pdf

[7] Ehrig, H., U. Prange and G. Taentzer, *Fundamental theory for typed attributed graph transformation*, in: F. Parisi-Presicce, P. Bottoni and G. Engels, editors, *Proc. 2nd Int. Conference on Graph Transformation (ICGT'04), Rome, Italy*, LNCS 3256, Springer, 2004 pp. 161–177.
URL http://www.cs.tu-berlin.de/~ehrig/publications/ICGT04paper2.pdf

[8] Huet, G., *Confluent reductions: Abstract properties and applications to term rewriting systems* (1980).

[9] Plump, D., *Hypergraph Rewriting: Critical Pairs and Undecidability of Confluence*, in: M. Sleep, M. Plasmeijer and M. C. van Eekelen, editors, *Term Graph Rewriting*, Wiley, 1993 pp. 201–214.

[10] Taentzer, G., *AGG: A Graph Transformation Environment for Modeling and Validation of Software*, in: J. Pfaltz, M. Nagl and B. Boehlen, editors, *Application of Graph Transformations with Industrial Relevance (AGTIVE'03)*, LNCS 3062, Springer, 2004 pp. 446 – 456.