Embedding and Confluence of Graph Transformations with Negative Application Conditions

Leen Lambers[1], Hartmut Ehrig[1], Ulrike Prange[1], and Fernando Orejas[2]

[1] Institute for Software Engineering and Theoretical Informatics
Technical University Berlin, Germany
{leen,ehrig,uprange}@cs.tu-berlin.de
[2] Department L.S.I, Technical University of Catalonia, Spain
orejas@lsi.upc.edu

Abstract. The goal of this paper is the generalization of embedding and confluence results for graph transformation systems to transformation systems with negative application conditions (NACs). These conditions restrict the application of a rule by expressing that a specific structure must not be present before or after applying the rule to a certain context. Such a condition influences each rule application and transformation and therefore changes significantly the properties of the transformation system. This behavior modification is reflected by the generalization of the Embedding Theorem and the Critical Pair Lemma or Local Confluence Theorem, formulated already for graph transformation systems without negative application conditions. The results hold for adhesive high-level replacement systems with NACs and are formulated in this paper for the instantiation to double-pushout graph transformation systems with NACs. All constructions and results are explained on a running example.

1 Introduction

In graph transformation, negative application conditions (NACs) express that certain structures at a given time are forbidden. They are a widely used feature for several applications of graph transformation e.g., [1,2]. In order to allow confluence analysis for these applications, the theory already worked out for graph transformation systems (gts) without NACs has to be generalized to gts with NACs. The notion of critical pairs is central in this theory. It was first developed in the area of term rewriting systems (e.g., [3]) and, later, introduced in the area of graph transformation for hyper-graph rewriting [4,5] and then for all kinds of transformation systems fitting into the framework of adhesive high-level replacement (HLR) categories [6]. We tailored the theory presented in this paper for gts with NACs and not for other kind of constraints or application conditions, since NACs are already widely used in practice.

For gts without NACs, embedding of a graph transformation sequence without NACs and local confluence of a gts without NACs has been investigated in detail in [6]. Recall that in order to be able to embed a transformation without

H. Ehrig et al. (Eds.): ICGT 2008, LNCS 5214, pp. 162–177, 2008.

NACs into some larger context by some extension morphism k, this morphism should be consistent as defined in [6]. Using the results on concurrency for graph transformation with NACs [7] we introduce in this paper the definition of NAC-consistency of an extension morphism. This is an additional condition on top of standard consistency enabling the generalization of the Embedding Theorem to transformations with NACs. Recall moreover that, for a gts without NACs, in order to be locally confluent it suffices that all critical pairs are strictly confluent. Having generalized the notion of critical pairs [8], completeness [8], and embedding to transformations with NACs in this paper, we moreover introduce a sufficient condition on the critical pairs with NACs. This condition implies local confluence of a gts with NACs as stated in the introduced Critical Pair Lemma with NACs. The proofs of these results are given in a technical report [9] on the level of adhesive HLR systems. In return, all results are illustrated in this paper by an example modeling order and payment transactions in a restaurant by typed graphs and rules with NACs.

The structure of this paper is as follows. In Section 2, we introduce preliminaries on gts with NACs and main results on concurrency for graph transformation with NACs. In Section 3, it is explained under which conditions it is possible to embed transformations with NACs. In Section 4, results on confluence of transformation systems with NACs are formulated. Section 5 concludes this paper with remarks on future work and a short summary.

2 Graph Transformation Systems with NACs

In this section, we reintroduce gts with NACs and some preliminary results that we need for the remaining paper. NACs are an important feature for the modeling of transformation systems, expressing that a certain structure is not present when performing the transformation [10] and thus enhancing the expressiveness of the transformation. In order to provide a rich theory for such transformations with NACs, they are integrated into the framework of adhesive HLR systems [6]. In [7] it is remarked that gts with NACs are a valid instantiation of adhesive HLR systems with NACs. In this paper, we concentrate on formulating the results for graph transformation with NACs and showing their significance on an example.

Definition 1 (typed 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* type graph *is a distinguished graph* TG*. A* typed graph $G^T : (G, type)$ *over* TG *is a graph* G *and a graph morphism* $type : G \to TG$*. A* typed graph morphism $f : G_1^T \to G_2^T$ *is a graph morphism* $f : G_1 \to G_2$ *with* $type_2 \circ f = type_1$*.*

From now on we only consider typed graphs and morphisms over a given type graph TG and omit the prefix typed and the index T in our notations.

Definition 2 (injective, surjective, overlapping, pair factorization). *A graph morphism $f : G_1 \to G_2$ is injective (resp. surjective) if f_V and f_E are injective (resp. surjective) mappings. 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* pair factorization *of two graph morphisms $(m_1 : G_1 \to H, m_2 : G_2 \to H)$ consists of a pair of jointly surjective morphisms $(e_1 : G_1 \to E, e_2 : G_2 \to E)$ and an injective morphism $m : E \to H$ such that $m \circ e_1 = m_1$ and $m \circ e_2 = m_2$ and is unique up to isomorphism.*

Definition 3 (rule and match). *A graph transformation* rule $p : L \overset{l}{\leftarrow} K \overset{r}{\rightarrow} R$ *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. Given a rule $p : L \overset{l}{\leftarrow} K \overset{r}{\rightarrow} R$ and a graph G, one can try to apply p to G if there is an occurrence of L in G i.e. a graph morphism, called* match $m : L \to G$.

A negative application condition or NAC as introduced in [10] forbids a certain graph structure to be present before or after applying a rule.

Definition 4 (negative application condition)

- *A* negative application condition *or $NAC(n)$* on *L is a graph morphism $n : L \to N$. A graph morphism $g : L \to G$* satisfies *$NAC(n)$ on L i.e. $g \models NAC(n)$ if and only if $\nexists\ q : N \to G$ which is injective such that $q \circ n = g$.*

$$\begin{tikzcd} L \arrow[r, "n"] \arrow[d, "g"] & N \arrow[dl, dotted, bend left, "X" description, "q"] \\ G & \end{tikzcd}$$

- *A $NAC(n)$ on L (resp. R) for a rule $p : L \overset{l}{\leftarrow} K \overset{r}{\rightarrow} R$ is called* left *(resp.* right*) NAC on p. $NAC_{p,L}$ (resp. $NAC_{p,R}$) is a set of left (resp. right) NACs on p. $NAC_p = (NAC_{p,L}, NAC_{p,R})$, consisting of a set of left and a set of right NACs on p is called a set of NACs on p.*

Definition 5 (graph transformation with NACs)

- *A* graph transformation system with NACs *is a set of rules where each rule $p : L \overset{l}{\leftarrow} K \overset{r}{\rightarrow} R$ has a set $NAC_p = (NAC_{p,L}, NAC_{p,R})$ of NACs on p.*
- *A* direct graph transformation *$G \overset{p,g}{\Rightarrow} H$ via a rule $p : L \overset{l}{\leftarrow} K \overset{r}{\rightarrow} R$ with $NAC_p = (NAC_{p,L}, NAC_{p,R})$ and a match $g : L \to G$ consists of the double pushout [11] (DPO) at the right where g satisfies each NAC in $NAC_{p,L}$, written $g \models NAC_{p,L}$, and h satisfies each NAC in $NAC_{p,R}$, written $h \models NAC_{p,R}$. 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 g satisfies the gluing condition of rule p. A* graph transformation*, denoted as $G_0 \overset{*}{\Rightarrow} G_n$, is a sequence $G_0 \Rightarrow G_1 \Rightarrow \cdots \Rightarrow G_n$ of direct graph transformations.*

$$\begin{tikzcd} L \arrow[d, "g"'] & K \arrow[l, "l"'] \arrow[r, "r"] \arrow[d] & R \arrow[d, "h"] \\ G & D \arrow[l] \arrow[r] & H \end{tikzcd}$$

Remark 1. From now on we only consider gts with rules having an empty set of right NACs. This is without loss of generality, because each right NAC can be translated into an equivalent left NAC as explained in [6], where Theorem 7.17 can be specialized to NACs as remarked in [7].

Example 1. Here we introduce our running example *Restaurant* modeling order and payment transactions in a restaurant. The type and start graph of *Restaurant* are depicted in Fig. 1. Note that all results reviewed and introduced in this paper hold in particular for typed gts with NACs, since they are a valid instantiation of adhesive HLR systems with NACs. The rule *openAccountTable* shown in Fig. 1 models the opening of an account for one of the tables in the restaurant. This rule holds two NACs: *notOpened* forbids the rule to be applied twice to the same table and *noAlert* specifies that an account can not be opened if there exists an alert for the table. An alert can be generated by rule *alertUnpaidTable* if a staff member of the restaurant notices that guests have left a table without paying. In this case an exception handling starts making sure that unpaid orders are considered when doing the daily accounting. The NAC *noAlert* for this rule avoids it to be applied if an alert for the table already exists. *noteUnpaidOrder* notes an unpaid order for a table by deleting it. *resetAlert* can reset the alert for a table if it e.g. appeared to be a false alarm or all unpaid orders are processed by *noteUnpaidOrder* in the meanwhile. Furthermore, *gatherOrder* can assign an order for a table if there is no alert, *payOrder* models the paying of an order for the case that there is no alert expressed by NAC *noAlert* and finally *closeAccountTable* can close the account of a table if all orders of a table are processed which is expressed by NAC *noOrders*. Note that the rules *payOrder* and *noteUnpaidOrder* have the same effect on the system since they both just delete an order, but the first one can only be applied if there is no alert in contrast to the second one. Of course it is possible to augment this system with information on the price of the order, keeping track of a list of paid resp. unpaid orders etc. For the purpose of this paper though we restrict ourselves to these more simple operations.

In the following sections, we repeatedly need two constructions translating NACs via morphisms and rules. More precisely, the mapping D translates NACs downwards along morphisms. Given a diagram as depicted in Fig. 2, consider a $NAC_{L'_c}$ on L'_c and a morphism m_c, then $D_{m_c}(NAC_{L'_c})$ translates $NAC_{L'_c}$ into equivalent NACs on L_c. The basic idea of the construction of D is to consider all suitable overlappings of L_c and $NAC_{L'_c}$. The mapping DL translates NACs down- and leftwards, i.e. given the diagram in Fig. 2 with NAC_{L_n} on L_n, a morphism e_n, and a rule $L_c \leftarrow C_c \rightarrow E$, then $DL_{p_c}(NAC_{L_n})$ translates the NACs on L_n to equivalent NACs on L_c. The construction of DL is based on D translating NAC_{L_n} to equivalent NACs on E, and then on the well-known construction of right to left application conditions [6]. For more details see [7,9].

Now we introduce the concurrent rule with NACs p_c induced by a transformation with NACs $t : G_0 \overset{n+1}{\Longrightarrow} G_{n+1}$ via a sequence of rules $p_0, \ldots, p_n$. Intuitively, a concurrent rule with NACs summarizes in one rule which parts of a graph G_0

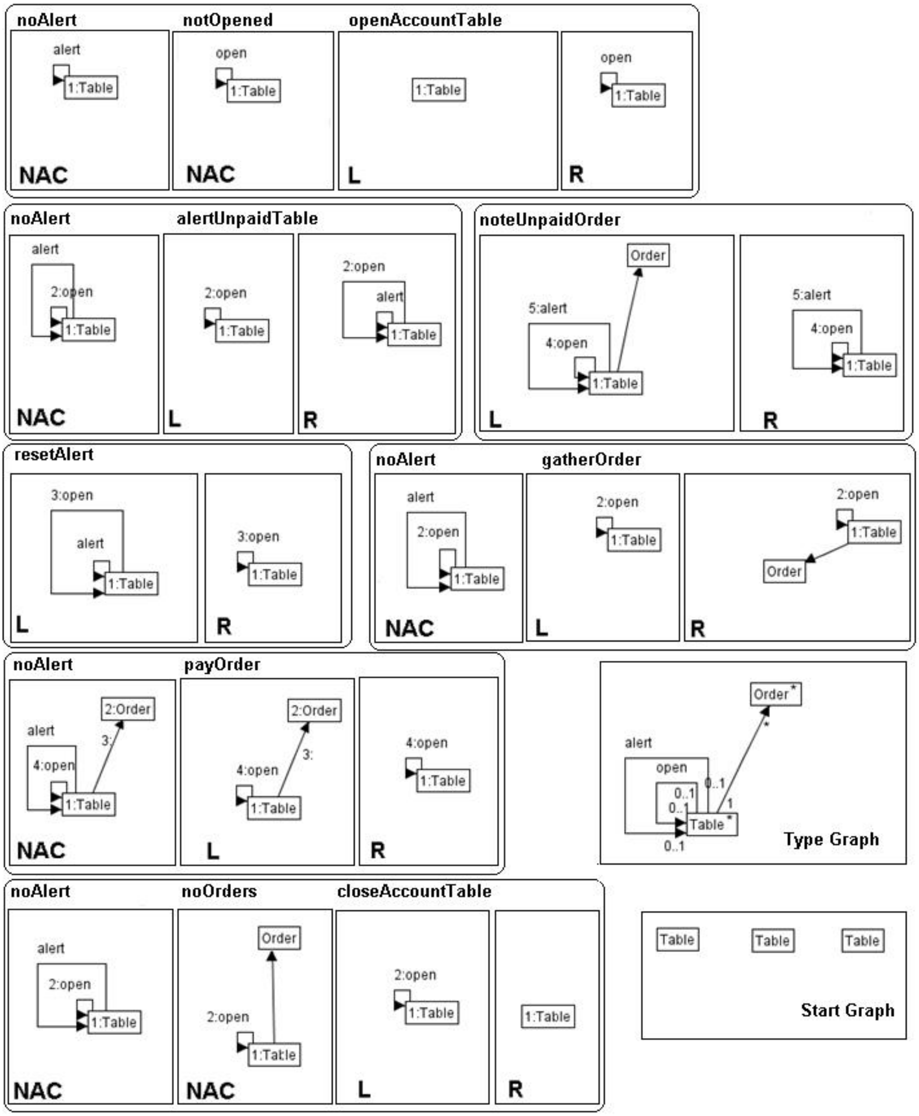


Fig. 1. Start graph, type graph and rules of *Restaurant*

should be present, preserved, deleted, and produced by t. Moreover we have a summarized set of NACs on the concurrent rule p_c expressing which graph parts are forbidden when applying $p_0, \ldots, p_n$ to G_0 leading to t. Note that in [7,9] it is proven that it is possible to repeat the transformation t in one step via the concurrent rule p_c with NACs. In addition, whenever it is possible to apply a concurrent rule with NACs p_c it is also possible to resequentialize this one-step

$$\begin{tikzcd}
L_0 \arrow[r, "m'_c"] & L'_c \arrow[d, dotted, "m_c"] \arrow[ddd, bend right, "g'_c"'] & K'_c \arrow[l] \arrow[r] \arrow[d, dotted] & R'_c \arrow[dr, dotted, "e'_c"] \arrow[dddr, bend right=15, "h'_c"] & & L_n \arrow[dl, dotted, "e_n"'] \arrow[dddl, bend left=15, "g_n"] & K_n \arrow[l] \arrow[r] \arrow[d, dotted] & R_n \arrow[d, dotted] \arrow[ddd, bend left, "g_{n+1}"] \\
& L_c \arrow[dd, dotted, "g_c"] & C_c \arrow[l, dotted, "l"'] \arrow[rr, dotted] \arrow[dd, dotted] \arrow[dr, phantom, "(1)"] & & E \arrow[dd, dotted, "h"] & & C_n \arrow[ll, dotted] \arrow[r, dotted, "r"] \arrow[dd, dotted] & R_c \arrow[dd, dotted, "h_c"'] \\
& & & K_c \arrow[ul, dotted, "k_c"] \arrow[urrr, dotted, "k_n"'] & & & & \\
& G_0 & D_n \arrow[l] \arrow[rr] & & G_n & & D \arrow[ll] \arrow[r] & G_{n+1}
\end{tikzcd}$$

Fig. 2. Definition of concurrent rule with NACs

transformation into single steps via the sequence of original rules with NACs which led to this concurrent rule.

Definition 6 (concurrent rule with NAC, concurrent (co-, lhs-)match induced by $G_0 \overset{n+1}{\Longrightarrow} G_{n+1}$)

$n = 0$ *For a direct transformation $G_0 \Rightarrow G_1$ via match $g_0 : L_0 \to G_0$, comatch $g_1 : R_0 \to G_1$ and rule $p_0 : L_0 \leftarrow K_0 \to R_0$ with NAC_{p_0} the* concurrent rule p_c with NAC *induced by $G_0 \Rightarrow G_1$ is defined by $p_c = p_0$ with $NAC_{p_c} = NAC_{p_0}$, the* concurrent comatch h_c *is defined by $h_c = g_1$, the* concurrent lhs-match *by $id : L_0 \to L_0$ and the* concurrent match g_c *by $g_c = g_0 : L_0 \to G_0$.*

$n \geq 1$ *Consider $p'_c : L'_c \leftarrow K'_c \to R'_c$ (resp. $g'_c : L'_c \to G_0$, $h'_c : R'_c \to G_n$,$m'_c : L_0 \to L'_c$), the concurrent rule with NACs (resp. concurrent match, comatch, lhs-match) induced by $G_0 \overset{n}{\Longrightarrow} G_n$. Let $((e'_c, e_n), h)$ be the pair factorization of the comatch h'_c and match g_n of $G_n \Rightarrow G_{n+1}$. According to Fact 5.29 in [6] PO-PB decomposition, PO composition and decomposition lead to the diagram in Fig. 2 in which (1) is a pullback and all other squares are pushouts. For a transformation sequence $G_0 \overset{n+1}{\Longrightarrow} G_{n+1}$ the* concurrent rule p_c with NACs *(resp. concurrent match, comatch, lhs-match) induced by $G_0 \overset{n+1}{\Longrightarrow} G_{n+1}$ is defined by $p_c = L_c \overset{l \circ k_c}{\leftarrow} K_c \overset{r \circ k_n}{\to} R_c$ ($g_c : L_c \to G_0$, $h_c : R_c \to G_{n+1}$, $m_c \circ m'_c : L_0 \to L_c$). Thereby NAC_{p_c} is defined by $NAC_{p_c} = DL_{p_c}(NAC_{L_n}) \cup D_{m_c}(NAC_{L'_c})$.*

Example 2. Consider the graph *Middle2Orders* depicted in Fig. 3 and a transformation t : *Middle2Orders* $\overset{alertUnpaidTable}{\Rightarrow} G_1 \overset{noteUnpaidOrder}{\Rightarrow} G_2 \overset{noteUnpaidOrder}{\Rightarrow} G_3 \overset{resetAlert}{\Rightarrow} G_4 \overset{closeAccountTable}{\Rightarrow} G_5 \overset{closeAccountTable}{\Rightarrow}$ *Startgraph* in which an alert is generated for the middle table, consequently both orders on the middle table are noted as unpaid, the alert is then reset, the middle table account is closed and the right table account is closed as well. The lhs L_c and rhs R_c of the concurrent rule p_c induced by transformation t is depicted in Fig. 3 together with the concurrent transformation via p_c summarizing t into one step. It deletes two

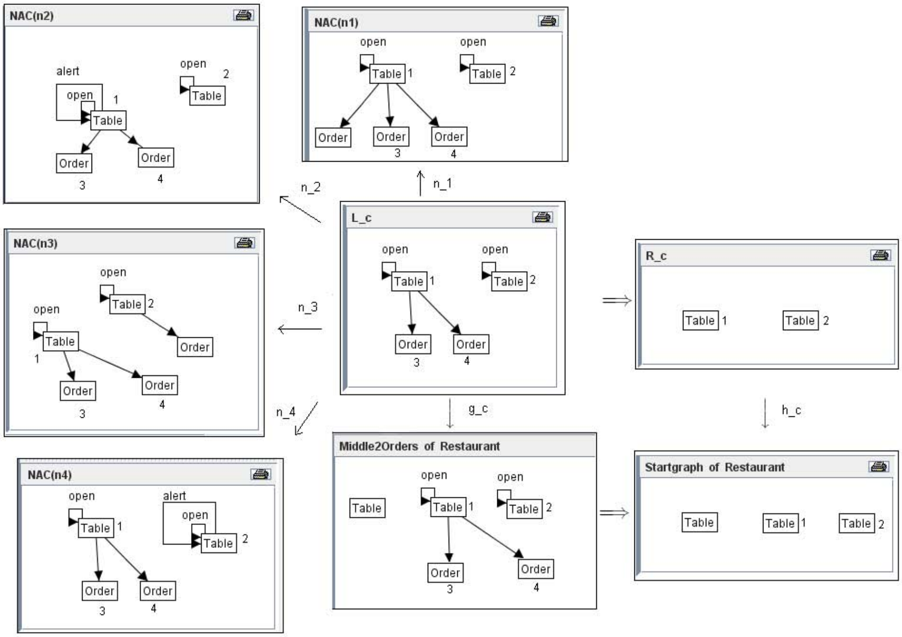


Fig. 3. Concurrent rule and transformation with NACs

different orders belonging to the middle table, closes its account and in addition closes the account of the right table. Note that the node ids in this figure define the morphisms. The concurrent NAC_{p_c} induced by t holds a $NAC(n_1)$ forbidding more than two orders, a $NAC(n_2)$ forbidding an alert for the table holding already two orders, a $NAC(n_3)$ forbidding any order, and $NAC(n_4)$ forbidding an alert for the other table. Note that the same $NAC(n_2)$ is induced by the NACs of *alertUnpaidTable* and *closeAccountTable* applied to the middle table.

Finally, for the Embedding Theorem with NACs we reintroduce the notion of extension diagram with NACs [8].

Definition 7 (extension diagram with NACs)

An extension diagram *is a diagram (1), where,* $k_0 : G_0 \rightarrow G'_0$ *is a graph morphism, called extension morphism, and* $t : G_0 \overset{*}{\Rightarrow} G_n$ *and* $t' : G'_0 \overset{*}{\Rightarrow} G'_n$ *are graph transformations via the same rules* $(p_0, \cdots, p_{n-1})$ *with NACs, and matches* $(m_0, \cdots, m_{n-1})$ *and extended matches* $(k_0 \circ m_0, \cdots, k_{n-1} \circ m_{n-1})$*, respectively, defined by the DPO diagrams on the right for each* p_i*. Since* t *and* t' *are transformations with NACs, the matches* $(m_0, \cdots, m_{n-1})$ *and extended matches* $(k_0 \circ m_0, \cdots, k_{n-1} \circ m_{n-1})$ *have to satisfy the NACs of the rules* $(p_0, \cdots, p_{n-1})$*.*

$$\begin{tikzcd}
G_0 \arrow[r, Rightarrow, "t"] \arrow[d, "k_0"'] \arrow[dr, phantom, "(1)"] & {}^{*}G_n \arrow[d, "k_n"] \\
G'_0 \arrow[r, Rightarrow, "t'"] & {}^{*}G'_n
\end{tikzcd}$$

$$\begin{tikzcd}
L_i \arrow[d, "m_i"'] & K_i \arrow[l, "l_i"] \arrow[r, "r_i"'] \arrow[d, "j_i"'] & R_i \arrow[d, "n_i"'] \\
G_i \arrow[d, "k_i"'] & D_i \arrow[l, "f_i"] \arrow[r, "g_i"'] \arrow[d, "d_i"'] & G_{i+1} \arrow[d, "k_{i+1}"'] \\
G'_i & D'_i \arrow[l, "f'_i"] \arrow[r, "g'_i"'] & G'_{i+1}
\end{tikzcd}$$

3 Embedding of Transformations with NACs

Recall [6] that in order to be able to embed a transformation without NACs into some other context by an extension morphism k, this morphism should not identify graph elements which are deleted and on the other hand preserved by the transformation. Moreover k should not map any node which is deleted by the transformation to a node which is the source or target of an additional edge in the new context. This condition on the extension morphism k can be checked by computing its boundary and context, and checking then consistency as defined in [6]. Combined with the results on concurrency for graph transformation with NACs [7] it is possible to define also NAC-consistency of an extension morphism. This is an additional condition needed on top of standard consistency to generalize the embedding of transformations to transformations with NACs. Note that in order to make the difference between consistency and NAC-consistency of an extension morphism clear, we call consistency from now on *boundary consistency.*

Now we can formulate the definition of NAC-consistency for an extension morphism k_0 w.r.t. a transformation t. It expresses that the extended concurrent match induced by t should fulfill the concurrent NAC induced by t.

Definition 8 (NAC-consistency). *A morphism $k_0 : G_0 \to G'_0$ is called NAC-consistent w.r.t. a transformation $t : G_0 \overset{*}{\Rightarrow} G_n$ if $k_0 \circ g_c \models NAC_{p_c}$ with NAC_{p_c} the* concurrent NAC *and g_c the* concurrent match induced by *t.*

The Embedding Theorem for rules with NACs needs NAC-consistency of the extension morphism k_0 on top of boundary consistency. Note that in [9] also the Extension Theorem with NACs is proven describing the fact that boundary and NAC-consistency are not only sufficient, but also necessary conditions for the construction of extension diagrams for transformations with NACs.

Theorem 1 (Embedding Theorem with NACs [9])

Given a transformation $t : G_0 \overset{n}{\Longrightarrow} G_n$ with NACs. If $k_0 : G_0 \to G'_0$ is boundary consistent and NAC-consistent w.r.t. t then there exists an extension diagram with NACs over t and k_0 as defined in Def. 7 and depicted on the right.

$$\begin{tikzcd}
G_0 \arrow[r, Rightarrow, "t"] \arrow[d, "k_0"'] \arrow[dr, phantom, "(1)"] & {}^{*}G_n \arrow[d, "k_n"] \\
G'_0 \arrow[r, Rightarrow, "t'"] & {}^{*}G'_n
\end{tikzcd}$$

Example 3. Consider the transformation $t : Middle2Orders \overset{*}{\Rightarrow} Startgraph$ with its concurrent rule p_c and NAC_{p_c} as described in Example 2 and depicted as concurrent transformation in Fig. 3. In addition, consider an inclusion morphism $k : Middle2Orders \to Middle3Orders$ in which the graph $Middle3Orders$ has an additional order for the middle table. The morphism k is not NAC-consistent, since the concurrent NAC induced by t is not satisfied by $k \circ g_c$. This is because $NAC(n_1)$ forbidding more than 2 orders on the middle table is not satisfied. Thus it is not possible to embed transformation t into *Middle3Orders* by the extension morphism k, since k is not NAC-consistent. Intuitively speaking, it is not possible to change the state of the tables in the restaurant in the same way as transformation t if the middle table holds an extra order. This is because rule *closeAccountTable* would forbid closing the account for the middle table still holding one order. Consider a different inclusion morphism $k' : \mathit{Middle2Orders} \rightarrow \mathit{4Tables}$ in which *4Tables* just holds an extra table. Now k' is NAC-consistent and it is possible to embed t into *4Tables*. Intuitively speaking, the embedded transformation t' changes the states of the tables in the restaurant in the same way as transformation t, but does this in a restaurant with an extra table.

4 Confluence of Transformations with NACs

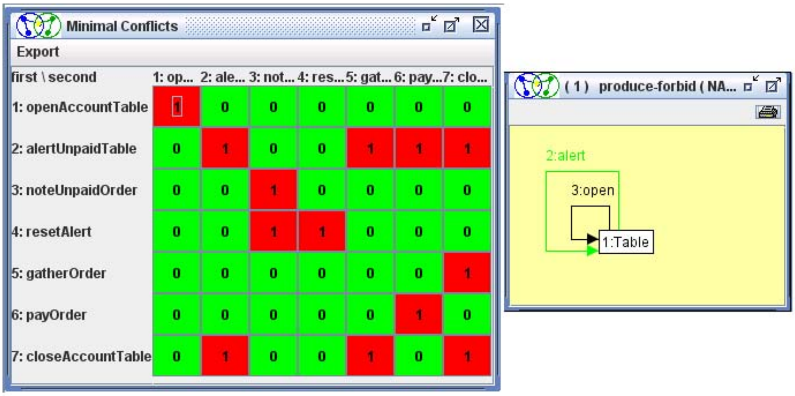
Local confluence of a transformation system without NACs can be inferred from the strict confluence of all critical pairs (see Critical Pair Lemma [6]). If a critical pair is strictly confluent via some transformations t_1 and t_2, we say that (t_1, t_2) is a *strict solution* of the critical pair. Intuitively speaking, strict confluence means that the common structure which is preserved by the critical pair should be preserved by the strict solution of this critical pair as well. For the Critical Pair Lemma with NACs we need a stronger condition though to obtain local confluence of the transformation system. In addition to strict confluence of all critical pairs we need also NAC-confluence. NAC-confluence of a critical pair ensures that for each context into which the critical pair can be embedded, such a strict solution can be embedded into this context as well without violating the NACs present in t_1 and t_2. In particular, a critical pair is NAC-confluent if the NAC-consistency (as defined in Def. 8) of each extension morphism w.r.t. a strict solution of the critical pair follows from the NAC-consistency of the extension morphism w.r.t. the critical pair itself.

First we state the definition of a critical pair with NACs. A critical pair describes a conflict between two rules in a minimal context. Therefore we consider in the following only overlaps of graphs in order to rule out superfluous context. Moreover, it is proven in [8] that the following critical pair definition satisfies completeness. This means intuitively that for each pair of conflicting transformations there exists a critical pair expressing the same conflict in a minimal context.

Definition 9 (critical pair). *A critical pair is a pair of direct transformations* $K \overset{(p_1,m_1)}{\Rightarrow} P_1$ *with* NAC_{p_1} *and* $K \overset{(p_2,m_2)}{\Rightarrow} P_2$ *with* NAC_{p_2} *such that:*

1. (a) $\nexists h_{12} : L_1 \rightarrow D_2 : d_2 \circ h_{12} = m_1$ *and* (m_1, m_2) *jointly surjective (use-delete-conflict)*
   or
   (b) *there exists* $h_{12} : L_1 \rightarrow D_2$ *s.t.* $d_2 \circ h_{12} = m_1$*, but for one of the NACs* $n_1 : L_1 \rightarrow N_1$ *of* p_1 *there exists an injective morphism* $q_{12} : N_1 \rightarrow P_2$ *s.t.* $q_{12} \circ n_1 = e_2 \circ h_{12}$ *and* (q_{12}, h_2) *jointly surjective (forbid-produce-conflict)*

   or
2. (a) $\nexists h_{21} : L_2 \rightarrow D_1 : d_1 \circ h_{21} = m_2$ *and* (m_1, m_2) *jointly surjective (delete-use-conflict)*
   or



Minimal Conflicts

Export

first \ second	1: op...	2: ale...	3: not...	4: res...	5: gat...	6: pay...	7: clo...
1: openAccountTable	1	0	0	0	0	0	0
2: alertUnpaidTable	0	1	0	0	1	1	1
3: noteUnpaidOrder	0	0	1	0	0	0	0
4: resetAlert	0	0	1	1	0	0	0
5: gatherOrder	0	0	0	0	0	0	1
6: payOrder	0	0	0	0	0	1	0
7: closeAccountTable	0	1	0	0	1	0	1

(1) produce-forbid (NA...

Fig. 4. Conflict Matrix for *Restaurant* and minimal context for *(alertUnpaidTable,gatherOrder)*

(b) there exists $h_{21} : L_2 \to D_1$ *s.t.* $d_1 \circ h_{21} = m_2$*, but for one of the NACs* $n_2 : L_2 \to N_2$ *of* p_2 *there exists an injective morphism* $q_{21} : N_2 \to P_1$ *s.t.* $q_{21} \circ n_2 = e_1 \circ h_{21}$ *and* (q_{21}, h_1) *jointly surjective (produce-forbid-conflict).*

$$\begin{tikzcd}
 & & N_1 \arrow[rr, dotted, dash] \arrow[ddrrrr, dotted, bend left=10, "q_{12}"] & & N_2 \arrow[ddllll, dotted, bend right=10, "q_{21}"'] & & \\
R_1 \arrow[d, "h_1"'] & K_1 \arrow[l, "r_1"'] \arrow[r, "l_1"] \arrow[d] & L_1 \arrow[u, "n_1"'] \arrow[dr, "m_1"] \arrow[drrr, dotted, "h_{12}"] & & L_2 \arrow[u, "n_2"] \arrow[dl, "m_2"'] \arrow[dlll, dotted, "h_{21}"'] & K_2 \arrow[l, "l_2"'] \arrow[r, "r_2"] \arrow[d] & R_2 \arrow[d, "h_2"] \\
P_1 & D_1 \arrow[l, "e_1"] \arrow[rr, "d_1"'] & & K & & D_2 \arrow[ll, "d_2"] \arrow[r, "e_2"'] & P_2
\end{tikzcd}$$

Example 4. All critical pairs of a gts can be computed by the graph transformation tool AGG [12]. They are illustrated by a Conflict Matrix as for our *Restaurant* example in the left part of Fig. 4. More precisely, entry (p_j, p_i) (row, column) in this matrix denotes the number of critical pairs $K \overset{(p_j,m_j)}{\Rightarrow} P_j$ with NAC_{p_j} and $K \overset{(p_i,m_i)}{\Rightarrow} P_i$ with NAC_{p_i} describing delete-use and produce-forbid-conflicts as defined in Def. 9. Consider in particular entry $(alertUnpaidTable, gatherOrder)$. This critical pair expresses a produce-forbid conflict, since when an alert is set for a certain table it is impossible to gather an order for it afterwards. The minimal context expressing this conflict is in the right part of Fig. 4 depicting graph P_j. The critical pair itself is depicted in the left part of Fig. 5.

Definition 10 (strict NAC-confluence). *A critical pair* $P_1 \overset{p_1,g_1}{\Leftarrow} K \overset{p_2,g_2}{\Rightarrow} P_2$ *is* strictly NAC-confluent *if*

- *it is* strictly confluent *via some transformations* $t_1 : K \overset{p_1,g_1}{\Rightarrow} P_1 \Rightarrow^* X$ *and* $t_2 : K \overset{p_2,g_2}{\Rightarrow} P_2 \Rightarrow^* X$ *(see [6])*
- *and it is* NAC-confluent for t_1 and t_2, *i.e. for every injective morphism* $k_0 : K \to G$ *which is NAC-consistent w.r.t.* $K \overset{p_1,g_1}{\Rightarrow} P_1$ *and* $K \overset{p_2,g_2}{\Rightarrow} P_2$ *it follows that* k_0 *is NAC-consistent w.r.t.* t_1 *and* t_2.

$$\begin{tikzcd}
 & & P_1 \arrow[dr, Rightarrow, "*"] & \\
G & K \arrow[l, "k_0"'] \arrow[ur, Rightarrow, "{(p_1,g_1)}"] \arrow[dr, Rightarrow, "{(p_2,g_2)}"'] & & X \\
 & & P_2 \arrow[ur, Rightarrow, "*"'] &
\end{tikzcd}$$

Remark 2. Injectivity of k_0 is sufficient by completeness of critical pairs [8].

Example 5. Consider again the Conflict Matrix of *Restaurant* depicted in Fig. 4. Let us investigate the following critical pairs for strict NAC-confluence:

- *(resetAlert,noteUnpaidOrder).* It describes a delete-use-conflict, since *resetAlert* deletes the alert and *noteUnpaidOrder* can only be applied if an alert is set for the table. This critical pair can be resolved by a strict solution by on the one hand applying *payOrder* and on the other hand *resetAlert.* Now we investigate if this critical pair is also NAC-confluent for this solution. Rule *resetAlert* does not hold a NAC and therefore there is nothing to prove. On the other hand *payOrder* holds a NAC which forbids an alert on the table. The rules *resetAlert* and *noteUnpaidOrder* can only be applied if an alert on the table is present. After applying *resetAlert* though this alert is in any case deleted. The only problem that can occur is that in another context more than one alert is present for a table. Cardinality constraints on the type graph of *Restaurant* as depicted in Fig. 1 forbid this possibility though. Therefore we can conclude that this critical pair is strictly NAC-confluent.
- *(gatherOrder,closeAccountTable).* It describes a produce-forbid-conflict, since *gatherOrder* produces an order which is forbidden by *closeAccountTable.* This pair can be resolved on the one hand by paying the order and closing the table account and on the other hand nothing. This solution demands no alert on the table because of NAC *noAlert* on *payOrder* and no orders because of NAC *noOrders* on *closeAccountTable.* It becomes clear that this critical pair is NAC-confluent for this solution because it only occurs on a table without an alert and without any order. These are exactly the restrictions for which the above solution holds.
- *(alertUnpaidTable,gatherOrder).* We described this critical pair already in Example 4 and it is depicted in Fig. 5. There exists a strict solution for this critical pair resetting the alert on the one hand and paying the order on the other hand. This solution is depicted also in Fig. 5 (part without rounded rectangle). This critical pair is NAC-confluent for this solution since *alertUnpaidTable* can be applied only on a table without any order and therefore when paying the order there will not be any alert either.

  Consider though a somewhat larger strict solution for this critical pair, namely on the one hand resetting the alert and closing the account and on the other hand paying the order and then closing the account. In Fig. 5 this means we consider now as well the rounded rectangle. The critical pair is

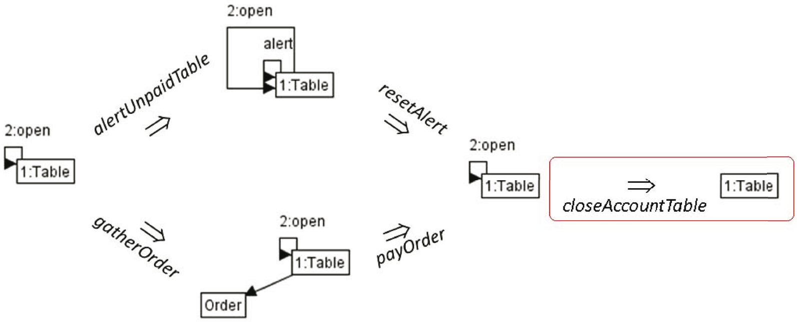


Fig. 5. Critical pair *(alertUnpaidTable,gatherOrder)* with its solution

not NAC-confluent for this solution. This is because *closeAccountTable* has a NAC *noOrders* which is not present in the NACs of the critical pair. This means that the graph consisting of a table with an open account could be embedded into a graph with a table with an open account and some order already present. In this case it is not possible to apply the same strict solution to the produce-forbid-conflict, since there are too many orders on the table. This example demonstrates very nicely why in this case the minimal context in which the conflict is resolved is not sufficient to resolve the conflict also in any other valid context in the same way.

Theorem 2 (Local Confluence Theorem - Critical Pair Lemma with NACs [9]). *Given a gts with NACs, it is locally confluent if all its critical pairs are strictly NAC-confluent.*

Example 6. Our running example transformation system *Restaurant* is locally confluent, since all critical pairs are strictly NAC-confluent. Consider again the Conflict Matrix of *Restaurant* as depicted in Fig. 4.

- All critical pairs on the diagonal of the matrix are trivially strictly NAC-confluent since they are of the form $P \overset{p,g}{\Leftarrow} K \overset{p,g}{\Rightarrow} P$.
- *(gatherOrder,closeAccountTable)* has been discussed in Example 5.
- *(closeAccountTable,gatherOrder)* denotes a critical pair in delete-use-conflict and can be resolved analogously to *(gatherOrder,closeAccountTable)*.
- *(closeAccountTable,alertUnpaidTable)* denotes a critical pair in delete-use-conflict and can be resolved, without transformation on the result of *closeAccountTable*, just applying the rules *resetAlert* and then *closeAccountTable* to the result of *alertUnpaidTable*. Thereby *closeAccountTable* can be applied since an alert can only be generated if no alert was there yet. Therefore after resetting this one alert, there will be no alert anymore on the table. Moreover there are no orderings on the table, since the table could be closed already at the start of the transformation.
- *(alertUnpaidTable,gatherOrder)* has been discussed in Example 5.
- *(alertUnpaidTable,payOrder)* will be discussed in Example 8.

- *(alertUnpaidTable,closeAccountTable)* denotes a critical pair in produce-forbid-conflict and can be resolved analogously to *(closeAccountTable, alertUnpaidTable)*.
- *(resetAlert,noteUnpaidOrder)* has been discussed in Example 5

Given a critical pair and a strict solution for it, it would be desirable to have a constructive method to check for NAC-confluence of this solution. Therefore the following theorem formulates a constructive sufficient condition for a critical pair $P_1 \overset{p_1}{\Leftarrow} K \overset{p_2}{\Rightarrow} P_2$ which is strictly confluent via some transformations $t_1 : K \overset{p_1,g_1}{\Rightarrow} P_1 \Rightarrow^* X$ and $t_2 : K \overset{p_2,g_2}{\Rightarrow} P_2 \Rightarrow^* X$ to be also NAC-confluent for t_1 and t_2. This means by definition that for every injective extension morphism $k_0 : K \rightarrow G$ which is NAC-consistent w.r.t. $K \overset{p_1,g_1}{\Rightarrow} P_1$ and $K \overset{p_2,g_2}{\Rightarrow} P_2$ it follows that k_0 is also NAC-consistent w.r.t. t_1 and t_2. In the following theorem two different conditions on each single $NAC(n_{1,j})$ (resp. $NAC(n_{2,j})$) of the concurrent NAC induced by transformation t_1 (resp. t_2) are given which lead to NAC-confluence if one of them is satisfied. The first condition expresses that there exists a suitable NAC on p_1 (resp. p_2) which evokes the satisfaction of $NAC(n_{1,j})$ (resp. $NAC(n_{2,j})$). The second condition first asks for a suitable morphism between the lhs's of the concurrent rules induced by both transformations t_1 and t_2. Moreover it expresses that there exists a suitable NAC on p_2 (resp. p_1) which evokes the satisfaction of $NAC(n_{1,j})$ (resp. $NAC(n_{2,j})$). Note that in the following theorem Def. 6 is used in order to refer to a concurrent rule, match and lhs-match induced by a transformation t, and it is referred to the downward translation of a NAC on L via a morphism $m_c : L \rightarrow L_c$ to a set of equivalent NACs on L_c denoted as $D_{m_c}(NAC_L)$ and defined explicitly in [7].

Theorem 3 (Sufficient Condition for NAC-confluence). *Given a critical pair* $P_1 \overset{p_1}{\Leftarrow} K \overset{p_2}{\Rightarrow} P_2$ *which is strictly confluent via the transformations* $t_1 : K \overset{p_1,g_1}{\Rightarrow} P_1 \Rightarrow^* X$ *and* $t_2 : K \overset{p_2,g_2}{\Rightarrow} P_2 \Rightarrow^* X$. *Let* $L_{c,1}$ *(resp.* $L_{c,2}$*) be the left-hand side of the concurrent rule* $p_{c,1}$ *(resp.* $p_{c,2}$*),* $m_{c,1} : L_1 \rightarrow L_{c,1}$ *(resp.* $m_{c,2} : L_2 \rightarrow L_{c,2}$*) the lhs-match and* $g_{c,1} : L_{c,1} \rightarrow K$ *(resp.* $g_{c,2} : L_{c,2} \rightarrow K$*) the concurrent match induced by* t_1 *(resp.* t_2*). Then the critical pair* $P_1 \overset{p_1}{\Leftarrow} K \overset{p_2}{\Rightarrow} P_2$ *is also NAC-confluent for* t_1 *and* t_2 *and thus strictly NAC-confluent if one of the following conditions holds for each* $NAC(n_{1,j}) : L_{c,1} \rightarrow N_{1,j}$ *(resp.* $NAC(n_{2,j}) : L_{c,2} \rightarrow N_{2,j}$*) of the concurrent* $NAC_{p_{c,1}}$ *induced by* t_1 *(resp.* $NAC_{p_{c,2}}$ *induced by* t_2*)*

- *there exists a* $NAC(n'_{1,i}) : L_{c,1} \rightarrow N'_{1,i}$ *(resp.* $NAC(n'_{2,i}) : L_{c,2} \rightarrow N'_{2,i}$*) in* $D_{m_{c,1}}(NAC_{L_1})$ *(resp.* $D_{m_{c,2}}(NAC_{L_2})$*) and an injective morphism* $d_{ij} : N'_{1,i} \rightarrow N_{1,j}$ *(resp. an injective morphism* $d_{ij} : N'_{2,i} \rightarrow N_{2,j}$*) such that (1) (resp. (1')) commutes.*
- *there exists a morphism* $l_{21} : L_{c,2} \rightarrow L_{c,1}$ *(resp.* $l_{12} : L_{c,1} \rightarrow L_{c,2}$*) s.t. (2) (resp. (2')) commutes and in addition a* $NAC(n'_{2,i}) : L_{c,2} \rightarrow N'_{2,i}$ *(resp.* $n'_{1,i} : L_{c,1} \rightarrow N'_{1,i}$*) in* $D_{m_{c,2}}(NAC_{L_2})$ *(resp.* $D_{m_{c,1}}(NAC_{L_1})$*) with an injective morphism* $m_{ij} : N'_{2,i} \rightarrow N_{1,j}$ *(resp. an injective morphism* $m_{ij} : N'_{1,i} \rightarrow N_{2,j}$*) s.t.* $n_{1,j} \circ l_{21} = m_{ij} \circ n'_{2,i}$ *(resp.* $n_{2,j} \circ l_{12} = m_{ij} \circ n'_{1,i}$*).*

$$\begin{tikzcd}
N_{1,j} & N'_{1,i} \arrow[l, dotted, "d_{ij}"'] & (3) & N'_{2,i} \arrow[lll, dotted, bend right, "m_{ij}"'] \\
(1) & L_{c,1} \arrow[ul, "n_{1,j}"] \arrow[u, "n'_{1,i}"'] \arrow[dr, "g_{c,1}"'] & (2) & L_{c,2} \arrow[ll, dotted, "l_{21}"] \arrow[u, "n'_{2,i}"'] \arrow[dl, "g_{c,2}"] \\
& & K &
\end{tikzcd}
\qquad
\begin{tikzcd}
N'_{1,i} \arrow[rrr, dotted, bend left, "m_{ij}"] & (3') & N'_{2,i} \arrow[r, dotted, "d_{ij}"] & N_{2,j} \\
L_{c,1} \arrow[u, "n'_{1,i}"'] \arrow[rr, dotted, "l_{12}"] \arrow[dr, "g_{c,1}"'] & (2') & L_{c,2} \arrow[u, "n'_{2,i}"] \arrow[ur, "n_{2,j}"'] \arrow[dl, "g_{c,2}"] & (1') \\
& K & &
\end{tikzcd}$$

Example 7
- Consider again the critical pair *(gatherOrder,closeAccountTable)* as described in Example 5. The strict solution consists on the one hand of a transformation that pays the order and then closes the account and on the other hand nothing. Thus for the solution of this critical pair t_2 merely consists of $t_2 : K \overset{closeAccountTable}{\Rightarrow} P_2$ and therefore this case is trivial. Thus it remains to consider the diagram on the left in Theorem 3. Namely, on the other hand $t_1 : K \overset{gatherOrder}{\Rightarrow} K_1 \overset{payOrder}{\Rightarrow} K_2 \overset{closeAccount}{\Rightarrow} P_2$ is not trivial. The concurrent rule $p_{c,1}$ of this transformation closes the account of a table with a concurrent $NAC_{p_{c,1}}$ consisting of a $NAC_{n_{1,1}}$ forbidding any alert and a single NAC $NAC_{n_{1,2}}$ forbidding any order for this table. Now $NAC_{n_{1,1}}$ is induced on the one hand by the downward translation of NAC *noAlert* of rule *gatherOrder*, since they can be connected by an identity i.e. $d_{1,1} = id$. On the other hand $NAC_{n_{1,2}}$ is induced by NAC *noOrders* on rule *closeAccountTable*. This is because the lhs $L_{c,1}$ of the concurrent rule $p_{c,1}$ consists of an open table and it is thus identical to the lhs $L_{c,2}$ of *closeAccountTable*. Therefore the morphism l_{21} is in this case the identity as well as the morphism m_{12} connecting both single NACs.
- Consider also the critical pair *(resetAlert,noteUnpaidOrder)* as described in Example 5. The sufficient condition as described in Theorem 3 is not fulfilled for the strict solution described in Example 5. It was possible though to conclude NAC-confluence according to Def. 10 for this solution anyway as explained in Example 5.

The following corollary follows directly from Theorem 3. It states that NAC-confluence for a critical pair is automatically fulfilled if a strict solution can be found via rules without NACs.

Corollary 1. *A critical pair $P_1 \overset{p_1}{\Leftarrow} K \overset{p_2}{\Rightarrow} P_2$ is strictly NAC-confluent if it is strictly confluent via the transformations $t_1 : K \overset{p_1,g_1}{\Rightarrow} P_1 \Rightarrow^* X$ (resp. $t_2 : K \overset{p_2,g_2}{\Rightarrow} P_2 \Rightarrow^* X$) and both $P_1 \Rightarrow^* X$ and $P_2 \Rightarrow^* X$ are transformation sequences without NACs.*

Example 8. Consider now in the Conflict Matrix in Fig. 4 the critical pair corresponding to *(alertUnpaidTable,payOrder)*. A strict solution for this critical pair on the one hand notes the unpaid order and resets the alert and on the other hand does nothing. Then we have a table with an open account and without orders. The rules *noteUnpaidOrder* and *resetAlert* are rules without NACs therefore according to the former corollary this critical pair is automatically strictly NAC-confluent and does not have to be investigated further.

5 Conclusion

In this paper, the Embedding Theorem and Local Confluence Theorem formulated in [6] for graph transformations without NACs are extended to graph transformations with NACs. These results hold not only for the instantiation of double-pushout gts with NACs as shown in this paper, but also for more general adhesive HLR systems with NACs as proven in [9]. In our results including NACs extra conditions such as NAC-consistency of the extension morphism (resp. NAC-confluence of the set of critical pairs) are required in order to lead to a correct embedding with NACs (resp. confluent gts with NACs). These additional conditions are explained in our running example.

Future work consists of trimming the results towards efficient tool support and generalizing the theory for transformation systems with NACs described in [9] to transformation systems with more general application conditions as defined in [13].

References

1. Taentzer, G., Ehrig, K., Guerra, E., de Lara, J., Lengyel, L., Levendovsky, T., Prange, U., Varro, D., Varro-Gyapay, S.: Model Transformation by Graph Transformation: A Comparative Study. In: Proc. Workshop Model Transformation in Practice, Montego Bay, Jamaica (October 2005)
2. Bottoni, P., Schürr, A., Taentzer, G.: Efficient Parsing of Visual Languages based on Critical Pair Analysis and Contextual Layered Graph Transformation. In: Proc.IEEE Symposium on Visual Languages, Long version available as technical report SI-2000-06, University of Rom (September 2000)
3. Huet, G.: Confluent reductions: Abstract properties and applications to term rewriting systems. JACM 27(4), 797–821 (1980)
4. Plump, D.: Hypergraph Rewriting: Critical Pairs and Undecidability of Confluence. In: Sleep, M., Plasmeijer, M., van Eekelen, M.C. (eds.) Term Graph Rewriting, pp. 201–214. Wiley, Chichester (1993)
5. Plump, D.: Confluence of graph transformation revisited. In: Middeldorp, A., van Oostrom, V., van Raamsdonk, F., de Vrijer, R. (eds.) Processes, Terms and Cycles: Steps on the Road to Infinity. LNCS, vol. 3838, pp. 280–308. Springer, Heidelberg (2005)
6. Ehrig, H., Ehrig, K., Prange, U., Taentzer, G.: Fundamentals of Algebraic Graph Transformation. EATCS Monographs in Theoretical Computer Science. Springer, Heidelberg (2006)
7. Lambers, L., Ehrig, H., Orejas, F., Prange, U.: Parallelism and Concurrency in Adhesive High-Level Replacement Systems with Negative Application Conditions. In: Ehrig, H., Pfalzgraf, J., Prange, U. (eds.) CC 2007. Elsevier, Amsterdam (to appear, 2008)

8. Lambers, L., Ehrig, H., Orejas, F.: Conflict Detection for Graph Transformation with Negative Application Conditions. In: Corradini, A., Ehrig, H., Montanari, U., Ribeiro, L., Rozenberg, G. (eds.) ICGT 2006. LNCS, vol. 4178, pp. 61–76. Springer, Heidelberg (2006)
9. Lambers, L.: Adhesive high-level replacement systems with negative application conditions. Technical report, Technische Universität Berlin (2007), `http://iv.tu-berlin.de/TechnBerichte/2007/2007-14.pdf`
10. Habel, A., Heckel, R., Taentzer, G.: Graph grammars with negative application conditions 26, 287–313 (1996)
11. Corradini, A., Montanari, U., Rossi, F., Ehrig, H., Heckel, R., Löwe, M.: Algebraic Approaches to Graph Transformation I: Basic Concepts and Double Pushout Approach. In: Rozenberg, G. (ed.) Handbook of Graph Grammars and Computing by Graph Transformation, Foundations, vol. 1, pp. 163–245. World Scientific, Singapore (1997)
12. Taentzer, G.: AGG: A Graph Transformation Environment for Modeling and Validation of Software. In: Pfaltz, J.L., Nagl, M., Böhlen, B. (eds.) AGTIVE 2003. LNCS, vol. 3062, pp. 446–456. Springer, Heidelberg (2004)
13. Ehrig, H., Ehrig, K., Habel, A., Pennemann, K.H.: Constraints and application conditions: From graphs to high-level structures. In: Ehrig, H., Engels, G., Parisi-Presicce, F., Rozenberg, G. (eds.) ICGT 2004. LNCS, vol. 3256, pp. 287–303. Springer, Heidelberg (2004)