Confluence detection for transformations of labelled transition systems

Confluence Detection for Transformations of Labelled Transition Systems

Anton Wijs
Department of Mathematics and Computer Science
Eindhoven University of Technology
P.O. Box 513, 5600 MB, Eindhoven, The Netherlands

A.J.Wijs@tue.nl

The development of complex component software systems can be made more manageable by first creating an abstract model and then incrementally adding details. Model transformation is an approach to add such details in a controlled way. In order for model transformation systems to be useful, it is crucial that they are confluent, i.e. that when applied on a given model, they will always produce a unique output model, independent of the order in which rules of the system are applied on the input. In this work, we consider Labelled Transition Systems (LTSs) to reason about the semantics of models, and LTS transformation systems to reason about model transformations. In related work, the problem of confluence detection has been investigated for general graph structures. We observe, however, that confluence can be detected more efficiently in special cases where the graphs have particular structural properties. In this paper, we present a number of observations to detect confluence of LTS transformation systems, and propose both a new confluence detection algorithm and a conflict resolution algorithm based on them.

1 Introduction

In Model-Driven Software Development, model transformation is a well-known technique to incrementally construct complex, often concurrent systems through manageable steps. It allows reasoning about a system at a high level of abstraction, and incrementally adding more information until a model has been constructed from which source code can be automatically derived. Some transformations add details or components to an existing model of a system under development, others refactor a model to make it easier to interpret, or translate a model to one written in a different modelling language. To reason about model transformations, often graph transformation is chosen as the underlying mechanism [7, 23].

It is crucial, though, that transformations are verifiable, i.e. that the definitions of transformations can be qualitatively analysed. Much work has been done on verifying model transformations, e.g. [10, 16], using many different techniques [1, 22]. In earlier work, we have developed a formal verification technique to determine whether the definition of a model transformation preserves specific safety or liveness properties, *regardless* of the model it is applied on [6, 24–26]. It is applicable on any modelling language with a formal semantics that can be captured by *Labelled Transition Systems* (LTSs), i.e. it must be action (or event) based. For example, in our research we focus on the visual modelling language SLCO [5], which allows the specification and development of concurrent and distributed systems by defining sets of (interacting) finite state machines.

In our setting, the semantics of models is captured by LTSs, and the semantics of transformations is captured by systems of pairs of LTSs, describing which patterns in an input LTS should be transformed into which new patterns. By applying the technique from [6, 24–26] on those pairs of LTSs, we are able to determine whether transformation is guaranteed to preserve the structure of any LTS w.r.t. a particular

A. Rensink & E. Zambon (Eds.): Graphs as Models 2015 (GaM'15)
EPTCS 181, 2015, pp. 1–15, doi:10.4204/EPTCS.181.1

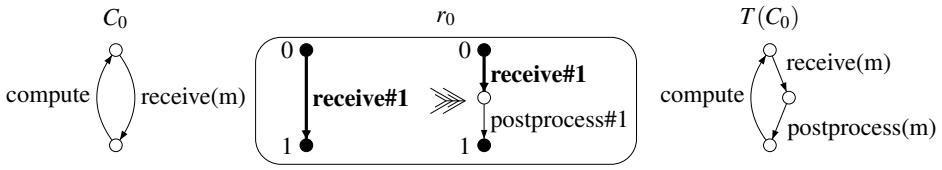

Figure 1: Example of an LTS transformation

temporal logic formula expressing a desired functional property. If that is the case, then models for which this property holds can be safely transformed.

For example, consider the LTS C_0 on the left in Figure 1. It describes a system that can alternatingly receive messages m and perform computations. The transformation rule r_0 in the middle defines that after each receiving of a message, a postprocessing step should be added. The result of applying the rule on the LTS on the left in the figure, produces the LTS $T(C_0)$ on the right. More interesting cases involve multiple LTSs describing the semantics of different components running concurrently, and interaction between those components is taken into account.

This setting allows to formally reason about model transformations, which has a number of practical applications. For instance, if one desires to develop a system running on specific hardware, then an abstract model can be transformed to make it compatible with that hardware. In [24], a transformation rule system is given to transform multi-party communication in models, i.e. involving more than two parties at once, into a number of two-party communications following a specific protocol. Multi-party communication can be useful in modelling languages to reason about system behaviour at an abstract level, but if the eventual implementation cannot use it, then such a transformation rule system is useful to automatically remove it at some point in the development process. The ability to verify the transformation rule system in isolation means that we are able to determine that it will always produce a correct output model when applied on a correct input model. Other elaborate examples of LTS transformations are given in [25, 26].

For model transformation, it is crucial that the transformation is always *terminating* and *confluent*, i.e. that transformation is guaranteed to finish, and that it always leads to the same solution, i.e. reduct, independent of the order in which matches are processed. This is important, since a user defining how a particular system should be transformed typically has a specific resulting model in mind. Therefore, if a rule system is not confluent, it usually means that the user made some mistake. Both termination and confluence of transformation systems has been studied before; for instance, in [3], criteria are given to determine whether a system is terminating or not.

Confluence has been the subject of research in for example [12, 13, 19–21]. From term rewriting, we know that a rewrite system is confluent if it is both terminating and locally confluent [9]. A system is locally confluent iff all possible conflicts between two transformation applications, i.e. direct transformations, can be resolved according to the Critical Pair Lemma [19–21]. For a terminating system, determining confluence therefore boils down to doing two operations: firstly, to construct so-called *critical pairs* representing possible conflicts between two direct transformations, and secondly, to try to resolve all constructed critical pairs.

In earlier work on critical pair detection, general graphs have always been considered, meaning that vertices and edges may or may not have labels, edges may or may not be directed, and graphs can consist of several disconnected subgraphs. Such a general approach is of course very useful, but when considering a more restricted setting, it may be possible to detect critical pairs more efficiently. The complexity of standard critical pair detection for general graph structures is exponential in the number of

vertices and edges in the left patterns of two transformation rules, since all possible overlappings between the two left patterns need to be taken into account. In [12], it is demonstrated that if one transformation rule deletes elements and the other does not, a check with a linear complexity can be obtained, and when both rules do not delete elements, a check with quadratic complexity is possible.

The contributions of this paper follow from the observation that in our setting, graphs are directed, edge-labelled graphs, i.e. LTSs, with all vertices connected with each other via edges (ignoring the direction). These structural properties can be exploited further, along the reasoning of [12], to find critical pairs more efficiently; the presence of edge labels allows to check in constant time in some cases, and furthermore, we are also able to define a check for critical pairs with quadratic complexity for the case that two rules both delete elements. The main conclusion for our setting is that transitions, as opposed to states, turn out to play a crucial role in the detection of critical pairs.

Contribution In this paper, we propose a new critical pair detection algorithm when working with LTSs as opposed to general graphs. When formally reasoning about model transformations as transformations of LTSs, we can immediately benefit from the new algorithm since it can handle many cases more efficiently than existing detection techniques [12,13,19–21]. It uses a novel approach, constructing partial morphisms between LTSs. Whenever such a partial morphism meets certain requirements, a conflict can be directly derived from it. Although the worst-case complexity of the algorithm is comparable to that of algorithms proposed in related work, it is very efficient in particular cases. The circumstances of those cases are explained in detail. Besides that, we also propose an algorithm to try to resolve detected conflicts, which is based on observations made in [21], but we focus on our particular setting.

Roadmap Section 2 presents the basic notions, in particular LTS and LTS transformation. In Section 3, we investigate conflict detection. From this, we construct a conflict detection algorithm in Section 4 and a conflict resolution algorithm in the same section. Finally, Section 5 contains our conclusions and pointers for future work.

2 Background

In this paper, we focus on action-based semantics of (concurrent) systems. Such semantics are often captured using *Labelled Transition Systems* (LTSs), indicating how a system as a whole or an individual component in a system can change state by performing particular actions.

Definition 1 (Labelled Transition System) *An LTS $\mathcal{G}$ is a tuple $\langle \mathcal{S}_\mathcal{G}, \mathcal{A}_\mathcal{G}, \mathcal{T}_\mathcal{G} \rangle$, where $\mathcal{S}_\mathcal{G}$ is a (finite) set of states, $\mathcal{A}_\mathcal{G}$ is a set of actions, and $\mathcal{T}_\mathcal{G} \subseteq \mathcal{S}_\mathcal{G} \times \mathcal{A}_\mathcal{G} \times \mathcal{S}_\mathcal{G}$ is a transition relation. Actions in $\mathcal{A}_\mathcal{G}$ are denoted by a, b, c, etc. We use $s_1 \xrightarrow{a}_\mathcal{G} s_2$ to denote $\langle s_1, a, s_2 \rangle \in \mathcal{T}_\mathcal{G}$. If $s_1 \xrightarrow{a}_\mathcal{G} s_2$, this means that in $\mathcal{G}$, an action a can be performed in state s_1, leading to state s_2.*

We use operations on LTSs such as intersection and difference in the usual graph-theoretical way. Finally, an LTS is *weakly connected* iff the undirected version of an LTS is a single connected component (from each state, there is path to each other state).

In the context of systems consisting of a finite number of concurrent components, we actually represent system semantics as *networks of LTSs* [14], where the potential behaviour of each component is described by a separate LTS, and a synchronisation mechanism is defined describing the potential for those LTSs to interact. For example, consider the network on the left in Figure 2, which besides LTS C_0 from Figure 1 also contains the potential behaviour of a component C_1 that can at any time send a

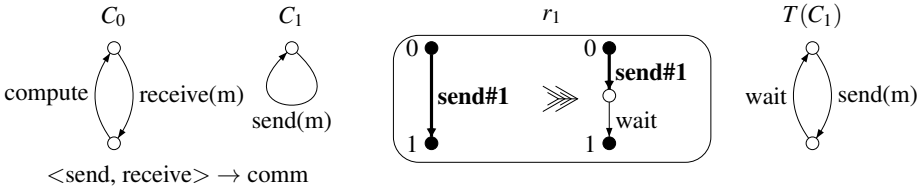

Figure 2: Transforming networks of LTSs

message. Below the LTSs, a synchronisation rule is defined stating that *send* and *receive* actions can synchronise, leading to a *comm* action in the LTS of the system. For this to be possible, the parameters of *send* and *receive* must be identical, i.e. they must involve the same message *m*.

When considering transformation rule systems applicable on networks of LTSs, confluence depends on whether the rules in the rule system do not give rise to conflicts in either of the individual LTSs, so it again boils down to considering single LTSs. For this reason, we do not consider networks of LTSs in most of this paper.

Transformation In our setting, changes applied on a concurrent system model are represented by LTS *transformation rules* applied on the semantics of the components of the model, i.e. on their LTSs. We only consider weakly connected LTSs as the semantics of components, since naturally, the possible states that components can be in should be reachable from their initial state. To reason about the changes, we define the notions of a rule, and matches of rules on component LTSs. But first, we introduce the notion of *LTS morphisms*.

Definition 2 (LTS morphism) *An* LTS morphism $f : \mathcal{G}_0 \to \mathcal{G}_1$ *between two LTSs* $\mathcal{G}_0 = \langle \mathcal{S}_{\mathcal{G}_0}, \mathcal{A}_{\mathcal{G}_0}, \mathcal{T}_{\mathcal{G}_0} \rangle$, $\mathcal{G}_1 = \langle \mathcal{S}_{\mathcal{G}_1}, \mathcal{A}_{\mathcal{G}_1}, \mathcal{T}_{\mathcal{G}_1} \rangle$ *is a pair of functions* $f = \langle f_{\mathcal{S}} : \mathcal{S}_{\mathcal{G}_0} \to \mathcal{S}_{\mathcal{G}_1}, f_{\mathcal{T}} : \mathcal{T}_{\mathcal{G}_0} \to \mathcal{T}_{\mathcal{G}_1} \rangle$ *which preserve sources, targets, and transition labels, i.e. for all* $s \xrightarrow{a}_{\mathcal{G}_0} s'$, *we have* $f_{\mathcal{T}}(s \xrightarrow{a}_{\mathcal{G}_0} s') = f_{\mathcal{S}}(s) \xrightarrow{a}_{\mathcal{G}_1} f_{\mathcal{S}}(s')$.

We denote the existence of an *injective* LTS morphism f from an LTS $\mathcal{G}_0$ to an LTS $\mathcal{G}_1$, meaning that $f_{\mathcal{S}}$ and $f_{\mathcal{T}}$ are injective, by $\mathcal{G}_0 \sqsubseteq \mathcal{G}_1$, and say that $\mathcal{G}_0$ and $\mathcal{G}_1$ are *isomorphic*, denoted by $\mathcal{G}_0 \simeq \mathcal{G}_1$, iff there exists a morphism $f : \mathcal{G}_0 \to \mathcal{G}_1$ such that both $f_{\mathcal{S}}$ and $f_{\mathcal{T}}$ are bijections. An *LTS inclusion* $i : \mathcal{G}_0 \to \mathcal{G}_1$ is an LTS morphism with for all $s \in \mathcal{S}_{\mathcal{G}_0}$, $i_{\mathcal{S}}(s) = s$, and for all $s \xrightarrow{a}_{\mathcal{G}_0} s'$, $i_{\mathcal{T}}(s \xrightarrow{a}_{\mathcal{G}_0} s') = s \xrightarrow{a}_{\mathcal{G}_1} s'$. LTS $\mathcal{G}_0$ is a sub-LTS of $\mathcal{G}_1$, denoted by $\mathcal{G}_0 \subseteq \mathcal{G}_1$, iff there exists an LTS inclusion $i : \mathcal{G}_0 \to \mathcal{G}_1$. Finally, we denote the fact that a morphism is undefined for a particular state or transition with $\perp$, for example $f_{\mathcal{S}}(s) = \perp$ means that $f_{\mathcal{S}}(s)$ is undefined.

Definition 3 (Transformation Rule) *A* transformation rule $r = \langle \mathcal{L} \xleftarrow{f} \mathcal{K} \xrightarrow{g} \mathcal{R} \rangle$ *consists of two LTS morphisms* $f : \mathcal{K} \to \mathcal{L}$, $g : \mathcal{K} \to \mathcal{R}$, *where* $\mathcal{K} \to \mathcal{L}$ *is an inclusion. LTSs* $\mathcal{L}$ *and* $\mathcal{R}$ *are both weakly connected, and are called the* left *and* right patterns *of r. LTS* $\mathcal{K}$ *is the* interface.

We consider injective transformation rules, meaning that $\mathcal{K} \to \mathcal{R}$ is injective. With $\mathcal{S}_{\mathcal{L} \setminus \mathcal{K}}$, we refer to the states s in $\mathcal{L}$ that are not represented in $\mathcal{K}$, i.e. $f_{\mathcal{S}}^{-1}(s) = \perp$. A similar convention is used for the functions $f_{\mathcal{T}}$, $g_{\mathcal{S}}$, and $g_{\mathcal{T}}$. States $s \in \mathcal{S}_{\mathcal{L}}$ for which $f_{\mathcal{S}}^{-1}(s)$ is defined are called *glue-states*.

Definition 4 (Rule Match) *A transformation rule* $r = \langle \mathcal{L} \xleftarrow{f} \mathcal{K} \xrightarrow{g} \mathcal{R} \rangle$ *has a* match $m : \mathcal{L} \to \mathcal{G}$ *on an LTS* $\mathcal{G} = \langle \mathcal{S}_{\mathcal{G}}, \mathcal{A}_{\mathcal{G}}, \mathcal{T}_{\mathcal{G}} \rangle$ *iff* $m : \mathcal{L} \to \mathcal{G}$ *is an injective LTS morphism and* $\forall s \in \mathcal{S}_{\mathcal{L} \setminus \mathcal{K}}, p \in \mathcal{S}_{\mathcal{G}}$:

- $m_{\mathcal{S}}(s) \xrightarrow{a}_{\mathcal{G}} p \implies \exists s' \in \mathcal{S}_{\mathcal{L}}. s \xrightarrow{a}_{\mathcal{L}} s' \wedge m_{\mathcal{S}}(s') = p$;

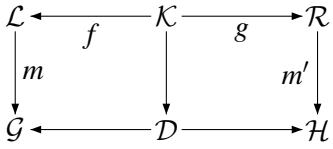

Figure 3: Double-pushout diagram

- $p \xrightarrow{a}_{\mathcal{G}} m_{\mathcal{S}}(s) \implies \exists s' \in \mathcal{S}_{\mathcal{L}}.s' \xrightarrow{a}_{\mathcal{L}} s \wedge m_{\mathcal{S}}(s') = p.$

The conditions in Def. 4 correspond with the *gluing conditions* of the *double-pushout* (DPO) method [4] for graph transformation, preventing so-called *dangling transitions*, which are transitions where only the source or target state will be removed, but not both. It expresses that for a state s to be removed, all connected transitions must be removed as well.

Let $\mathcal{G}$, $\mathcal{H}$ be LTSs, and $m : \mathcal{L} \to \mathcal{G}$ a match for rule r. Then $\mathcal{G}$ *directly transforms* $\mathcal{H}$ by r and m, denoted by $\mathcal{G} \Rightarrow_{r,m} \mathcal{H}$, iff there are two pushouts as in Figure 3.

Direct transformation is defined as follows, with $m' : \mathcal{R} \to \mathcal{H}$ a match between the right pattern and the result of the transformation.

Definition 5 (Direct Transformation) *The* direct transformation $\mathcal{G} \Rightarrow_{r,m} \mathcal{H}$ *of an LTS* $\mathcal{G} = \langle \mathcal{S}_{\mathcal{G}}, \mathcal{A}_{\mathcal{G}}, \mathcal{T}_{\mathcal{G}} \rangle$ *according to a rule r and a given match* $m : \mathcal{L} \to \mathcal{G}$ *is defined as* $\mathcal{H} = \langle \mathcal{S}_{\mathcal{H}}, \mathcal{A}_{\mathcal{H}}, \mathcal{T}_{\mathcal{H}} \rangle$*, where*

- $\mathcal{S}_{\mathcal{H}} = (\mathcal{S}_{\mathcal{G}} \setminus \{m_{\mathcal{S}}(s) \mid s \in (\mathcal{S}_{\mathcal{L} \setminus \mathcal{K}})\}) \cup (\mathcal{S}_{\mathcal{R} \setminus \mathcal{K}})$*;*
- $\mathcal{T}_{\mathcal{H}} = (\mathcal{T}_{\mathcal{G}} \setminus \{m_{\mathcal{T}}(\langle s,a,s' \rangle) \mid s \xrightarrow{a}_{\mathcal{L} \setminus \mathcal{K}} s'\}) \cup \{m'_{\mathcal{T}}(\langle s,a,s' \rangle) \mid s \xrightarrow{a}_{\mathcal{R} \setminus \mathcal{K}} s'\}$*;*
- $\mathcal{A}_{\mathcal{H}} = \{a \mid \exists \langle s,a,s' \rangle \in \mathcal{T}_{\mathcal{H}}\}$.

The new set of states $\mathcal{S}_{\mathcal{H}}$ consists of $\mathcal{S}_{\mathcal{G}}$ without the states that correspond to the states in the left pattern that are not represented in the interface, i.e. the removed states, and with new representatives, here represented in the match m', of the states in the right pattern that are not represented in the interface, i.e. the newly added states. In a similar way, $\mathcal{T}_{\mathcal{H}}$ consists of the transitions in $\mathcal{T}_{\mathcal{G}}$ without the transitions corresponding to left pattern transitions that are not represented in the interface, and with transitions corresponding to right pattern transitions that are not represented in the interface.

An example of a transformation rule introducing a new action is given in the middle of Figure 1. Black states with the same index correspond with each other, i.e. for two such states $s \in \mathcal{S}_{\mathcal{L}}$, $t \in \mathcal{S}_{\mathcal{R}}$, we have $g_{\mathcal{S}}(f_{\mathcal{S}}^{-1}(s)) = t$. This also holds for the highlighted transition labelled *receive(m)*.

It is crucial to note at this point that we expect transformation rules to be *well-specified*, i.e. that they specify that input is actually altered, and not replaced by something that can be considered equivalent. In particular, we assume that a transition $s \xrightarrow{a}_{\mathcal{G}} s$ is not replaced by a new transition between states s and s' with the same label a, nor that a state s is replaced by another state $\hat{s}$ without transforming any of the transitions connected to s (note that LTSs to be transformed are weakly connected, so s always has connected transitions).

Sets of rules together make up a *rule system* Σ. Transformation of an LTSs $\mathcal{G}$ according to a rule system Σ involves identifying all possible matches for each $r \in \Sigma$ on $\mathcal{G}$, and applying transformation on those matches. A *transformation* from $\mathcal{G}$ to $\mathcal{H}$ is a sequence of direct transformations $\mathcal{G} = \mathcal{G}_0 \Rightarrow \ldots \Rightarrow \mathcal{G}_n = \mathcal{H}$, with $n \geq 0$. We denote this by $\mathcal{G} \Rightarrow^*_{\Sigma} \mathcal{H}$.

Figure 2 presents a transformation rule r_1 which can only be applied on C_1, leading to LTS $T(C_1)$. When combining r_1 with r_0 into a rule system Σ, it is clear that Σ is confluent w.r.t. the network given in Figure 2, since r_0 and r_1 are not applicable on the same LTSs. The question that remains is whether Σ is confluent for arbitrary single LTSs as well.

3 Conflicts Between Direct Transformations

By Newman's Lemma [17], a terminating transformation system is confluent iff it is *locally confluent*, i.e. if for all direct transformations $\mathcal{H}_0 \Leftarrow_{r_0,m_0} \mathcal{G} \Rightarrow_{r_1,m_1} \mathcal{H}_1$, there is a common reduct $\mathcal{H}$ with $\mathcal{H}_0 \Rightarrow_\Sigma^* \mathcal{H}$ and $\mathcal{H}_1 \Rightarrow_\Sigma^* \mathcal{H}$. To determine local confluence, first of all, it has been shown [19] that if two direct transformations are *parallel independent*, then they are locally confluent. In the following, we reason about two LTS transformation rules $r_0 = \langle \mathcal{L}^{r_0}, \mathcal{R}^{r_0} \rangle$ and $r_1 = \langle \mathcal{L}^{r_1}, \mathcal{R}^{r_1} \rangle$.

Definition 6 (Parallel Independence) *Direct transformations $\mathcal{H}_0 \Leftarrow_{r_0,m_0} \mathcal{G} \Rightarrow_{r_1,m_1} \mathcal{H}_1$ are* parallel independent *iff*

$$m_0(\mathcal{L}^{r_0}) \cap m_1(\mathcal{L}^{r_1}) \subseteq m_0(\mathcal{K}^{r_0}) \cap m_1(\mathcal{K}^{r_1})$$

The intuition behind Def. 6 is that if two matches of one or two rules on an LTS only overlap w.r.t. the interfaces of those two rules, then the related direct transformations are parallel independent since applying one direct transformation does not invalidate the match for the other direct transformation. We say that two direct transformations are *in conflict* iff they are not parallel independent. The presence of a conflict can cause the transformation system to be not locally confluent (specifically, if the two derivations do not lead to a common reduct [4]). Informally, the conflict is caused because r_0 deletes something that r_1 uses and/or r_1 deletes something that r_0 uses. A concrete conflict can be represented by a *critical pair*, which defines an LTS on which two matches of the given pair of rules exist that imply derivations that are in conflict. Clearly, in such an LTS, the two matches must overlap.

Definition 7 (Critical Pair) *Direct transformations $\mathcal{H}_0 \Leftarrow_{r_0,m_0} \mathcal{G} \Rightarrow_{r_1,m_1} \mathcal{H}_1$ form a* critical pair *iff they are not parallel independent and $\mathcal{G} = m_0(\mathcal{L}^{r_0}) \cup m_1(\mathcal{L}^{r_1})$.*

Furthermore, we require that $m_0 \neq m_1$ if $r_0 = r_1$, and we equate isomorphic critical pairs, which informally means that two critical pairs are different if either there exists no isomorphism between their $\mathcal{G}$'s, or their matches m_0, m_1 are different.

The main task when detecting conflicts is to construct a suitable *conflict situation* $\mathcal{G}$ for pairs of rules r_0, r_1 that gives rise to a conflict. Such a $\mathcal{G}$ should be minimal, in the sense that there does not exist an LTS $\mathcal{G}'$ with $\mathcal{G}' \subset \mathcal{G}$ and matches $m_0' : \mathcal{L}^{r_0} \to \mathcal{G}'$, $m_1' : \mathcal{L}^{r_1} \to \mathcal{G}'$ such that $\mathcal{H}_0' \Leftarrow_{r_0,m_0'} \mathcal{G}' \Rightarrow_{r_1,m_1'} \mathcal{H}_1'$ is also a critical pair.

In this section, we focus on how to construct a suitable $\mathcal{G}$ efficiently. First, we establish that in order to have a conflict in $\mathcal{G}$, $m_0(\mathcal{L}^{r_0})$ and $m_1(\mathcal{L}^{r_1})$ must at least overlap in one transition. This relies on the fact that our LTSs $\mathcal{G}$ are weakly connected. If a rule specifies that all states matched on a left pattern state s should be removed, then so must all transitions that connect with those states. By the fact that such transitions always exist ($\mathcal{G}$ is weakly connected) and Def. 4, it follows that the rule must also specify explicitly that these transitions must be removed. Hence, a conflict between rules concerning the removal of states also must involve the removal of transitions.

Lemma 1 *Direct transformations $\mathcal{H}_0 \Leftarrow_{r_0,m_0} \mathcal{G} \Rightarrow_{r_1,m_1} \mathcal{H}_1$ are parallel independent iff*

$$\mathcal{T}_{m_0(\mathcal{L}^{r_0}) \cap m_1(\mathcal{L}^{r_1})} \subseteq \mathcal{T}_{m_0(\mathcal{K}^{r_0}) \cap m_1(\mathcal{K}^{r_1})}$$

Proof. The *if* case is trivial. If the LTS $m_0(\mathcal{L}^{r_0}) \cap m_1(\mathcal{L}^{r_1})$ is contained in the LTS $m_0(\mathcal{K}^{r_0}) \cap m_1(\mathcal{K}^{r_1})$, then all transitions of $m_0(\mathcal{L}^{r_0}) \cap m_1(\mathcal{L}^{r_1})$ are contained in $m_0(\mathcal{K}^{r_0}) \cap m_1(\mathcal{K}^{r_1})$.

For the *only if* case, we reason towards a contradiction. Assume that the direct transformations are not parallel independent, i.e. $m_0(\mathcal{L}^{r_0}) \cap m_1(\mathcal{L}^{r_1}) \not\subseteq m_0(\mathcal{K}^{r_0}) \cap m_1(\mathcal{K}^{r_1})$, but that $\mathcal{T}_{m_0(\mathcal{L}^{r_0}) \cap m_1(\mathcal{L}^{r_1})} \subseteq$

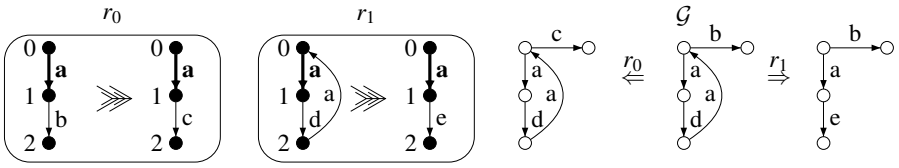

Figure 4: Two rules r_0 and r_1 with a possible conflict situation $\mathcal{G}$

$\mathcal{T}_{m_0(\mathcal{K}^{r_0})\cap m_1(\mathcal{K}^{r_1})}$. Then, we must have that $\mathcal{S}_{m_0(\mathcal{L}^{r_0})\cap m_1(\mathcal{L}^{r_1})} \not\subseteq \mathcal{S}_{m_0(\mathcal{K}^{r_0})\cap m_1(\mathcal{K}^{r_1})}$. We will prove that this cannot be the case by reasoning towards a contradiction. Let $p \in \mathcal{S}_{m_0(\mathcal{L}^{r_0})\cap m_1(\mathcal{L}^{r_1})}$ and $p \notin \mathcal{S}_{m_0(\mathcal{K}^{r_0})\cap m_1(\mathcal{K}^{r_1})}$. Then, there must exist $s \in \mathcal{S}_{\mathcal{L}^{r_0}}$ with $m_{0,\mathcal{S}}(s) = p$ and $t \in \mathcal{S}_{\mathcal{L}^{r_1}}$ with $m_{1,\mathcal{S}}(t) = p$. Since $p \notin \mathcal{S}_{m_0(\mathcal{K}^{r_0})\cap m_1(\mathcal{K}^{r_1})}$, we have $s \notin \mathcal{S}_{\mathcal{K}^{r_0}}$ and $t \notin \mathcal{S}_{\mathcal{K}^{r_1}}$. Because $\mathcal{G}$ is weakly connected and r_1 is well-specified, p must have at least one in- or outgoing transition which will be removed by the direct transformation $\mathcal{G} \Rightarrow_{r_1,m_1} \mathcal{H}_1$. Let us assume that this is an incoming transition $\hat{p} \xrightarrow{a}_{\mathcal{G}} p$ (the case of an outgoing transition is similar). Since $m_1(t) = p$ and $t \in \mathcal{S}_{\mathcal{L}^{r_1}\setminus\mathcal{K}^{r_1}}$, by Def. 4, there must be a transition $\hat{t} \xrightarrow{a}_{\mathcal{L}^{r_1}} t$, with $m_{1,\mathcal{S}}(\hat{t}) = \hat{p}$, and $m_{1,\mathcal{T}}(\hat{t} \xrightarrow{a}_{\mathcal{L}^{r_1}} t) = \hat{p} \xrightarrow{a}_{\mathcal{G}} p$. Similarly, there must be an $\hat{s} \in \mathcal{S}_{\mathcal{L}^{r_0}}$ with $\hat{s} \xrightarrow{a}_{\mathcal{L}^{r_0}} s$, $m_{0,\mathcal{S}}(\hat{s}) = \hat{p}$ and $m_{0,\mathcal{T}}(\hat{s} \xrightarrow{a}_{\mathcal{L}^{r_0}} s) = \hat{p} \xrightarrow{a}_{\mathcal{G}} p$. This means that both $\hat{p} \xrightarrow{a}_{m_0(\mathcal{L}^{r_0})} p$ and $\hat{p} \xrightarrow{a}_{m_1(\mathcal{L}^{r_1})} p$. Also, since $\hat{p} \xrightarrow{a}_{\mathcal{G}} p$ is set for removal, we must have $\langle \hat{p}, a, p\rangle \notin \mathcal{T}_{m_0(\mathcal{K}^{r_0})}$ and $\langle \hat{p}, a, p\rangle \notin \mathcal{T}_{m_1(\mathcal{K}^{r_1})}$. But then, $\mathcal{T}_{m_0(\mathcal{L}^{r_0})\cap m_1(\mathcal{L}^{r_1})} \not\subseteq \mathcal{T}_{m_0(\mathcal{K}^{r_0})\cap m_1(\mathcal{K}^{r_1})}$, and we have a contradiction. □

The following lemma expresses that parallel independence of rules r_0, r_1 can be concluded if the sets of transition labels of r_0 and r_1 satisfy certain conditions. We use $\mathcal{A}_{\mathcal{L}|\mathcal{K}}$ to refer to the labels for which there exists at least one transition in $\mathcal{L}$ that is not represented in $\mathcal{K}$, i.e. there exists an $s \xrightarrow{a}_{\mathcal{L}} s'$ for which $f_{\mathcal{T}}^{-1}(s \xrightarrow{a}_{\mathcal{L}} s') = \perp$.

Lemma 2 *Direct transformations $\mathcal{H}_0 \Leftarrow_{r_0,m_0} \mathcal{G} \Rightarrow_{r_1,m_1} \mathcal{H}_1$ are parallel independent if $\mathcal{A}_{\mathcal{L}^{r_0}|\mathcal{K}^{r_0}} \cap \mathcal{A}_{\mathcal{L}^{r_1}} = \emptyset$ and $\mathcal{A}_{\mathcal{L}^{r_1}|\mathcal{K}^{r_1}} \cap \mathcal{A}_{\mathcal{L}^{r_0}} = \emptyset$.*

Proof. By reasoning towards a contradiction. Assume that $(\mathcal{A}_{\mathcal{L}^{r_0}|\mathcal{K}^{r_0}}) \cap \mathcal{A}_{\mathcal{L}^{r_1}} = \emptyset$ and $(\mathcal{A}_{\mathcal{L}^{r_1}|\mathcal{K}^{r_1}}) \cap \mathcal{A}_{\mathcal{L}^{r_0}} = \emptyset$, but that the direct transformations are not parallel independent. From Lemma 1, it follows that $\mathcal{T}_{m_0(\mathcal{L}^{r_0})\cap m_1(\mathcal{L}^{r_1})}$ must be non-empty. So, there must exist a transition $s \xrightarrow{a}_{m_0(\mathcal{L}^{r_0})\cap m_1(\mathcal{L}^{r_1})} s'$ that is not in $m_0(\mathcal{K}^{r_0}) \cap m_1(\mathcal{K}^{r_1})$. From this, it follows that $a \in \mathcal{A}_{\mathcal{L}^{r_0}}$, $a \in \mathcal{A}_{\mathcal{L}^{r_1}}$ and $a \notin \mathcal{A}_{\mathcal{K}^{r_0}}$. But then, $\mathcal{A}_{\mathcal{L}^{r_0}|\mathcal{K}^{r_0}} \cap \mathcal{A}_{\mathcal{L}^{r_1}} \neq \emptyset$, and we have a contradiction. □

Lemma 2 will be used as a first check in a conflict detection algorithm in the next section. If for two rules, the mentioned intersection of action sets of left patterns is empty, then it is not possible to construct critical pairs. Since this can be checked in linear time, assuming that set membership can be checked in constant time, it helps to avoid more involved conflict detection for many cases in practice.

Consider the example illustrated in Figure 4. For the two rules r_0, r_1, we have $\mathcal{A}_{\mathcal{L}^{r_0}|\mathcal{K}^{r_0}} = \{b\}$ and $\mathcal{A}_{\mathcal{L}^{r_1}} = \{a,d\}$, hence $\mathcal{A}_{\mathcal{L}^{r_0}|\mathcal{K}^{r_0}} \cap \mathcal{A}_{\mathcal{L}^{r_1}} = \emptyset$, but $\mathcal{A}_{\mathcal{L}^{r_1}|\mathcal{K}^{r_1}} = \{a,d\}$ and $\mathcal{A}_{\mathcal{L}^{r_0}} = \{a,b\}$, so $\mathcal{A}_{\mathcal{L}^{r_1}|\mathcal{K}^{r_1}} \cap \mathcal{A}_{\mathcal{L}^{r_0}} = \{a\}$. This means that there is potential for a conflict situation, and a valid conflict situation is actually illustrated on the right in Figure 4. In the given LTS $\mathcal{G}$, applying the direct transformation defined by $\mathcal{L}^{r_1}$ matched on the lower part of $\mathcal{G}$ results in an LTS on which $\mathcal{L}^{r_0}$ can still be matched. However, note that $\mathcal{L}^{r_0}$ can be matched on $\mathcal{G}$ involving the curved a-transition and the b-transition. Since r_0 removes the matched a-transition, this means that the possible match of $\mathcal{L}^{r_1}$ on $\mathcal{G}$ is removed when applying the direct transformation of r_0.

Next, we concentrate on constructing minimal conflict situations $\mathcal{G}$ for pairs of rules r_0, r_1. We will do so by constructing a relation between states $s \in \mathcal{S}_{\mathcal{L}^{r_0}}$, $t \in \mathcal{S}_{\mathcal{L}^{r_1}}$ that expresses the potential to match

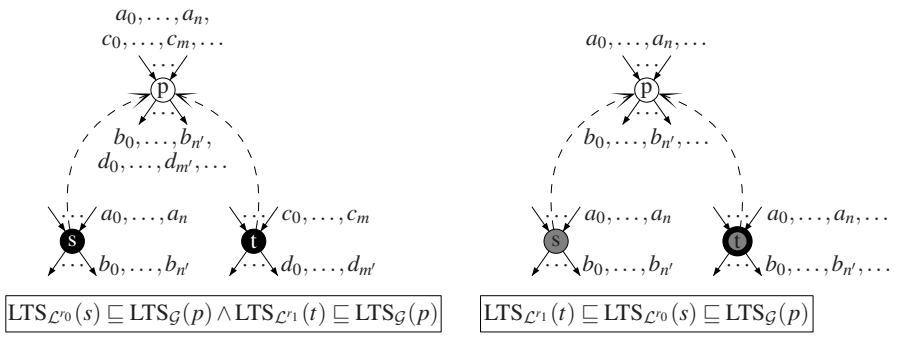

Figure 5: Matching rule left pattern states on the same LTS states

them on the same state p in an arbitrary LTS. If a non-empty relation can be constructed, then a conflict situation can be derived from it.

Two states $s \in \mathcal{S}_{\mathcal{L}^{r_0}}$, $t \in \mathcal{S}_{\mathcal{L}^{r_1}}$ can only be matched on the same state p if their in- and outgoing transitions are in some sense compatible. Since $\mathcal{L}^{r_0}$ and $\mathcal{L}^{r_1}$ are weakly connected, we know that s and t must have in- and/or outgoing transitions. To reason about these, we define the notion of a *context LTS* of a state.

Definition 8 (Context LTS) *Given an LTS $\mathcal{G}$, we say that for a state $p \in \mathcal{S}_{\mathcal{G}}$, the* context LTS *$LTS_{\mathcal{G}}(p) = \langle \mathcal{S}_p, \mathcal{A}_p, \mathcal{T}_p \rangle$ is defined as follows:*

- $\mathcal{S}_p = \{p' \mid \exists a \in \mathcal{A}_{\mathcal{G}}.p \xrightarrow{a}_{\mathcal{G}} p' \vee p' \xrightarrow{a}_{\mathcal{G}} p\} \cup \{p\}$;
- $\mathcal{A}_p = \{a \mid \exists p' \in \mathcal{S}_{\mathcal{G}}.p \xrightarrow{a}_{\mathcal{G}} p' \vee p' \xrightarrow{a}_{\mathcal{G}} p\}$;
- $\mathcal{T}_p = \{\langle p, a, p' \rangle \mid p \xrightarrow{a}_{\mathcal{G}} p'\} \cup \{\langle p', a, p \rangle \mid p' \xrightarrow{a}_{\mathcal{G}} p\}$.

Figure 5 shows the conditions under which we are able to match two left pattern states on the same state. On the left, the case where both states are glue is covered. In the figure, glue-states are coloured black. The figure expresses that any two glue-states s and t can be matched on a state p as long as p has matchable transitions for all the transitions of s and t. Say that state s has $n+1$ incoming transitions with labels $a_0,\ldots,a_n$, and $n'+1$ outgoing transitions with labels $b_0,\ldots,b_{n'}$, and state t has $m+1$ incoming transitions with labels $c_0,\ldots,c_m$, and $m'+1$ outgoing transitions with labels $d_0,\ldots,d_{m'}$, then p should have matchable transitions with all those labels. Of course, some transitions of s and t may be matched on common transitions of p, i.e. the two matches together could be non-injective.

Below the figure on the left, this condition is formalised as follows: we must have that $\mathrm{LTS}_{\mathcal{L}^{r_0}}(s) \sqsubseteq \mathrm{LTS}_{\mathcal{G}}(p) \wedge \mathrm{LTS}_{\mathcal{L}^{r_1}}(t) \sqsubseteq \mathrm{LTS}_{\mathcal{G}}(p)$. This directly follows from the fact that matches are injective LTS morphisms (Def. 4).

On the right in Figure 5, the condition for the possibility to match two states on the same state is given for the case that at least one of those states is non-glue. In the figure, non-glue states are coloured grey, and state t may be either non-glue or glue. The condition expresses that for all incoming and outgoing transitions of t, s must have corresponding transitions with the same label. This is formalised as $\mathrm{LTS}_{\mathcal{L}^{r_1}}(t) \sqsubseteq \mathrm{LTS}_{\mathcal{L}^{r_0}}(s) \sqsubseteq \mathrm{LTS}_{\mathcal{G}}(p)$. The idea is that if t has transitions that s does not have, then matching s and t on the same state p would not be possible due to the gluing conditions. Again, the fact that $\mathrm{LTS}_{\mathcal{L}^{r_0}}(s) \sqsubseteq \mathrm{LTS}_{\mathcal{G}}(p)$ and $\mathrm{LTS}_{\mathcal{L}^{r_1}}(t) \sqsubseteq \mathrm{LTS}_{\mathcal{G}}(p)$ should hold follows from the fact that matches are injective LTS morphisms. That $\mathrm{LTS}_{\mathcal{L}^{r_1}}(t) \sqsubseteq \mathrm{LTS}_{\mathcal{L}^{r_0}}(s)$ needs to hold follows from the following lemma.

Lemma 3 *Let $s \in \mathcal{S}_{\mathcal{L}^{r_0} \setminus \mathcal{K}^{r_0}}$, $t \in \mathcal{S}_{\mathcal{L}^{r_1}}$ be states. Then there can be no matches $m_0 : \mathcal{L}^{r_0} \to \mathcal{G}$, $m_1 : \mathcal{L}^{r_1} \to \mathcal{G}$ on an arbitrary LTS $\mathcal{G}$ with $m_{0,\mathcal{S}}(s) = p$ and $m_{1,\mathcal{S}}(t) = p$ for some $p \in \mathcal{S}_{\mathcal{G}}$ if $LTS_{\mathcal{L}^{r_1}}(t) \not\sqsubseteq LTS_{\mathcal{L}^{r_0}}(s)$.*

Proof. By reasoning towards a contradiction. Assume that $\mathrm{LTS}_{\mathcal{L}^{r_1}}(t) \not\sqsubseteq \mathrm{LTS}_{\mathcal{L}^{r_0}}(s)$ holds, and that we have matches $m_0 : \mathcal{L}^{r_0} \to \mathcal{G}$, $m_1 : \mathcal{L}^{r_1} \to \mathcal{G}$ with $m_{0,\mathcal{S}}(s) = p$ and $m_{1,\mathcal{S}}(t) = p$. Since $\mathrm{LTS}_{\mathcal{L}^{r_1}}(t) \not\sqsubseteq \mathrm{LTS}_{\mathcal{L}^{r_0}}(s)$ and $\mathcal{L}^{r_0}$ and $\mathcal{L}^{r_1}$ are weakly connected, we must have that at least one transition in $\mathrm{LTS}_{\mathcal{L}^{r_1}}(t)$ cannot be mapped on a transition in $\mathrm{LTS}_{\mathcal{L}^{r_0}}(s)$. Say that this is an incoming transition of t. The case that it is an outgoing transition of t is similar. We refer to this transition as $t' \xrightarrow{a}_{\mathcal{L}^{r_1}} t$. Since, by Def. 8, $\mathrm{LTS}_{\mathcal{L}^{r_1}}(t) \subseteq \mathcal{L}^{r_1}$, and since m_1 is an injective LTS morphism, $m_{1,\mathcal{S}}(t')$ and $m_{1,\mathcal{T}}(t' \xrightarrow{a}_{\mathcal{L}^{r_1}} t)$ must be defined. Let us say that $m_{1,\mathcal{S}}(t') = p'$, and $m_{1,\mathcal{T}}(t' \xrightarrow{a}_{\mathcal{L}^{r_1}} t) = p' \xrightarrow{a}_{\mathcal{G}} p$. By the fact that $\mathrm{LTS}_{\mathcal{L}^{r_0}}(s)$ contains all the incoming and outgoing transitions of s (Def. 8), and by the facts that $t' \xrightarrow{a}_{\mathcal{L}^{r_1}} t$ cannot be mapped on a transition in $\mathrm{LTS}_{\mathcal{L}^{r_0}}(s)$ and $\mathrm{LTS}_{\mathcal{L}^{r_0}}(s) \subseteq \mathcal{L}^{r_0}$, it follows that p' and $p' \xrightarrow{a}_{\mathcal{G}} p$ are not matched by the LTS morphism m_0. But, since $s \in \mathcal{S}_{\mathcal{L}^{r_0} \setminus \mathcal{K}^{r_0}}$ and $p' \xrightarrow{a}_{\mathcal{G}} m_{0,\mathcal{S}}(s)$, by Def. 4, we must have that there exists an $s' \in \mathcal{S}_{\mathcal{L}^{r_0}}$ such that $m_{0,\mathcal{S}}(s') = p'$, and we have a contradiction. □

Note that the condition on the right in Figure 5 implies what needs to hold if both s and t are non-glue. If s is non-glue, we must have that $\mathrm{LTS}_{\mathcal{L}^{r_1}}(t) \sqsubseteq \mathrm{LTS}_{\mathcal{L}^{r_0}}(s) \sqsubseteq \mathrm{LTS}_{\mathcal{G}}(p)$, but if t is non-glue, we must have $\mathrm{LTS}_{\mathcal{L}^{r_0}}(s) \sqsubseteq \mathrm{LTS}_{\mathcal{L}^{r_1}}(t) \sqsubseteq \mathrm{LTS}_{\mathcal{G}}(p)$, i.e. we must have that $\mathrm{LTS}_{\mathcal{L}^{r_0}}(s) \simeq \mathrm{LTS}_{\mathcal{L}^{r_1}}(t)$.

Figure 5 gives rise to defining a relation between left patterns of rules, where states s and t are related iff it is conceivable to construct a situation (in the form of an LTS) in which matches m_0 and m_1 relate s and t to a common state p, and likewise for transitions. Having such a relation, it follows from Lemma 1 that if it at least relates two transitions of which at least one is not represented in the interface of the corresponding rule, then the inferred situation is a conflict situation, i.e. there are direct transformations that are in conflict. We will use such a relation later on to reason about all possible conflicts involving two given rules, by iterating over all pairs of states from their left patterns.

Next, we define this relation between left patterns, and after that, we explain how a conflict situation can be constructed from a concrete relation.

Definition 9 (Conflict Compatibility Morphism) *Let $s \in \mathcal{S}_{\mathcal{L}^{r_0}}$, $t \in \mathcal{S}_{\mathcal{L}^{r_1}}$. A partial LTS morphism $f : \mathcal{L}^{r_0} \to \mathcal{L}^{r_1}$ is a* conflict compatibility morphism *if it is injective and $f_{\mathcal{S}}(s) = t$ implies that*

- *If $s \in \mathcal{S}_{\mathcal{L}^{r_0} \setminus \mathcal{K}^{r_0}}$ and $t \in \mathcal{S}_{\mathcal{L}^{r_1}}$ then*
  - *if $t \xrightarrow{a}_{\mathcal{L}^{r_1}} t'$ then $s \xrightarrow{a}_{\mathcal{L}^{r_0}} s'$ with $f_{\mathcal{S}}(s') = t'$, $f_{\mathcal{T}}(s \xrightarrow{a}_{\mathcal{L}^{r_0}} s') = t \xrightarrow{a}_{\mathcal{L}^{r_1}} t'$;*
  - *if $t \xleftarrow{a}_{\mathcal{L}^{r_1}} t'$ then $s \xleftarrow{a}_{\mathcal{L}^{r_0}} s'$ with $f_{\mathcal{S}}(s') = t'$, $f_{\mathcal{T}}(s \xleftarrow{a}_{\mathcal{L}^{r_0}} s') = t \xleftarrow{a}_{\mathcal{L}^{r_1}} t'$.*
- *If $s \in \mathcal{S}_{\mathcal{L}^{r_0}}$ and $t \in \mathcal{S}_{\mathcal{L}^{r_1} \setminus \mathcal{K}^{r_1}}$ then*
  - *if $s \xrightarrow{a}_{\mathcal{L}^{r_0}} s'$ then $t \xrightarrow{a}_{\mathcal{L}^{r_1}} t'$ with $f_{\mathcal{S}}(s') = t'$, $f_{\mathcal{T}}(s \xrightarrow{a}_{\mathcal{L}^{r_0}} s') = t \xrightarrow{a}_{\mathcal{L}^{r_1}} t'$;*
  - *if $s \xleftarrow{a}_{\mathcal{L}^{r_0}} s'$ then $t \xleftarrow{a}_{\mathcal{L}^{r_1}} t'$ with $f_{\mathcal{S}}(s') = t'$, $f_{\mathcal{T}}(s \xleftarrow{a}_{\mathcal{L}^{r_0}} s') = t \xleftarrow{a}_{\mathcal{L}^{r_1}} t'$.*
- *If $s \in \mathcal{S}_{\mathcal{L}^{r_0} \setminus \mathcal{K}^{r_0}}$ and $t \in \mathcal{S}_{\mathcal{L}^{r_1} \setminus \mathcal{K}^{r_1}}$ then*
  - *if $s \xrightarrow{a}_{\mathcal{L}^{r_0}} s'$ then $t \xrightarrow{a}_{\mathcal{L}^{r_1}} t'$ with $f_{\mathcal{S}}(s') = t'$, $f_{\mathcal{T}}(s \xrightarrow{a}_{\mathcal{L}^{r_0}} s') = t \xrightarrow{a}_{\mathcal{L}^{r_1}} t'$;*
  - *if $s \xleftarrow{a}_{\mathcal{L}^{r_0}} s'$ and $t \xleftarrow{a}_{\mathcal{L}^{r_1}} t'$, then $f_{\mathcal{S}}(s') = t'$, $f_{\mathcal{T}}(s \xleftarrow{a}_{\mathcal{L}^{r_0}} s') = t \xleftarrow{a}_{\mathcal{L}^{r_1}} t'$.*
  - *if $t \xrightarrow{a}_{\mathcal{L}^{r_1}} t'$ then $s \xrightarrow{a}_{\mathcal{L}^{r_0}} s'$ with $f_{\mathcal{S}}(s') = t'$, $f_{\mathcal{T}}(s \xrightarrow{a}_{\mathcal{L}^{r_0}} s') = t \xrightarrow{a}_{\mathcal{L}^{r_1}} t'$;*
  - *if $s \xleftarrow{a}_{\mathcal{L}^{r_0}} s'$ then $t \xleftarrow{a}_{\mathcal{L}^{r_1}} t'$ with $f_{\mathcal{S}}(s') = t'$, $f_{\mathcal{T}}(s \xleftarrow{a}_{\mathcal{L}^{r_0}} s') = t \xleftarrow{a}_{\mathcal{L}^{r_1}} t'$.*

Note that Def. 9 does not state anything about the case that both s and t are glue-states. Unlike in the other cases, in which the gluing conditions are relevant because at least one non-glue state is involved, two glue-states can always be related to each other. This means that an LTS morphism which only relates glue-states and no transitions is also a conflict compatibility morphism. However, by Lemma 1, such a morphism does not directly represent a conflict situation. We are not interested in just any conflict compatibility morphism, but those for which $f_{\mathcal{T}}$ is defined for some transitions. In fact, given two left patterns $\mathcal{L}^{r_0}$, $\mathcal{L}^{r_1}$ and two states $s \in \mathcal{S}_{\mathcal{L}^{r_0}}$, $t \in \mathcal{S}_{\mathcal{L}^{r_1}}$, we are interested in the *largest* conflict compatibility

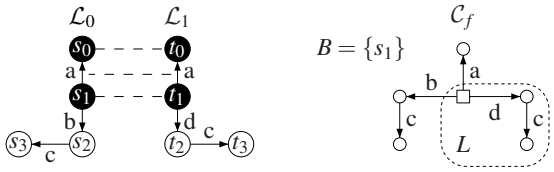

Figure 6: A conflict compatibility morphism between left patterns $\mathcal{L}_0$, $\mathcal{L}_1$, and the corresponding conflict situation $\mathcal{C}_f$

morphism f for which $f_{\mathcal{S}}(s) = t$, and its *domain of definition*, i.e. the part of $\mathcal{L}^{r_0}$ for which f is defined, is a *weakly connected* LTS.

For example, consider the two LTSs $\mathcal{L}_0$ and $\mathcal{L}_1$ on the left in Figure 6. A conflict compatibility morphism f with $f_{\mathcal{S}}(s_1) = t_1$ could be defined without relating any other states and transitions, but it would not represent a conflict. Instead, the largest possible morphism f with $f_{\mathcal{S}}(s_1) = t_1$ and a weakly connected domain of definition also relates s_0 with t_0, and the a-transitions. In particular, $s_2 \xrightarrow{c} s_3$ and $t_2 \xrightarrow{c} t_3$ are not related by f, since that would make its domain of definition not weakly connected.

Note that for two states s, t, there can be more than one conflict compatibility morphism of interest, particularly if there are multiple options to relate states and transitions.

In the remainder of this paper, each conflict compatibility morphism is the largest possible with a weakly connected domain of definition, in the sense that no morphism can be constructed that contains it and also has a weakly connected domain of definition.

Given $s \in \mathcal{S}_{\mathcal{L}^{r_0}}$ and $t \in \mathcal{S}_{\mathcal{L}^{r_1}}$, we can now construct conflict compatibility morphisms f. From f, $\mathcal{L}^{r_0}$, and $\mathcal{L}^{r_1}$, we can construct a conflict situation. For this, we use m_0, m_1 to map $\mathcal{L}^{r_0}$ and $\mathcal{L}^{r_1}$ to isomorphic LTS structures.

Definition 10 (Conflict Situation) *Let $f : \mathcal{L}^{r_0} \to \mathcal{L}^{r_1}$ be a conflict compatibility morphism, and $m_0(\mathcal{L}^{r_0})$, $m_1(\mathcal{L}^{r_1})$ LTSs isomorphic to $\mathcal{L}^{r_0}$ and $\mathcal{L}^{r_1}$, respectively. Then, a* conflict situation *LTS $\mathcal{C}_f$ can be constructed as follows: first, determine the boundary B of f consisting of all states $s \in \mathcal{L}^{r_0}$ such that $f_{\mathcal{S}}(s)$ is defined, but for some transition $s \xrightarrow{a}_{\mathcal{L}^{r_0}} s'$ (or $s \xleftarrow{a}_{\mathcal{L}^{r_0}} s'$), $f_{\mathcal{T}}(s \xrightarrow{a}_{\mathcal{L}^{r_0}} s')$ (or $f_{\mathcal{T}}(s \xleftarrow{a}_{\mathcal{L}^{r_0}} s')$) is not. Then, $L = m_1(\mathcal{L}^{r_1} \setminus f(\mathcal{L}^{r_0})) \cup m_1(f(B))$ can be glued to $m_0(\mathcal{L}^{r_0})$ by merging each state $s \in m_0(B)$ with the corresponding state $s' \in m_1(f(B))$. The result is $\mathcal{C}_f$.*

On the right of Figure 6, the conflict situation is presented which results from applying Def. 10 on the conflict compatibility morphism between $\mathcal{L}_0$ and $\mathcal{L}_1$ given on the left in the figure. The boundary B is defined as $B = \{s_1\}$, since $f_{\mathcal{S}}(s_1) = t_1$, but $f_{\mathcal{T}}(s_1 \xrightarrow{b}_{\mathcal{L}_0} s_2) = \perp$. Then, L is the LTS isomorphic to $\mathcal{L}_1$ without t_0 and the a-transition between t_0 and t_1 (indicated in C_f in the figure), and L is glued to an LTS isomorphic to $\mathcal{L}_0$ by merging the states related to s_1 and t_1 (resulting in the square state in the figure).

4 Conflict Detection and Resolution Algorithms

The findings presented in Section 3 can be used to construct a new conflict detection algorithm. It is presented in Alg. 1. Given two rules, the algorithm tries to determine whether there can be an LTS for which it is possible to construct direct transformations that are in conflict. The decision procedures are sorted by their complexity. If full analysis is needed, i.e. all possible conflict compatibility morphisms have to be computed, then attempts can be restricted to those pairs of states $s \in \mathcal{S}_{\mathcal{L}^{r_0}}$, $t \in \mathcal{S}_{\mathcal{L}^{r_1}}$ that share an outgoing transition label, and for at least one of the two states, an outgoing transition with that label

Algorithm 1 Conflict detection algorithm

Require: Rules $r_0 = \langle \mathcal{L}^{r_0}, \mathcal{R}^{r_0} \rangle$, $r_1 = \langle \mathcal{L}^{r_1}, \mathcal{R}^{r_1} \rangle$
Ensure: Returns set of conflicts between r_0 and r_1
$C = \emptyset$
2: **if** $\mathcal{A}_{\mathcal{L}^{r_0} \setminus \mathcal{K}^{r_0}} \cap \mathcal{A}_{\mathcal{L}^{r_1}} = \emptyset$ **and** $\mathcal{A}_{\mathcal{L}^{r_1} \setminus \mathcal{K}^{r_1}} \cap \mathcal{A}_{\mathcal{L}^{r_0}} = \emptyset$ **then**
return $\emptyset$ // Lemma 2
4: **if** $\mathcal{L}^{r_0} \simeq \mathcal{K}^{r_0} \wedge \mathcal{L}^{r_1} \simeq \mathcal{K}^{r_1}$ **then**
return $\emptyset$ // See [12]
6: **for all** $s \in \mathcal{S}_{\mathcal{L}^{r_0}}, t \in \mathcal{S}_{\mathcal{L}^{r_1}}$ **do**
if $\mathcal{A}_{out}(s) \cap \overline{\mathcal{A}_{out}}(t) \neq \emptyset$ **or** $\overline{\mathcal{A}_{out}}(s) \cap \mathcal{A}_{out}(t) \neq \emptyset$ **then**
8: **for all** conflict compatibility morphisms $f : \mathcal{L}^{r_0} \to \mathcal{L}^{r_1}$ with $f_{\mathcal{S}}(s) = t$ **do**
if $f_{\mathcal{T}}$ is defined for at least one transition **then**
10: add $\mathcal{C}_f$ to C // Definition 9
return C

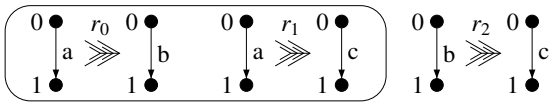

Figure 7: A Critical Pair may be resolvable

will be removed when transforming. This directly follows from Lemma 1, which implies that in order to have a conflict, at least one transition needs to be involved which is matched on by both rules r_0 and r_1 and removed by at least one of these rules. To formalise this, we use the following notation: $\mathcal{A}_{out}(s) = \{a \in \mathcal{A}_{\mathcal{L}^{r_0}} \mid \exists s \xrightarrow{a}_{\mathcal{L}^{r_0}} s'\}$ is the set of labels of outgoing transitions of s, and $\overline{\mathcal{A}_{out}}(s) = \{a \in \mathcal{A}_{\mathcal{L}^{r_0}} \mid \exists s \xrightarrow{a}_{\mathcal{L}^{r_0}} s'.f_{\mathcal{T}}^{-1}(s \xrightarrow{a}_{\mathcal{L}^{r_0}} s') = \perp\}$ is the set of labels of outgoing transitions that are set for removal. At line 2, the action sets are compared, which can be done in $\mathcal{O}(|\mathcal{A}|)$ time, with $\mathcal{A} = \mathcal{A}_{\mathcal{L}^{r_0}} \cup \mathcal{A}_{\mathcal{L}^{r_1}}$. At line 4, we use a check from [12], based on the fact that two non-deleting rules can never be in conflict. This can be determined in $\mathcal{O}(|\mathcal{S}| + |\mathcal{T}|)$ time, with $|\mathcal{S}|$ and $|\mathcal{T}|$ the total number of states and transitions in the two left rule patterns together. Next, full checking for conflict compatibility morphisms (lines 7-10) requires worst-case to compare the two left LTS patterns for all pairs of states, i.e. its complexity is $\mathcal{O}(|\mathcal{S}|^2 \cdot |\mathcal{T}| \cdot \log|\mathcal{S}|)$, since a comparison of two LTSs can be done in $\mathcal{O}(|\mathcal{T}| \cdot \log|\mathcal{S}|)$ time, using the equivalence checking algorithm of Paige & Tarjan [18].

Compared to earlier work, our detection algorithm has a number of advantages. First of all, comparison of the action sets can be done in linear time, and is, unlike other special case optimisations, such as those in [12], also applicable when both r_0 and r_1 remove some transitions. Second of all, not all possible pairs of states $s \in \mathcal{S}_{\mathcal{L}^{r_0}}$, $t \in \mathcal{S}_{\mathcal{L}^{r_1}}$ need to be considered in detail. Just by considering their outgoing transitions first can we quickly resolve many combinations in practice. This exploits the fact that LTSs are weakly connected, or more specifically, that most states have outgoing transitions.

It is a known fact in graph transformation that the existence of critical pairs does not guarantee that a transformation system is not confluent. For example, consider the system in Figure 7. Rules r_0 and r_1 are clearly in conflict, since they both concern an a-transition in their left pattern. They also define different transformation results, namely a b- and a c-transition, respectively, so direct transformations on an LTS $\mathcal{G}$ consisting of a transition $s \xrightarrow{a}_{\mathcal{G}} s'$ constitute a critical pair. However, consider that there is a third rule r_2 in the system, which transforms b-transitions into c-transitions. Then $\mathcal{G}$ can be transformed to an LTS consisting of a single c-transition either by first applying r_0 and then r_2, or by applying r_1. The conflict represented by the critical pair can be resolved.

Another example is the conflict situation in Figure 4. It cannot be resolved if the rule system only consists of rules r_0 and r_1. Applying first the direct transformation of r_1 removes the match of m_0 on $\mathcal{G}$,

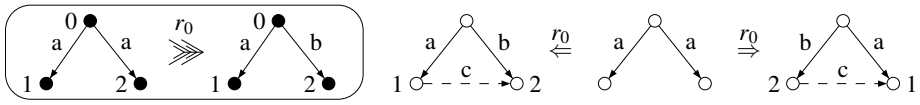

Figure 8: The need for strong joinability

and applying first the direct transformation of r_0, followed by the one of r_1 leads to a different LTS, in which instead of the b-transition there is now a c-transition.

Since the existence of critical pairs does not mean that a transformation system is not confluent, a necessary condition needs to be found for a critical pair to actually be an example why a system is not confluent. Plump [21] demonstrates that for so-called *coverable* transformation systems of hypergraphs, i.e. graphs where the edges can be associated with multiple source and target vertices, it suffices to show that all the critical pairs are *strongly joinable*, meaning that independent of which of the two involved direct transformations is applied on the conflict situation, the system can transform the resulting graph to a graph that is structurally equivalent to the graph that can be obtained if the other direct transformation had been applied first. Since LTSs are a special kind of hypergraph, and since our LTSs are always coverable (an important criterium is that LTSs can be extended with a cover consisting of transitions with fresh labels), we can directly take the result from [21] for our setting. To define strong joinability formally, we first need to define the notion of a *track morphism*, based on [20]. Such a morphism explicitly involves relations between states based on the fact that they have been matched by the same interface state.

Definition 11 (Track Morphism) *Given a direct transformation $\mathcal{G} \Rightarrow_{r,m} \mathcal{H}$, the* track morphism $tr_{\mathcal{G}\Rightarrow\mathcal{H}} : \mathcal{G} \to \mathcal{H}$ *is the partial LTS morphism defined by*

$$tr_{\mathcal{G}\Rightarrow\mathcal{H}}(s) = \begin{cases} m'_{\mathcal{S}}(g_{\mathcal{S}}(f_{\mathcal{S}}^{-1}(m_{\mathcal{S}}^{-1}(s)))) & \text{if } f_{\mathcal{S}}^{-1}(m_{\mathcal{S}}^{-1}(s)) \text{ is defined} \\ \bot & \text{otherwise} \end{cases}$$

The morphisms m, m', f and g are as shown in Figure 3. Track morphisms can be defined for sequences of direct transformations in a similar way, where for two direct transformations $\mathcal{G} \Rightarrow \mathcal{H}$ and $\mathcal{H} \Rightarrow \mathcal{H}'$, $tr_{\mathcal{G}\Rightarrow\mathcal{H}\Rightarrow\mathcal{H}'}(s)$ is defined as $tr_{\mathcal{H}\Rightarrow\mathcal{H}'} \circ tr_{\mathcal{G}\Rightarrow\mathcal{H}}(s)$.

Definition 12 (Strong Joinability) *Given a transformation system Σ, a critical pair $\mathcal{H}_0 \Leftarrow_{r_0,m_0} \mathcal{G} \Rightarrow_{r_1,m_1} \mathcal{H}_1$ is* strongly joinable *if there are derivations $\mathcal{H}_i \Rightarrow_{\Sigma}^{*} \mathcal{X}_i$, for $i = 0, 1$, an isomorphism $f : \mathcal{X}_0 \to \mathcal{X}_1$, and for each state $s \in \mathcal{S}_{\mathcal{G}}$, if both $tr_{\mathcal{G}\Rightarrow\mathcal{H}_0}(s)$ and $tr_{\mathcal{G}\Rightarrow\mathcal{H}_1}(s)$ are defined (that is, s is persisting), then*

1. *$tr_{\mathcal{G}\Rightarrow\mathcal{H}_0\Rightarrow_{\Sigma}^{*}\mathcal{X}_0}(s)$ and $tr_{\mathcal{G}\Rightarrow\mathcal{H}_1\Rightarrow_{\Sigma}^{*}\mathcal{X}_1}(s)$ are defined;*
2. *$f_{\mathcal{S}}(tr_{\mathcal{G}\Rightarrow\mathcal{H}_0\Rightarrow_{\Sigma}^{*}\mathcal{X}_0}(s)) = tr_{\mathcal{G}\Rightarrow\mathcal{H}_1\Rightarrow_{\Sigma}^{*}\mathcal{X}_1}(s)$.*

Def. 12 not only expresses that the LTSs $\mathcal{X}_0$ and $\mathcal{X}_1$ need to be isomorphic, but besides that, that the states that *persist* along direct transformations $\mathcal{G} \Rightarrow_{r_0,m_0} \mathcal{H}_0$, $\mathcal{G} \Rightarrow_{r_1,m_1} \mathcal{H}_1$, i.e. that are matched by glue-states of both r_0 and r_1, are in the end still present in both $\mathcal{X}_0$ and $\mathcal{X}_1$, and relatable to themselves. Consider the example in Figure 8. Rule r_0 can be applied in two ways on the given input. The results are isomorphic, but not in a bigger context (the dashed c-transition). To detect this, one should compare on which states the glue-states are matched.

Plump [21] gives some suggestions how a transformation system can be equipped with a cover to determine whether a critical pair is strongly joinable. Based on that, we use the following approach: for a given critical pair, we define copies of r_0 and r_1 which we call ${r_0}^{\kappa}$ and ${r_1}^{\kappa}$, respectively, and we

Algorithm 2 Conflict resolution algorithm

Require: Conflicts in set C
Ensure: Returns **false** iff there exists a conflict in C that cannot be resolved, **true** otherwise
for all $(\mathcal{H}_0 \Leftarrow_{r_0,m_0} \mathcal{C}_f \Rightarrow_{r_1,m_1} \mathcal{H}_1) \in C$ **do**
2: apply $\mathcal{C}_f \Rightarrow_{r_0{}^{\kappa},m_0} \mathcal{H}_0^{\kappa}$ and $\mathcal{C}_f \Rightarrow_{r_1{}^{\kappa},m_1} \mathcal{H}_1^{\kappa}$
compute $\mathcal{H}_0^{\kappa_s}$ and $\mathcal{H}_1^{\kappa_s}$
4: apply $\mathcal{H}_0^{\kappa_s} \Rrightarrow_{\Sigma}^{*} \mathcal{X}_0$ and $\mathcal{H}_1^{\kappa_s} \Rrightarrow_{\Sigma}^{*} \mathcal{X}_1$
if $\mathcal{X}_0 \not\cong \mathcal{X}_1$ **or transformation failed then**
6: **return false**
return true

extend both $\mathcal{R}^{r_0{}^{\kappa}}$ and $\mathcal{R}^{r_1{}^{\kappa}}$ such that for each state s in $\mathcal{S}_{\mathcal{K}^{r_0{}^{\kappa}}}$ (and likewise in $\mathcal{S}_{\mathcal{K}^{r_1{}^{\kappa}}}$), we add a self-loop transition with the fresh, unique label κ to $g_{\mathcal{S}}(s)$. These selfloops, when the left pattern of $r_0{}^{\kappa}$ or $r_1{}^{\kappa}$ is matched on part of the conflict situation, are therefore introduced when applying a direct transformation, and then serve the purpose of marking the states that have been matched on glue-states. In this way, we can obtain LTSs $\mathcal{H}_0^{\kappa}$ and $\mathcal{H}_1^{\kappa}$, i.e. the LTSs $\mathcal{H}_0$ and $\mathcal{H}_1$ extended with the κ-selfloops. Once we have these, we relabel the κ-selfloops in $\mathcal{H}_0^{\kappa}$ and $\mathcal{H}_1^{\kappa}$, such that each state s has a selfloop labelled κ_s, and after that, remove those κ_s-selfloops that do not appear in both LTSs, i.e. that are not associated with s in both $\mathcal{H}_0^{\kappa}$ and $\mathcal{H}_0^{\kappa}$. We call the resulting LTSs $\mathcal{H}_0^{\kappa_s}$ and $\mathcal{H}_1^{\kappa_s}$.

Subsequent matches for rules $r \in \Sigma$ can only be established if they do not match a non-glue state on a persisting state, since trying to do so would violate the gluing conditions w.r.t. the related κ_s-selfloop. If it is detected that such a violating 'match' can be made, then the critical pair is not strongly joinable, and the transformation fails. Besides that, each time a match has been established, we remove all κ_s-selfloops of states s that have not been matched by any state.

In this way, states that persist along a sequence of direct transformations will still have their κ_s-selfloop in both $\mathcal{X}_0$ and $\mathcal{X}_1$. Then, it suffices to check that $\mathcal{X}_0$ and $\mathcal{X}_1$ are isomorphic.

In Alg. 2, our conflict resolution algorithm is presented, which can be used to determine, based on the constructed critical pairs, whether a transformation system is locally confluent, by establishing that all critical pairs are strongly joinable. With $\Rrightarrow$, we refer to applying a direct transformation after the removal of κ_s-selfloops of all the states that are not matched by states of the related transformation rule. The complexity of Alg. 2 depends on the complexity of graph transformation, which is performed in lines 2 and 4, which in turn is dominated by the complexity of finding matches at line 4. In general, the graph matching problem [2] is NP-complete. However, it has been shown in [2] that if the graphs have a root, all states are reachable from that root, and each state has a bounded number b of outgoing transitions, then the complexity is independent of the size of the input graph, instead only depending on b and the number of transitions n in the left pattern of the transformation rule. The complexity is then $\mathcal{O}(\Sigma_{i=0}^{n} b^i)$. Since our LTSs are weakly connected, they meet these requirements. The other operations at lines 3 and 5 in Alg. 2 can be performed in $\mathcal{O}(|\mathcal{S}| + |\mathcal{T}|)$, since they require scanning all states and transitions in the LTSs once.

It has to be noted that if conflict detection is performed before resolution, all possible critical pairs need to be constructed. If instead, detection and resolution are mixed, i.e. each time a new critical pair is detected, it is immediately tested for resolvability, then non-confluent transformation systems can be identified as such as soon as a pair has been found that cannot be resolved. In practice, this means that the construction of all possible critical pairs can often be avoided.

Following, a proof sketch is given to show correctness of the technique. Consider a transformation system Σ that is not confluent. Therefore, there exists an LTS $\mathcal{G}$ such that there are two direct transformations $\mathcal{H}_0 \Leftarrow_{r_0,m_0} \mathcal{G} \Rightarrow_{r_1,m_1} \mathcal{H}_1$ that are not parallel independent. By Lemma 1, this means that there must

be at least one transition x in $\mathcal{G}$, say with label a, that is matched on by both m_0 and m_1, and at least one of the two rules removes x. Let s and t be the source states of the transitions x_s and x_t in $\mathcal{L}^{r_0}$ and $\mathcal{L}^{r_1}$ that match on x, respectively, and let r_0 define that x must be removed. In Alg. 1, since $a \in \mathcal{A}_{\mathcal{L}^{r_0} \setminus \mathcal{K}^{r_0}} \cap \mathcal{A}_{\mathcal{L}^{r_1}}$, line 3 is skipped, and since $\mathcal{L}^{r_0} \not\simeq \mathcal{K}^{r_0}$, line 5 is skipped. We have $a \in \overline{\mathcal{A}_{out}}(s) \cap \mathcal{A}_{out}(t)$, hence at line 8, conflict compatibility morphisms will be computed with $f_{\mathcal{S}}(s) = t$. Since we only consider the largest possible conflict compatibility morphisms with a weakly connected domain of definition, we must have that $f_{\mathcal{T}}(x_s) = x_t$. If not, then either the source or target states in $\mathcal{L}^{r_0}$ and $\mathcal{L}^{r_1}$ are not relatable via f, which would mean that there is a gluing condition violation (Defs. 4 and 9), but that would mean that there can be no overlap of matches of r_0 and r_1 that involves x_s and x_t. This would be in contradiction with the fact that there is a conflict between r_0 and r_1 involving x. By Def. 10, a conflict situation $\mathcal{C}_f = m_0(\mathcal{L}^{r_0}) \cup m_1(\mathcal{L}^{r_1})$ is constructed at line 10 in Alg. 1 with $m_{0,\mathcal{T}}(x_s)$ representing the overlap between x_s and x_t. The subsequent inability to resolve the conflict using Alg. 2 can be proven along the lines of the proof in [21].

The case that a given system is confluent can be proven as follows: in general, Alg. 1 will produce some (possibly zero) conflicts. These conflicts, though, will be resolvable. This can be proven along the lines of the proof in [21].

5 Conclusions

In this paper, we discussed how conflicts in LTS transformation systems can be efficiently detected and resolved. For the detection, we proposed a novel approach that tries to construct partial morphisms between the involved rule patterns. In particular cases, the absence of conflicts can be determined in linear time, for instance when one rule only removes transitions that another rule will never match on, because it does not refer to the particular transition label(s). This is a big improvement over previous approaches, like e.g. in [12], since it is also applicable for two deleting rules, i.e. rules that remove transitions. For the resolution of conflicts, we have proposed an algorithm inspired by [21], but taylored to our particular setting using LTSs. For future work, we will consider extensions, e.g. [8,11], to extend our framework in comparable ways. Finally, for formal verification purposes, a hierarchy of different forms of confluence (ranging from strong to weak) has been identified concerning the behaviour described by LTSs [15]. It would be interesting to see how these relate to confluence variants in the setting of graph and model transformation.

References

[1] M. Amrani, L. Lucio, G. Selim, B. Combemale, J. Dingel, H. Vangheluwe, Y. Le Traon & J.R. Cordy (2012): *A Tridimensional Approach for Studying the Formal Verification of Model Transformations*. In: *ICST'12*, pp. 921–928, doi:10.1109/ICST.2012.197.

[2] M. Dodds & D. Plump (2006): *Graph Transformation in Constant Time*. In: *ICGT'06*, *LNCS* 4178, Springer, pp. 367–382, doi:10.1007/11841883_26.

[3] H. Ehrig, K. Ehrig, J. de Lara, G. Taentzer, D. Varró & S. Varró-Gyapay (2005): *Termination Criteria for Model Transformation*. In: *FASE'05*, *LNCS* 3442, Springer, pp. 49–63, doi:10.1007/978-3-540-31984-9_5.

[4] H. Ehrig, K. Ehrig, U. Prange & G. Taentzer (2006): *Fundamentals of Algebraic Graph Transformation*. EATCS Monographs in Theoretical Computer Science, Springer, doi:10.1007/3-540-31188-2.

[5] L.J.P Engelen (2012): *From Napkin Sketches to Reliable Software*. Ph.D. thesis, Eindhoven University of Technology, doi:10.6100/IR740040.

[6] L.J.P. Engelen & A.J. Wijs (2012): *Incremental Formal Verification for Model Refining*. In: *MoDeVVa'12*, ACM Computer Society Press, pp. 29–34, doi:10.1145/2427376.2427382.

[7] L. Grunske, L. Geiger, A. Zündorf, N. Van Eetvelde, P. Van Gorp & D. Varró (2005): *Using Graph Transformation for Practical Model-Driven Software Engineering*. In: *Model-Driven Software Development*, Springer, pp. 91–118, doi:10.1007/3-540-28554-7_5.

[8] R. Heckel, J.M. Küster & G. Taentzer (2002): *Confluence of Typed Attributed Graph Transformation Systems*. In: *ICGT'02*, *LNCS* 2505, Springer, pp. 161–176, doi:10.1007/3-540-45832-8_14.

[9] G. Huet (1980): *Confluent Reductions: Abstract Properties and Applications to Term Rewriting Systems*. *J. ACM* 27(4), pp. 797–821, doi:10.1145/322217.322230.

[10] M. Hülsbusch, B. König, A. Rensink, M. Semenyak, C. Soltenborn & H. Wehrheim (2010): *Showing Full Semantics Preservation in Model Transformation - A Comparison of Techniques*. In: *IFM'10*, *LNCS* 6396, Springer, pp. 183–198, doi:10.1007/978-3-642-16265-7_14.

[11] L. Lambers, H. Ehrig & F. Orejas (2006): *Conflict Detection for Graph Transformation with Negative Application Conditions*. In: *ICGT'06*, *LNCS* 4178, Springer, pp. 61–76, doi:10.1007/11841883_6.

[12] L. Lambers, H. Ehrig & F. Orejas (2006): *Efficient Detection of Conflicts in Graph-Based Model Transformations*. In: *GraMoT'05*, ENTCS 152, pp. 97–109, doi:10.1016/j.entcs.2006.01.017.

[13] L. Lambers, H. Ehrig & F. Orejas (2008): *Efficient Conflict Detection in Graph Transformation Systems by Essential Critical Pairs*. In: *GT-VMT'06*, ENTCS 211, pp. 17–26, doi:10.1016/j.entcs.2008.04.026.

[14] F. Lang (2005): EXP.OPEN *2.0: A Flexible Tool Integrating Partial Order, Compositional, and On-the-Fly Verification Methods*. In: *IFM'05*, *LNCS* 3771, Springer, pp. 70–88, doi:10.1007/11589976_6.

[15] R. Mateescu & A.J. Wijs (2012): *Sequential and Distributed On-The-Fly Computation of Weak Tau-Confluence*. *Science of Computer Programming* 70(10,11), pp. 1075–1094, doi:10.1016/j.scico.2011.07.004.

[16] A. Narayanan & G. Karsai (2008): *Towards Verifying Model Transformations*. In: *GT-VMT'06*, ENTCS 211, pp. 191–200, doi:10.1016/j.entcs.2008.04.041.

[17] M.H.A. Newman (1942): *On Theories with a Combinatorial Definition of "Equivalence"*. *Annals of Mathematics* 43(2), pp. 223–243, doi:10.2307/1968867.

[18] R. Paige & R.E. Tarjan (1984): *A Linear Time Algorithm to Solve the Single Function Coarsest Partition Problem*. In: *ICALP*, *LNCS* 172, Springer, pp. 371–379, doi:10.1007/3-540-13345-3_33.

[19] D. Plump (1993): *Hypergraph rewriting: Critical pairs and undecidability of confluence*. In R. Sleep, R. Plasmeijer & M. van Eekelen, editors: *Term Graph Rewriting: Theory and Practice*, chapter 15, John Wiley, pp. 201–213.

[20] D. Plump (2005): *Confluence of Graph Transformation Revisited*. In: *Processes, Terms and Cycles: Steps on the Road to Infinity*, *LNCS* 3838, Springer, pp. 280–308, doi:10.1007/11601548_16.

[21] D. Plump (2010): *Checking Graph Transformation Systems for Confluence*. In: *Essays Dedicated to Hans-Jörg Kreowski*, ECEASST 26, EASST.

[22] L.A. Rahim & J. Whittle (2013): *A Survey of Approaches for Verifying Model Transformations*. *Software and Systems Modeling*, doi:10.1007/s10270-013-0358-0.

[23] G. Taentzer, K. Ehrig, E. Guerra, J. de Lara, L. Lengyel, T. Levendovszky, U. Prange, D. Varró & S. Varró-Gyapay (2006): *Model Transformation by Graph Transformation: A Comparative Study*. In: *MTIP'05*, pp. 71–80.

[24] A.J. Wijs (2013): *Define, Verify, Refine: Correct Composition and Transformation of Concurrent System Semantics*. In: *FACS'13*, *LNCS* 8348, Springer, pp. 348–368, doi:10.1007/978-3-319-07602-7_21.

[25] A.J. Wijs & L.J.P. Engelen (2013): *Efficient Property Preservation Checking of Model Refinements*. In: *TACAS'13*, *LNCS* 7795, Springer, pp. 565–579, doi:10.1007/978-3-642-36742-7_41.

[26] A.J. Wijs & L.J.P. Engelen (2014): *REFINER: Towards Formal Verification of Model Transformations*. In: *NFM'14*, *LNCS* 8430, Springer, pp. 258–263, doi:10.1007/978-3-319-06200-6_21.