On the Essence of Parallel Independence for the Double-Pushout and Sesqui-Pushout Approaches

Andrea Corradini[1(✉)], Dominique Duval[2], Michael Löwe[3], Leila Ribeiro[4], Rodrigo Machado[4(✉)], Andrei Costa[4], Guilherme Grochau Azzi[4], Jonas Santos Bezerra[4], and Leonardo Marques Rodrigues[4]

[1] Università di Pisa, Pisa, Italy
andrea@di.unipi.it
[2] Université Grenoble-Alpe, Grenoble, France
dominique.duval@imag.fr
[3] Fachhochschule für die Wirtschaft Hannover, Hannover, Germany
michael.loewe@fhdw.de
[4] Universidade Federal do Rio Grande do Sul, Porto Alegre, Brazil
{leila,rma,acosta,ggazzi,jsbezerra,lmrodrigues}@inf.ufrgs.br

Abstract. Parallel independence between transformation steps is a basic notion in the algebraic approaches to graph transformation, which is at the core of some static analysis techniques like Critical Pair Analysis. We propose a new categorical condition of parallel independence and show its equivalence with two other conditions proposed in the literature, for both left-linear and non-left-linear rules. Next we present some preliminary experimental results aimed at comparing the three conditions with respect to computational efficiency. To this aim, we implemented the three conditions, for left-linear rules only, in the Verigraph system, and used them to check parallel independence of pairs of overlapping redexes generated from some sample graph transformation systems over categories of typed graphs.

1 Introduction

Graph transformation is a well-developed computational model suited to describe the evolution of distributed systems. System states are represented by graphs, and rules typically describe local changes of some part of the state. One central topic in the theory of graph transformation, first addressed in [21,12], has been the identification of conditions that guarantee that two transformation steps from a given state are independent, and thus can be applied in any order generating the same result. This is known as the *Local Church-Rosser Problem*, that is presented in the following way in [8]:

> Find a condition, called *parallel independence*, such that two alternative transformation steps $H_1 \overset{\rho_1}{\Longleftarrow} G \overset{\rho_2}{\Longrightarrow} H_2$ are parallel independent if and only if there are transformation steps $H_1 \overset{\rho_2}{\Longrightarrow} X$ and $H_2 \overset{\rho_1}{\Longrightarrow} X$ such that $G \overset{\rho_1}{\Longrightarrow} H_1 \overset{\rho_2}{\Longrightarrow} X$ and $G \overset{\rho_2}{\Longrightarrow} H_2 \overset{\rho_1}{\Longrightarrow} X$ are equivalent.

This final publication is avaliable at Springer via https://doi.org/10.1007/978-3-319-75396-6_1

The "equivalence" just mentioned informally means that the rules ρ_1 and ρ_2 consume the same items of G in the two transformation sequences. A formal definition, based on the classical *shift equivalence*, can be found in Section 3.5 of [8]. The above statement fixes a standard pattern for addressing the Local Church-Rosser Problem in the various approaches to algebraic graph transformation: first, a definition of *parallel independence* for transformation steps has to be provided, next a *Local Church-Rosser Theorem* proves that given two parallel independent transformation steps from a given graph, they can be applied in both orders obtaining the same result (up to isomorphism).

The efficient verification of parallel independence is important for the analysis of graph transformation systems. It is needed for example in *Critical Pair Analysis*, a static analysis technique originally introduced in term rewriting systems [15] and, starting with [20], widely used also in graph transformation and supported by some tools [22,9]. It relies on the generation of all possible *critical pairs*, i.e. pairs of transformation steps in minimal context that are not parallel independent, which can be used to prove local confluence or to provide the modeler with all possible conflicts between transformation rules. Efficient parallel independence verification could also be exploited by partial-order reduction techniques in tools supporting model checking of graph transformation systems, like GROOVE [14].

In the first part of the paper we discuss three definitions of parallel independence proposed for the classical Double-Pushout Approach (DPO) [11], and also applicable to the richer setting of the Sesqui-Pushout Approach (SQPO) [7], which extends DPO by allowing also the specification of cloning or copying of items. The third of such definitions is new, and we claim that it captures the essence of parallel independence, being simpler than the other ones. We exploit the third condition as a pivot in proving that all presented conditions are equivalent.

In the second part of the paper we report on some experimental evaluations of the complexity of verifying parallel independence according with the three conditions. They have been implemented in Verigraph, a framework for the specification and verification of graph transformation systems written in Haskell, under development at the Universidade Federal do Rio Grande do Sul [9]. Since the current version of Verigraph does not support Sesqui-Pushout transformation, only left-linear rules are considered in the evaluation. After describing the basic data structures used in Verigraph to model categorical constructions, we discuss the implementation of the three equivalent conditions and compare the time efficiency of verifying them on a collection of test cases. The newly proposed condition turns out to be the in most cases the most efficient.

The reader is assumed to be familiar with the DPO approach and with typed graphs [8]. Some background notions are introduced in Section 2. In Section 3 we introduce the three conditions for parallel independence, and Section 4 is devoted to the proof of their equivalence. The Verigraph system is presented in Section 5, which also describes how the parallel independence conditions were implemented. Experimental results are described and analysed in Section 6. Finally, Section 7 presents concluding remarks.

2 Background

In order to fix the terminology, let us recall the standard definition of Double-Pushout [11] and Sesqui-Pushout transformation [7] in a generic category $\mathbf{C}$. Conditions on $\mathbf{C}$ will be introduced when needed.

Definition 1 (Double-Pushout transformation). *A* rule $\rho = (L \stackrel{l}{\leftarrow} K \stackrel{r}{\rightarrow} R)$ *is a span of arrows in* $\mathbf{C}$*. Rule* ρ *is* left-linear *if* l *is mono,* right-linear *if* r *is mono, and* linear *if both* l *and* r *are monos. A* match *for rule* ρ *in an object* G *is an arrow* $m : L \to G$*. If the diagram in (1) exists in* $\mathbf{C}$*, where both squares are pushouts, then we say that there is a* DPO transformation step *from* G *to* H *via* (ρ, m)*, and we write* $G \stackrel{\rho,m}{\Longrightarrow} H$*. In this case we call the pair* (ρ, m) *a* redex *in* G*, and* $K \stackrel{n}{\rightarrow} D \stackrel{g}{\rightarrow} G$ *a* pushout complement (POC) *of* $K \stackrel{l}{\rightarrow} L \stackrel{m}{\rightarrow} G$*. We write* $G \stackrel{\rho}{\Longrightarrow} H$ *if* $G \stackrel{\rho,m}{\Longrightarrow} H$ *for some match* m *for* ρ *in* G*.*

$$\begin{array}{ccccccc} L & \longleftarrow{\scriptstyle l} & K & \longrightarrow{\scriptstyle r} & R \\ \downarrow{\scriptstyle m} & & \downarrow{\scriptstyle n} & & \downarrow{\scriptstyle q} \\ G & \longleftarrow{\scriptstyle g} & D & \longrightarrow{\scriptstyle h} & H \end{array} \qquad (1)$$

Therefore if (ρ, m) is a redex in G we know that ρ can be applied to match m in G. In Diagram (1), K is called the *interface* and D the *context*.

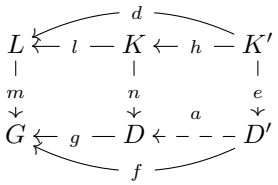
In most presentations of the DPO approach, suitable conditions are imposed to guarantee a form of determinism, i.e. that if $G \stackrel{\rho,m}{\Longrightarrow} H$ and $G \stackrel{\rho,m}{\Longrightarrow} H'$ then H and H' are isomorphic. This is often achieved by requiring rules to be left-linear, because in several categories of interest (as in *adhesive categories* [16]) if l is mono then the pushout complement of $K \stackrel{l}{\rightarrow} L \stackrel{m}{\rightarrow} G$ is uniquely determined (up to isomorphism) if it exists, and thus also object H is unique up to iso by the universal property of the right-hand side pushout.[5]

Sesqui-Pushout transformations were proposed in [7] as a little variation of DPO ones, able to guarantee the above form of determinism by the very definition. The only difference with respect to DPO is the property that the left square of (1) has to satisfy. We first recall the definition of final pullback complement.

Definition 2 (final pullback complement). *In Diagram (2),* $K \stackrel{n}{\rightarrow} D \stackrel{g}{\rightarrow} G$ *is a* final pullback complement (FPBC) *of* $K \stackrel{l}{\rightarrow} L \stackrel{m}{\rightarrow} G$ *if*

[5] In non-adhesive categories stronger conditions might be necessary. For example, in the category of term graphs (which is not adhesive but just *quasi-adhesive* [6]), two non-isomorphic pushout complements may exist for a monic l, while uniqueness is ensured by requiring l to be a *regular mono*, i.e. the equalizer of a pair of parallel arrows. It is worth recalling here that every adhesive category is quasi-adhesive, and that in an adhesive category all monos are regular [16].

1. *the resulting square is a pullback, and*
2. *for each pullback* $G \overset{m}{\leftarrow} L \overset{d}{\leftarrow} K' \overset{e}{\rightarrow} D' \overset{f}{\rightarrow} G$ *and arrow* $K' \overset{h}{\rightarrow} K$ *such that* $l \circ h = d$*, there is a unique arrow* $D' \overset{a}{\rightarrow} D$ *such that* $g \circ a = f$ *and* $a \circ e = n \circ h$*.*



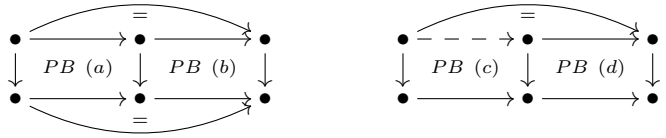
(2)

Definition 3 (Sesqui-Pushout transformation). *Under the premises of Definition 1, we say that there is a* SQPO transformation step *from* G *to* H *via* (ρ, m) *if the diagram in (1) can be constructed in* $\mathbf{C}$*, where the left square is a final pullback complement of* $K \overset{l}{\rightarrow} L \overset{m}{\rightarrow} G$ *and the right square is a pushout.*

The final pullback complement of two arrows is characterised by a universal property, and thus it is unique up to isomorphism, if it exists. Therefore SQPO transformation is deterministic in the above sense also for non-left-linear rules. Furthermore, as discussed in [7], in an adhesive category every DPO transformation for a left-linear rule is also an SQPO transformation, and thus SQPO rewriting can be considered as a conservative extension of DPO transformation.

Along the paper we shall often use some well-known properties of pullbacks:

Fact 1 (composition and decomposition of pullbacks) *1. In the diagram on the left if squares* (a) *and* (b) *are pullbacks, so is the composed square:*



2. *In the diagram made of solid arrows above on the right, if square* (d) *and the outer square are pullbacks, then there is a unique arrow (the dotted one) such that the top triangle commutes and square* (c) *is a pullback.*

The following definition will be useful in the following.

Definition 4 (reflection). *Given objects* X*,* Y*,* Z *and arrows* $f : X \to Z$*,* $g : Y \to Z$ *we say that* f is reflected along g *if the pullback object of* $Y \overset{g}{\rightarrow} Z \overset{f}{\leftarrow} X$ *is isomorphic to* X*, as in square* ① *or, equivalently, if there is an arrow* $h : X \to Y$ *such that* $Y \overset{h}{\leftarrow} X \overset{id}{\rightarrow} X$ *is the pullback of* $Y \overset{g}{\rightarrow} Z \overset{f}{\leftarrow} X$*, as in square* ②*.*

Note that such an arrow $h : X \to Y$*, if it exists, is necessarily unique. In this case we also say that* f is reflected along g by h*.*

Intuitively, this means that g is an isomorphism when restricted to the image of f. If objects are concrete structures like graphs, then every item of the image of f in Z has exactly one inverse image along g in Y. The following facts are easy to check by properties of pullbacks.

Fact 2 (some properties of reflection) *1. Arrow $f : X \to Z$ is reflected along $g : Y \to Z$ iff there exists an arrow $h : X \to Y$ such that for all pairs of arrows $m : W \to Y$ and $n : W \to X$, if $g \circ m = f \circ n$ then $h \circ n = m$.*
2. If g is mono, then f is reflected along g iff there exists an arrow $h : X \to Y$ such that $f = g \circ h$.

We will use as running example the following SQPO graph grammar, based on the category of graphs typed over the left graph of Fig. 1.

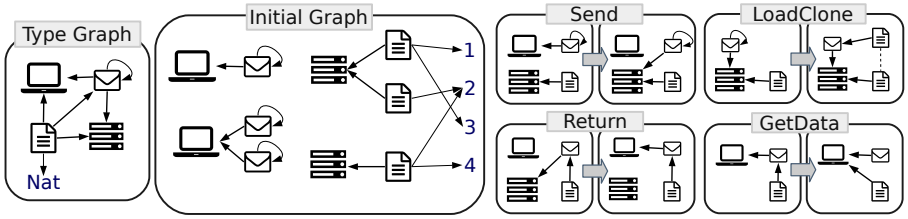
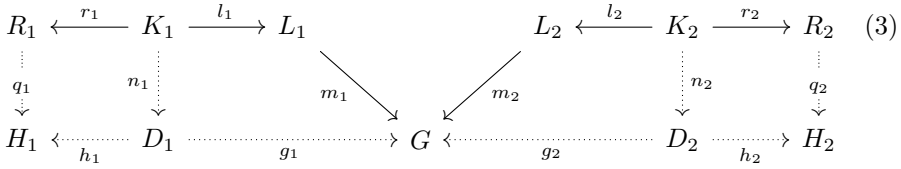


Fig. 1. Graph grammar for clients and servers

Example 1 (SQPO graph grammar for clients and servers). The graph grammar of Fig. 1 represents clients (computers) obtaining data (documents) from servers via messages (envelopes). In the model, data nodes represent subsets of $\{1, 2, 3, 4\}$ (elements of type Nat), which are initially stored in servers. Arrows represent locations and references. Loops in messages are used to ensure that only one data node is loaded at each time. Clients transmit messages to servers via rule Send. Messages are loaded with a cloned version of a data node via rule LoadClone, with a dotted line being used to represent the cloning effect. It is worth recalling that all edges from the cloned data node to numbers are also cloned as a side-effect of SQPO rewriting. Rule Return transmits messages with data back to clients. Finally, rule GetData deletes a message, placing the data node directly into the client node. Rules are presented by showing only the left- and right-hand sides: the interface is their intersection for all rules but LoadClone, which is shown in detail in Fig. 2.

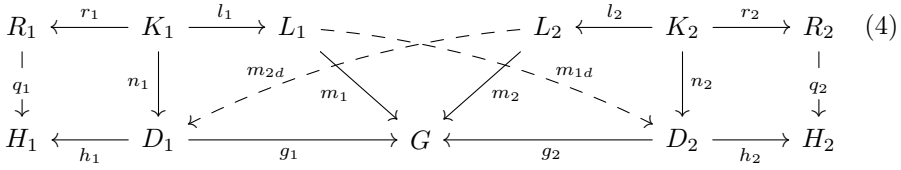
3 Conditions for parallel independence

Parallel independence is a property that can be satisfied or not by two (DPO or SQPO) transformation steps from the same object G, like in the situation depicted in Diagram (3), where we have two redexes (ρ_1, m_1) and (ρ_2, m_2) and transformation steps $G \overset{\rho_1, m_1}{\Longrightarrow} H_1$ and $G \overset{\rho_2, m_2}{\Longrightarrow} H_2$.



(3)

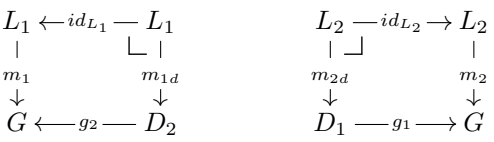
In the framework of the classical DPO approach, parallel independence was formulated in a categorical way [12,11,8] by requiring that each match factorizes through the context of the other transformation step. That is, there must exist the two dotted arrows of Diagram (4) so that the resulting triangles commute, i.e., $g_1 \circ m_{2d} = m_2$ and $g_2 \circ m_{1d} = m_1$. However, as shown explicitly with a counterexample in [4], this condition only works for DPO and SQPO with left-linear rules. For SQPO with non-left-linear rules the commutativity of the two triangles is not sufficient, but it is necessary to require the stronger condition that m_1 is reflected along g_2 by m_{1d}, and symmetrically for m_2.



(4)

Indeed, this is the condition of parallel independence that arose by addressing the Local Church-Rosser Problem in more general approaches where rules can be non-left-linear, like Reversible Sesqui-Pushout [10] and Rewriting in Span Categories [19]. We call this condition the *standard* one.

Definition 5 (Standard Condition). *Two redexes* (ρ_1, m_1) *and* (ρ_2, m_2) *as in Diagram (3) satisfy the* Standard Condition of parallel independence (STD-PI for short) *if the matches are reflected along the contexts, that is, if there are arrows* $m_{1d} : L_1 \to D_2$ *and* $m_{2d} : L_2 \to D_1$ *such that the two squares in Diagram (5) are pullbacks.*



(5)

Note that in most categories of interest, if the rules are left-linear then also morphisms $g_1 : D_1 \to G$ and $g_2 : D_2 \to G$ are mono. This holds either by Lemma 2.2 of [7] for the SQPO approach or, for the DPO approach in (quasi-)adhesive categories, because in such categories pushouts preserve (regular) monos [16]. Therefore in these cases by Fact 2(2) two redexes satisfy condition STD-PI if and only if they satisfy the more familiar condition of Diagram (4), i.e., there are arrows m_{1d} and m_{2d} making the two triangles commute.

Example 2 (the standard condition for parallel independence). Figure 2 shows a pair of parallel dependent redexes with rules LoadClone and Send, detected by the standard condition. Here and in the following, as a convention, we denote a pullback object (like $D1L2$) by concatenating the names of the cospan sources ($D1$ and $L2$ in this case), leaving implicit their morphisms to the common target (G). The conflict arises because the node cloned by LoadClone is used by Send, so the effect of Send after the application of LoadClone could differ depending on which of the clones is chosen by the match.

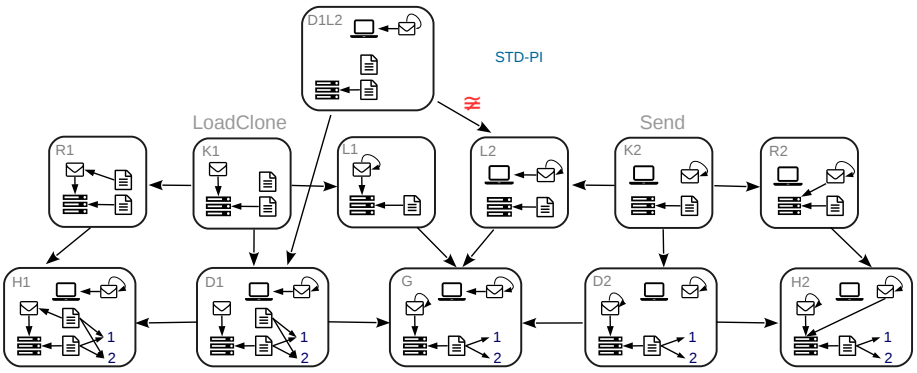


Fig. 2. Standard condition exposing a conflict between rules LoadClone and Send.

The condition of parallel independence for left-linear rules presented in [11] and shown in Diagram (4), being formulated in categorical terms, is very convenient for the proof of the Local Church-Rosser Theorem which is heavily based on diagrammatic constructions. But also a different, set-theoretical condition was proposed in [11] for DPO in the category of graphs, requiring that:

$$m_1(L_1) \cap m_2(L_2) = m_1(l_1(K_1)) \cap m_2(l_2(K_2)). \tag{6}$$

That is, each item needed by both redexes (in the image of both matches) must be preserved by both redexes (is also in the image of both interfaces).

$$\begin{array}{ccccc} K_1K_2 & & \xrightarrow{\pi_2^K} & & K_2 \\ \downarrow & \lrcorner & l & \lrcorner & \downarrow l_2 \\ & & L_1L_2 & \xrightarrow{\pi_2^L} & L_2 \\ \downarrow \pi_1^K & & \downarrow \pi_1^L & \lrcorner & \downarrow m_2 \\ K_1 & \xrightarrow{l_1} & L_1 & \xrightarrow{m_1} & G \end{array} \tag{7}$$

As discussed in [4], the classical condition of Diagram (4) is not a direct translation of this set-theoretical one, as the categorical counterpart of intersections are pullbacks. Diagram (7) shows the two pullbacks corresponding to

the left side (①) and to the right side (②) of Equation (6). The pullback objects are related by a unique mediating morphism $l : K_1K_2 \to L_1L_2$ such that $\pi_1^L \circ l = l_1 \circ \pi_1^K$ and $\pi_2^L \circ l = l_2 \circ \pi_2^K$, by the universal property of pullback ①.

By exploiting these pullbacks, the following condition of parallel independence was proposed in [4] as a direct categorical translation of Equation (6).

Definition 6 (Pullback Condition). *Redexes (ρ_1, m_1) and (ρ_2, m_2) as in Diagram (3) satisfy the* Pullback Condition of parallel independence (PB-PI for short) *if in Diagram (7) mediating arrow $l : K_1K_2 \to L_1L_2$ is an isomorphism.*

Next, we propose a third condition. To our opinion, it is simpler than the two previous ones and it captures the essence of parallel independence. It works for general rules, and it can be simplified in the left-linear case.

Definition 7 (Essential Condition). *Let (ρ_1, m_1) and (ρ_2, m_2) be two redexes in an object G, as in Diagram (3), and let $(L_1L_2, \pi_1^L, \pi_2^L)$ be the pullback defined by ① in Diagram 7. Then (ρ_1, m_1) and (ρ_2, m_2) satisfy the* Essential Condition of parallel independence (ESS-PI) *if the pullback of the matches is reflected along the left-hand sides. That is, if there exist arrows $\alpha_1 : L_1L_2 \to K_1$ and $\alpha_2 : L_1L_2 \to K_2$ such that the two squares of Diagram (8) are pullbacks.*

$$\begin{array}{ccc} L_1L_2 & \xrightarrow{id} & L_1L_2 \\ {\scriptstyle \alpha_1}\downarrow & & \downarrow{\scriptstyle \pi_1^L} \\ K_1 & \xrightarrow{l_1} & L_1 \end{array} \qquad \begin{array}{ccc} L_1L_2 & \xrightarrow{id} & L_1L_2 \\ {\scriptstyle \alpha_2}\downarrow & & \downarrow{\scriptstyle \pi_2^L} \\ K_2 & \xrightarrow{l_2} & L_2 \end{array} \tag{8}$$

Example 3 (the pullback and the essential conditions). The rule LoadClone is in conflict with itself when it tries to load a clone of the same data node in two distinct messages. Figure 3 shows how this conflict is captured by the pullback (PB-PI) and essential (ESS-PI) conditions of parallel independence.

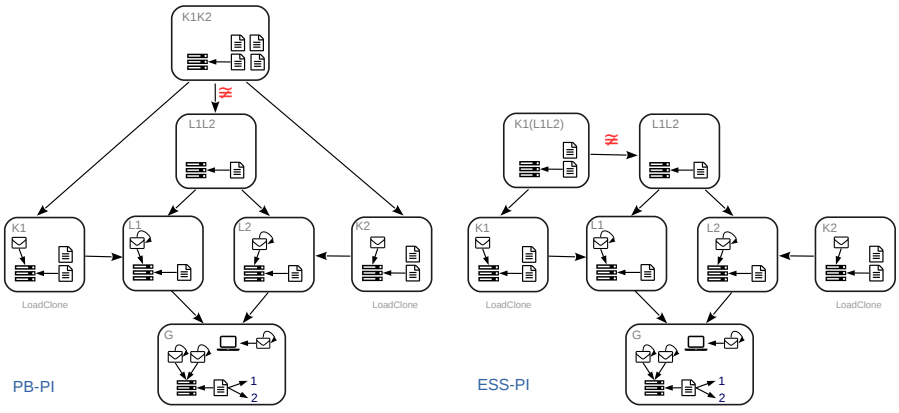


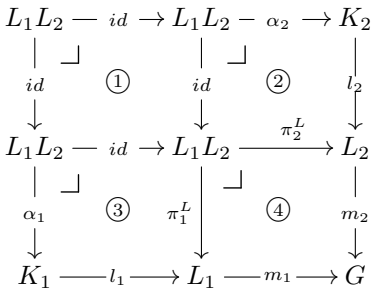
Fig. 3. Conditions PB-PI and ESS-PI show a conflict between two redexes of LoadClone.

4 Equivalence of conditions for parallel independence

We present here explicit proofs of the equivalence of the three conditions introduced in the previous section for DPO and SQPO rewriting. The equivalence between the Standard and the Pullback Conditions was proved in an indirect way in [4], by exploiting some results of [5]. The proofs that follow are complete and in a way simpler than those in [4], by reducing both conditions to the new one. In fact, we first prove the equivalence of conditions PB-PI and ESS-PI, and next the equivalence of conditions STD-PI and ESS-PI. The proofs are presented for SQPO rewriting with possibly non-left-linear rules. They also apply to DPO rewriting with left-linear rules under mild conditions recalled in Proposition 1.

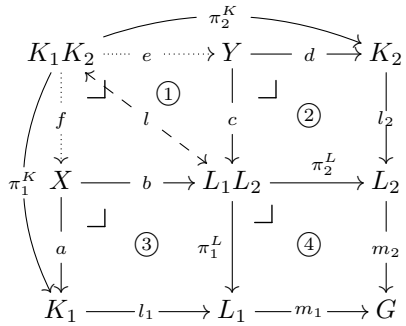
Theorem 1 (Equivalence of the Pullback and Essential Conditions). *Let* $\mathbf{C}$ *be a category with all pullbacks, and let* (ρ_1, m_1) *and* (ρ_2, m_2) *be two* SQPO *redexes in an object* G *of* $\mathbf{C}$*, as in Diagram (3). Then they satisfy condition* PB-PI *of Def. 6 if and only if they satisfy condition* ESS-PI *of Def. 7.*

Proof. **[If part]** Consider the following diagram:



(9)

All squares are pullbacks (④ by construction, ② and ③ by condition ESS-PI, and ① trivially). Thus by uniqueness of pullbacks up to a unique isomorphism commuting with the projections, we can deduce that the mediating arrow $l : K_1K_2 \to L_1L_2$ of (7) is an isomorphism.

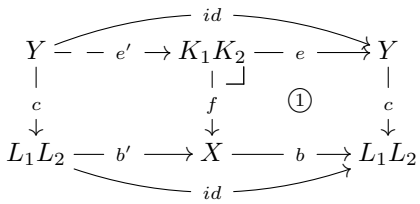


(10)

[Only if part] In Diagram (10) the outer diagram and ④ are pullbacks by construction, and their mediating arrow $l : K_1K_2 \to L_1L_2$ is an isomorphism

by condition PB-PI. Let ② and ③ be built as pullbacks: we have to show that b and c are isomorphism. Since ③ is a pullback and $\pi_1^L \circ l = l_1 \circ \pi_1^K$, there is a unique arrow $f : K_1K_2 \to X$ such that $b \circ f = l$ and $a \circ f = \pi_1^K$ and, symmetrically, there is a unique arrow $e : K_1K_2 \to Y$ such that $d \circ e = \pi_2^K$ and $c \circ e = l$. The resulting square ① is a pullback: in fact, the outer square and ② + ④ (by Fact 1(1)) are pullbacks, thus ① + ③ is a pullback by Fact 1(2); in turn since ③ is a pullback, also ① is, again by Fact 1(2). Since $b \circ f = l = c \circ e$ and l is an isomorphism, f and e are *sections* (that is, they have a left-inverse) and b and c are *retractions* (i.e., they have a right-inverse).

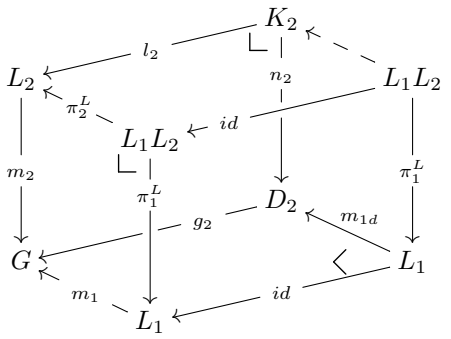
But pullbacks preserve retractions, as shown in the diagram to the right. If b is a retraction then there is a b' such that $b \circ b' = id$. Since ① is a pullback and the outer square commutes, there is an e' such that $e \circ e' = id$, thus e is a retraction as well.



Thus f and e are also retractions, and therefore they are isomorphisms. This implies that b and c are isomorphisms as well, as desired. □

Theorem 2 (Equivalence of the Standard and Essential Conditions). *Let* $\mathbf{C}$ *be a category with all pullbacks, and let* (ρ_1, m_1) *and* (ρ_2, m_2) *be two* SQPO *redexes in an object* G *of* $\mathbf{C}$*, as in Diagram (3). Then they satisfy condition* STD-PI *of Def. 5 if and only if they satisfy condition* ESS-PI *of Def. 7.*

Proof. **[Only if part]** Consider the following diagram:

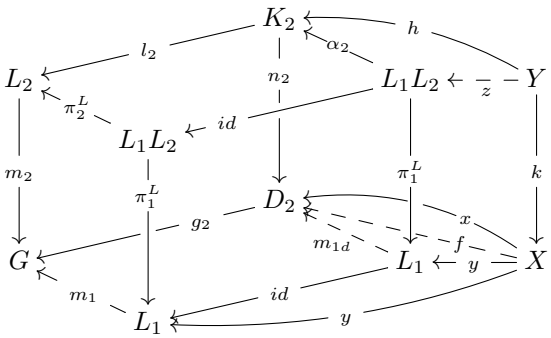


(11)

The back-left face is a pullback by construction, the bottom face is a pullback by condition STD-PI, and the front-right face is also trivially a pullback. Therefore since the front-left face commutes, by Fact 1(2) there is a unique arrow $L_1L_2 \to K_2$ making the upper face (i.e., the right square of Diagram (8)) a pullback. For the left square of Diagram (8) the proof is analogous.

[If part] Consider the cube in Diagram (12), ignoring for the time being objects X, Y, and the arrows starting from them. By Def. 3 the back-left face $K_2 \overset{n_2}{\to} D_2 \overset{g_2}{\to} G$ is a final pullback complement of $K_2 \overset{l_2}{\to} L_2 \overset{m_2}{\to} G$. Therefore, since the square made of the front-left and front-right faces is obviously a pullback

and the top face commutes by the essential condition, there is a unique arrow $m_{1d} : L_1 \to D_2$ such that the bottom and the back-right face commute.



(12)

In order to show that condition STD-PI holds, i.e. that the bottom face is a pullback, by Fact 2(1) it is sufficient to show that for each object X and arrows $x : X \to D_2, y : X \to L_1$ such that $g_2 \circ x = m_1 \circ y$, it holds (†) $m_{1d} \circ y = x$. In order to show this, let $K_2 \overset{h}{\leftarrow} Y \overset{k}{\to} X$ be the pullback of $K_2 \overset{n_2}{\to} D_2 \overset{x}{\leftarrow} X$. By composing this pullback with the back-left face we get a pullback with vertices Y, L_2, X and G, and therefore since the back-left face is a final pullback complement, there is a unique arrow $f : X \to D_2$ such that (a) $f \circ k = n_2 \circ h$ and (b) $g_2 \circ f = g_2 \circ x$. Thus (†) will follow by showing that both x and $m_{1d} \circ y$ satisfy properties (a) and (b). For x, (a') $x \circ k = n_2 \circ h$ holds by construction, and (b') is obvious. For $m_{1d} \circ y$, we have (b'') $g_2 \circ m_{1d} \circ y = m_1 \circ id \circ y = m_1 \circ y = g_2 \circ x$. In order to show (a'') $m_{1d} \circ y \circ k = n_2 \circ h$, observe that the composition of the top face (that is a pullback by the essential condition) and of the front-left face is a pullback, and that the outmost square commutes ($Y \overset{h}{\to} K_2 \overset{m_2 \circ l_2}{\to} G = Y \overset{y \circ k}{\to} L_1 \overset{m_1}{\to} G$), therefore there is a unique arrow $z : Y \to L_1 L_2$ such that (c) $\alpha_2 \circ z = h$ and (d) $\pi_1^L \circ id \circ z = y \circ k$. Thus we have $m_{1d} \circ y \circ k \overset{(d)}{=} m_{1d} \circ \pi_1^L \circ z = n_2 \circ \alpha_2 \circ z \overset{(c)}{=} n_2 \circ h$, as desired. □

Proposition 1 (Equivalence of Conditions for DPO rewriting). *If category* **C** *is quasi adhesive and the left-hand sides of the rules are regular monos (i.e. they are equalizers of pairs of parallel arrows), then Theorems 1 and 2 also apply to* DPO *redexes.*

Proof. By Proposition 12 of [7], the pushout complement of a regular-mono left-hand side and a match is also a final pullback complement. Therefore a DPO redex is also a SQPO redex. □

5 Implementation in the Verigraph System

Parallel independence is important for practical applications involving graph transformation, in particular for static analysis techniques. The equivalence of conditions STD-PI, ESS-PI and PB-PI means that tools are free to implement

any of them. In this context, time is usually the most critical resource, thus it should guide the choice of the algorithm. Here and in the next section we compare the performance of the three conditions based on their implementation in the Verigraph system and on their use, in various scenarios, for some concrete grammars defined in categories of typed graphs.

Verigraph [9] is implemented in Haskell, exploiting its abstraction mechanisms to promote separation between abstract and concrete code. This allows the algorithms for checking parallel independence to be implemented at an abstract level (based on arrows and composition) as an almost direct translation of categorical diagrams and definitions. Clearly, when such an abstract algorithm is applied to a concrete category of structures like (possibly typed) graphs, unary algebras, Petri nets, etc., its efficiency depends on the data structures and algorithms that implement the concrete structures, their morphisms and the primitive categorical operations on them. In general one cannot expect an instantiation of an abstract algorithm to be more efficient than an algorithm specifically designed for the concrete structures. This is why we compare only the algorithms as implemented in Verigraph, while we defer to future work a comparison with algorithms for parallel independence developed in other frameworks.

Verigraph supports the definition and analysis of DPO graph transformation systems, while the support for SQPO is under development. For this reason the experimental evaluation presented in the next section will consider left-linear rules only. Without loss of generality, we also assume that rules are right-linear, because the right-hand sides don't play a role in the conditions for parallel independence considered in this paper.

5.1 Data structures

We briefly describe now the data structures used to represent typed graphs and related concepts. Graphs are directed and unlabeled, and nodes and edges are identified by unique integers. Each graph is made of a list of node identifiers and a list of tuples (e, s, t), which are identifiers for the edge, its source and its target, respectively. This representation is convenient as most operations on graphs will traverse all its elements. Graph morphisms are represented as pairs of finite maps. We use the datatypes provided by the standard library of Haskell, which implements finite maps using balanced binary trees. A *typed graph* over a type graph T is a graph morphism $G^T : G \to T$. A *typed graph morphism* $f : G^T \to H^T$ is just a graph morphism $f : G \to H$ which commutes with the morphisms to T. Rules are spans $L^T \hookleftarrow K^T \rightarrowtail R^T$ of injective typed graph morphisms.

5.2 Primitive categorical operations

Verigraph provides many categorical operations as primitives for algorithm construction. For reasons of space, we only present a brief explanation of how the most relevant operations for testing parallel independence were implemented in the category of typed graphs. For more details, we refer to the source code [1].

Pullback Let $\mathbb{P}(X) = \{S \mid S \subseteq X\}$ and, given $f : X \to Y$, let the inverse $f^{-1} : Y \to \mathbb{P}(X)$ be defined as $f^{-1}(y) = \{x \mid f(x) = y\}$. The pullback of a cospan $(X \xrightarrow{f} Z \xleftarrow{g} Y)$ is obtained by taking the inverses $f^{-1} : Z \to \mathbb{P}(X)$ and $g^{-1} : Z \to \mathbb{P}(Y)$, then creating the disjoint union of the product of the preimages for each element of Z, that is, $\biguplus_{z \in Z} f^{-1}(z) \times g^{-1}(z)$. This is done independently for edges and nodes in order to construct the pullback graph and associated projection morphisms.

Pushout Complement The pushout complement (see Def. 1) of $(K \overset{l}{\rightarrowtail} L \xrightarrow{m} G)$ in the category of typed graphs exists iff the *gluing conditions* are satisfied [11]. These conditions are checked set-theoretically, before the actual construction. To construct the pushout complement of l and m when it exists, we compute the elements present in the image of m but not in that of $m \circ l$, and remove them from G obtaining the subgraph D. The morphism $g : D \rightarrowtail G$ is the inclusion, while $n : K \to D$ is obtained by restricting the codomain of $m \circ l$.

Isomorphism Check To check whether a morphism $f : X \to Y$ is an iso, we build the inverse $f^{-1} : Y \to \mathbb{P}(X)$ and then check that the image of each element of Y is a singleton.

Factorization Check Given a cospan $(X \xrightarrow{f} Z \overset{g}{\leftarrowtail} Y)$ with g mono, to search for morphisms $h : X \to Y$ such that $g \circ h = f$, we first take the inverse $g^{-1} : Z \to \mathbb{P}(Y)$. Then we determine the existence of a morphism h by computing $g^{-1} \circ f$ and checking if $\forall x \in X.\, g^{-1}(f(x)) \neq \emptyset$ holds.

5.3 Parallel independence test

Given the primitive operations just described, testing parallel independence can be achieved by constructing the required diagram elements and testing them for desired properties, for example, if a calculated morphism is iso. In the case of left-linear rules, there are two variants for conditions STD-PI and ESS-PI: as a consequence of Lemma 2, the reflection of f along g may either be tested by (i) constructing a pullback and checking for the isomorphism of one of its components, or (ii) by checking if f factorizes through g. We refer to the former by their regular names, and to the latter by STD-F-PI and ESS-F-PI. Figure 4 summarizes diagrammatically these five conditions: the left column contains the variants based on factorization and the right one those based on isomorphism tests. They are also categorized on whether they test the diagram elements *statically* (based on the rules and the matches only) or *dynamically* (using the construction of the FPBC/POC as part of the test). For readability, Figure 4 shows only the left part of the conditions, although the implemented routines test both sides. The complete source code together with the tested grammars are available as a Verigraph special release [1], in file `src/library/Analysis/ParallelIndependent.hs`. Since Haskell is a lazy language, strict evaluation was enforced to provide a better measure of the overall computational effort required in each test.

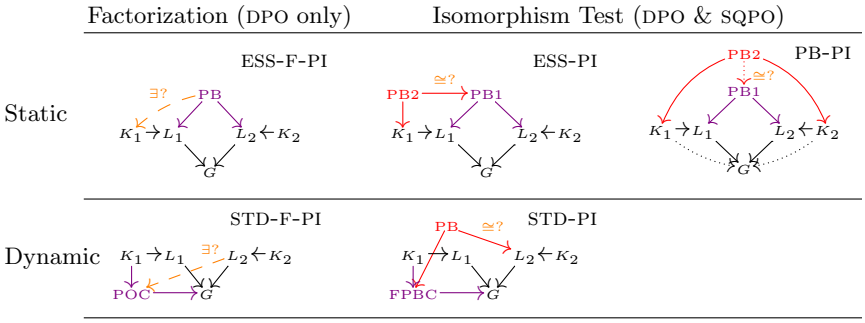


Fig. 4. Conditions for verifying parallel independence of transformations.

6 Experimental evaluation

The performance of evaluating these conditions should be sensitive to characteristics of the rules, of the *instance graphs* to which they are applied, and of the matches. The size of the instance graph G, for example, should strongly affect the dynamic conditions. The static conditions, on the other hand, should be less sensitive to the size of G, since the elements outside the matches are filtered out by the first pullback, leaving smaller graphs for the subsequent operations. Conversely, the presence of non-injective matches should affect the static conditions more than the dynamic ones, because the size of the pullback of the matches grows with the number of elements identified by them. We expect that considering non-left-linear rules this multiplicative effect may be reinforced.

To compare the performance of the five algorithms, two scenarios were used:

Elevator [17]: a grammar with 9 rules that models the behaviour of an elevator system. It will be referred to as ELEV.

Medical guidelines [2]: three grammars that model guidelines for a medical procedure, containing 36 rules in total. These grammars are referred to as MED1, MED2 and MED3.

These grammars are made of linear DPO rules. The files used to perform the benchmark and the obtained results are available at `https://verites.github.io/parallel-independence-benchmarks/`. The experiment compares the execution time needed to evaluate the five conditions over eight sets of inputs (pairs of redexes), two for each grammar, generated in the following ways.

Jointly epic pairs: Given a grammar GR, the input set GR-JE is obtained by considering for each pair of rules $(L_1 \leftarrow K_1 \rightarrow R_1, L_2 \leftarrow K_2 \rightarrow R_2)$, all possible partitions of $L_1 \uplus L_2$ (i.e. all epimorphisms $e : L_1 \uplus L_2 \twoheadrightarrow G$), and adding the pair $(m_1 = e \circ in_1, m_2 = e \circ in_2)$ to GR-JE iff both matches satisfy the gluing conditions. This is the standard approach when performing critical pair analysis.

Fixed instance graph: Given a grammar GR, a fixed instance graph G considerably larger than the left-hand sides of the rules of GR is constructed. Then for each pair of rules of GR all possible pairs of matches $(m_1 : L_1 \rightarrow G, m_2 :$

$L_2 \rightarrow G$) such that both matches satisfy the gluing conditions are added to set GR-IG. This represents testing parallel independence during concrete rewritings. For the grammars considered in the experimentation, graph G contains approximately 20 nodes.

For each input set, the five variants of the parallel independence test were executed ten times, the execution time was measured and the average time per pair of matches was calculated. The benchmark was executed on an Intel(R) Core(TM) i5-3330 machine with a 3.00GHz CPU and 16GB of RAM. Fig. 5 presents the results.

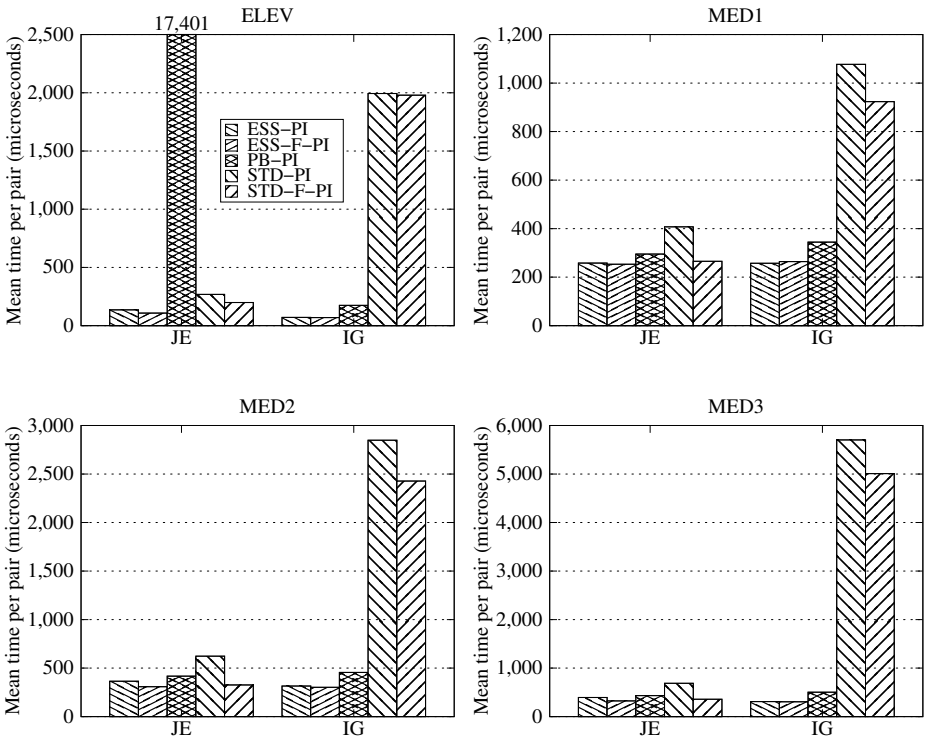


Fig. 5. Mean runtime of each algorithm, per pair of redexes, over all input sets. ELEV contains one outlier whose bar was truncated, with its numeric value written above it.

From the observed results, we conclude that the static variants (ESS-PI, ESS-F-PI and PB-PI) outperform the dynamic variants (STD-PI and STD-F-PI) in most cases, particularly with large instance graphs. One important exception was the case ELEV-JE, where PB-PI performed much worse than all other alternatives. We conjectured that this occurs due to the multiplicative effect of pullbacks in presence of non-injective matches. To confirm this, we repeated the experiment considering only the inputs with injective matches. In this case, the mean time

per pair of PB-PI was 146 microseconds, slower than ESS-PI ($114\mu s$) and ESS-F-PI ($99\mu s$) but faster than STD-PI ($265\mu s$) and STD-F-PI ($211\mu s$).

Regarding the comparison of using factorization (ESS-F-PI, STD-F-PI) or isomorphism test (ESS-PI, STD-PI), we observed that factorization seems consistently more efficient than pullback calculation, although in some cases the results were similar, as in MED3-JE. Thus, the factorization-based algorithms can be recommended when rules are left-linear.

In general, the essential conditions (ESS-PI and ESS-F-PI) presented very good performance in all situations. ESS-F-PI was in many cases the fastest for testing parallel independence. On the other hand, ESS-PI had an overall good performance, and often the difference between it and ESS-F-PI was insignificant. Therefore, the essential conditions can be recommended for all cases.

7 Conclusions

In this paper we have considered some definitions of parallel independence proposed for the Double-Pushout and the Sesqui-Pushout Approaches to graph transformation, and we proposed a new condition, that we called the *Essential Condition* (ESS-PI). We presented explicit proofs of equivalence of condition ESS-PI with the Standard (STD-PI) and the Pullback (PB-PI) Conditions previously proposed in the literature at an abstract categorical level. Next we have implemented five variants of the parallel independence test (two being optimized versions of STD-PI and ESS-PI for left-linear rules) for grammars based on categories of typed graphs in the Verigraph system. We evaluated the runtime efficiency of each condition over a collection of test cases based on DPO transformation with linear rules only, because the support of SQPO by Verigraph is still under development. Our experiments led to the conclusion that the essential condition has the best performance in most cases.

We foresee several developments of the work presented in this paper. From the more theoretical side we intend to investigate how the intuition behind condition ESS-PI could be exploited in other frameworks. For example, it should be possible to define a stronger version of ESS-PI equivalent to, but simpler than, the *strong parallel independence* considered in [13] for DPO transformations with injective matchings. We also intend to exploit condition ESS-PI for defining and computing efficiently *minimal conflict reasons* between redexes, as studied for example in [18,3], and to evaluate to what extent ESS-PI can be exploited to improve existing algorithms for Critical Pair Analysis. This is not obvious, because several optimization techniques have been developed (see e.g. [18]) that should be adapted to our condition for independence. In this context, we also intend to check formally under which assumptions condition ESS-PI is equivalent to the definition of parallel independence based on an initial pushout for the left-hand side of a rule, as proposed in [17,18]

Concerning the comparison of efficiency of the various conditions for parallel independence, the initial evaluations presented here should be completed to encompass non-left-linear rules as well. Next we intend to extend the compari-

son to algorithms for checking independence developed in other frameworks, like AGG [22].

References

1. Bezerra, J.S., Costa, A., Azzi, G., Rodrigues, L.M., Machado, R., Ribeiro, L.: Verites/verigraph: Parallel Independence Benchmarks (Jun 2017), https://doi.org/10.5281/zenodo.814246
2. Bezerra, J.S., Costa, A., Ribeiro, L., Cota, É.F.: Formal verification of health assessment tools: a case study. Electr. Notes Theor. Comput. Sci. 324, 31–50 (2016), https://doi.org/10.1016/j.entcs.2016.09.005
3. Born, K., Lambers, L., Strüber, D., Taentzer, G.: Granularity of conflicts and dependencies in graph transformation systems. In: ICGT 2017. LNCS, vol. 10373, pp. 125–141. Springer (2017), https://doi.org/10.1007/978-3-319-61470-0_8
4. Corradini, A.: On the definition of parallel independence in the algebraic approaches to graph transformation. In: STAF 2016 - GCM. LNCS, vol. 9946, pp. 101–111. Springer (2016), https://doi.org/10.1007/978-3-319-50230-4_8
5. Corradini, A., Duval, D., Prost, F., Ribeiro, L.: Parallelism in AGREE Transformations. In: ICGT 2016. LNCS, vol. 9761, pp. 35–51. Springer (2016), https://doi.org/10.1007/978-3-319-40530-8_3
6. Corradini, A., Gadducci, F.: On term graphs as an adhesive category. Electr. Notes Theor. Comput. Sci. 127(5), 43–56 (2005), https://doi.org/10.1016/j.entcs.2005.02.014
7. Corradini, A., Heindel, T., Hermann, F., König, B.: Sesqui-pushout rewriting. In: ICGT 2006. LNCS, vol. 4178, pp. 30–45. Springer (2006), http://dx.doi.org/10.1007/11841883_4
8. Corradini, A., Montanari, U., Rossi, F., Ehrig, H., Heckel, R., Löwe, M.: Algebraic Approaches to Graph Transformation - Part I: Basic Concepts and Double Pushout Approach. In: Handbook of Graph Grammars and Computing by Graph Transformations, Volume 1: Foundations. pp. 163–246 (1997)
9. Costa, A., Bezerra, J.S., Azzi, G., Rodrigues, L., Becker, T.R., Herdt, R.G., Machado, R.: Verigraph: A system for specification and analysis of graph grammars. In: SBMF 2016. pp. 78–94 (2016), https://doi.org/10.1007/978-3-319-49815-7_5
10. Danos, V., Heindel, T., Honorato-Zimmer, R., Stucki, S.: Reversible sesqui-pushout rewriting. In: ICGT 2014. LNCS, vol. 8571, pp. 161–176. Springer (2014), http://dx.doi.org/10.1007/978-3-319-09108-2_11
11. Ehrig, H.: Introduction to the algebraic theory of graph grammars (A survey). In: Graph-Grammars and Their Application to Computer Science and Biology. LNCS, vol. 73, pp. 1–69. Springer (1979), https://doi.org/10.1007/BFb0025714
12. Ehrig, H., Kreowski, H.: Parallelism of Manipulations in Multidimensional Information Structures. In: MFCS 1976. LNCS, vol. 45, pp. 284–293. Springer (1976), http://dx.doi.org/10.1007/3-540-07854-1_188
13. Habel, A., Müller, J., Plump, D.: Double-Pushout Graph Transformation Revisited. Mathematical Structures in Computer Science 11(5), 637–688 (2001), https://doi.org/10.1017/S0960129501003425
14. Kastenberg, H., Rensink, A.: Model checking dynamic states in GROOVE. In: SPIN 2006. LNCS, vol. 3925, pp. 299–305. Springer (2006), https://doi.org/10.1007/11691617_19

15. Knuth, D.E., Bendix, P.B.: Simple word problems in universal algebras†. In: LEECH, J. (ed.) Computational Problems in Abstract Algebra, pp. 263 – 297. Pergamon (1970), http://www.sciencedirect.com/science/article/pii/B978008012975450028X
16. Lack, S., Sobocinski, P.: Adhesive and quasiadhesive categories. Theoretical Informatics and Applications 39(3), 511–545 (2005), https://doi.org/10.1051/ita:2005028
17. Lambers, L.: Certifying Rule-Based Models using Graph Transformation. Ph.D. thesis, Technische Universität Berlin (2010), http://dx.doi.org/10.14279/depositonce-2348
18. Lambers, L., Ehrig, H., Orejas, F.: Efficient conflict detection in graph transformation systems by essential critical pairs. ENTCS 211, 17–26 (2008), https://doi.org/10.1016/j.entcs.2008.04.026
19. Löwe, M.: Graph rewriting in span-categories. In: ICGT 2010. LNCS, vol. 6372, pp. 218–233. Springer (2010), https://doi.org/10.1007/978-3-642-15928-2_15
20. Plump, D.: Evaluation of functional expressions by hypergraph rewriting. Ph.D. thesis, University of Bremen, Germany (1993), http://d-nb.info/940423774
21. Rosen, B.K.: A Church-Rosser theorem for graph grammars. ACM SIGACT News 7(3), 26–31 (1975), https://doi.org/10.1145/1008343.1008344
22. Taentzer, G.: AGG: A graph transformation environment for modeling and validation of software. In: AGTIVE 2003. pp. 446–453 (2003), https://doi.org/10.1007/978-3-540-25959-6_35