

This is a repository copy of *Confluence up to Garbage*.

White Rose Research Online URL for this paper:

<https://eprints.whiterose.ac.uk/161117/>

Version: Accepted Version

Proceedings Paper:

Plump, Detlef orcid.org/0000-0002-1148-822X and Campbell, Graham (2020) *Confluence up to Garbage*. In: Gadducci, Fabio and Kehrer, Timo, (eds.) *Proceedings 13th International Conference on Graph Transformation (ICGT 2020)*. *Lecture Notes in Computer Science*. Springer, pp. 20-37.

https://doi.org/10.1007/978-3-030-51372-6_2

Reuse

Items deposited in White Rose Research Online are protected by copyright, with all rights reserved unless indicated otherwise. They may be downloaded and/or printed for private study, or other acts as permitted by national copyright laws. The publisher or other rights holders may allow further reproduction and re-use of the full text version. This is indicated by the licence information on the White Rose Research Online record for the item.

Takedown

If you consider content in White Rose Research Online to be in breach of UK law, please notify us by emailing eprints@whiterose.ac.uk including the URL of the record and the reason for the withdrawal request.

Confluence up to Garbage

Graham Campbell^{1*} and Detlef Plump²

¹ School of Mathematics, Statistics and Physics, Newcastle University,
Newcastle upon Tyne, UK

`g.j.campbell12@newcastle.ac.uk`

² Department of Computer Science, University of York, York, UK

`detlef.plump@york.ac.uk`

Abstract. The transformation of graphs and graph-like structures is ubiquitous in computer science. When a system is described by graph-transformation rules, it is often desirable that the rules are both terminating and confluent so that rule applications in an arbitrary order produce unique resulting graphs. However, there are application scenarios where the rules are not globally confluent but confluent on a subclass of graphs that are of interest. In other words, non-resolvable conflicts can only occur on graphs that are considered as “garbage”. In this paper, we introduce the notion of confluence up to garbage and generalise Plump’s critical pair lemma for double-pushout graph transformation, providing a sufficient condition for confluence up to garbage by non-garbage critical pair analysis. We apply our results to language recognition by backtracking-free graph reduction, showing how to establish that a graph language can be decided by a system which is confluent up to garbage. We present two case studies with backtracking-free graph reduction systems which recognise a class of flow diagrams and a class of labelled series-parallel graphs, respectively. Both systems are non-confluent but confluent up to garbage.

Keywords: Graph Transformation · Confluence · Graph Languages · Decision Procedures

1 Introduction

Rule-based graph transformation and graph grammars date back to the late 1960s. The best developed theoretical framework is the so-called double-pushout (DPO) approach to graph transformation [12, 10]. When specifying systems in computer science by DPO graph transformation rules, it is often desirable that the rules are both terminating and confluent so that rule applications in an arbitrary order produce unique resulting graphs. However, there are application scenarios where the rules are not confluent but confluent on a subclass of graphs

* Supported by a Vacation Internship and a Doctoral Training Grant No. (2281162) from the Engineering and Physical Sciences Research Council (EPSRC) in the UK, while at University of York and Newcastle University, respectively.

that are of interest. In other words, non-resolvable conflicts can only occur on graphs that are considered as “garbage”.

In this paper, we introduce the notions of (local) confluence up to garbage and termination up to garbage in graph transformation. We generalise Plump’s Critical Pair Lemma [26,28] and Newmann’s Lemma [25] and thereby allow to check confluence up to garbage via non-garbage critical pair analysis. We apply our results to language recognition by backtracking-free graph reduction, showing how to establish that a graph language can be decided by a system which is confluent up to garbage. We present two case studies with backtracking-free graph reduction systems which recognise a class of flow diagrams and a class of labelled series-parallel graphs, respectively. Both systems are non-confluent but confluent up to garbage. Parts of this paper are based on Chapter 4 of an unpublished report [6], in turn developed from Campbell’s BSc Thesis [5].

2 Preliminaries

We review some terminology for binary relations, the DPO approach to graph transformation, graph languages, and confluence checking.

2.1 Abstract Reduction Systems

An *abstract reduction system* (ARS) is a pair (A, \rightarrow) where A is a *set* and \rightarrow a *binary relation* on A . We say that:

1. y is a *successor* to x if $x \xrightarrow{+} y$, and a *direct successor* if $x \rightarrow y$;
2. x and y are *joinable* if there is a z such that $x \xrightarrow{*} z \xleftarrow{*} y$. We write $x \downarrow y$;
3. \rightarrow is *confluent* if $y_1 \xleftarrow{*} x \xrightarrow{*} y_2$ implies $y_1 \downarrow y_2$;
4. \rightarrow is *locally confluent* if $y_1 \leftarrow x \rightarrow y_2$ implies $y_1 \downarrow y_2$;
5. \rightarrow is *terminating* if there is no infinite sequence $x_0 \rightarrow x_1 \rightarrow \dots$

The principle of *Noetherian Induction* is:

$$\frac{\forall x \in A, (\forall y \in A, x \xrightarrow{+} y \Rightarrow P(y)) \Rightarrow P(x)}{\forall x \in A, P(x)}$$

Theorem 1 (Noetherian Induction [1]). *Given an ARS (A, \rightarrow) , the principle of Noetherian induction holds if and only if \rightarrow is terminating.*

Theorem 2 (Newman’s Lemma [25]). *A terminating relation is confluent if and only if it is locally confluent.*

2.2 Labelled Graphs and Morphisms

We will be working with *directed labelled graphs* [15]. An *alphabet* is a pair $\Sigma = (\Sigma_V, \Sigma_E)$ of finite sets of node and edge labels from which a graph can be labelled. A *graph* (over Σ) is a tuple $G = (V, E, s, t, l, m)$ where V is a finite set

of nodes, E is a finite set of edges, $s : E \rightarrow V$ is the source function, $t : E \rightarrow V$ is the target function, $l : V \rightarrow \Sigma_V$ is the node labelling function, and $m : E \rightarrow \Sigma_E$ is the edge labelling function. We may write the components as V_G, E_G, s_G , etc.

A *graph morphism* $g : G \rightarrow H$ is a pair $g = (g_V, g_E)$ of functions $g_V : V_G \rightarrow V_H$ and $g_E : E_G \rightarrow E_H$ such that $g_V \circ s_G = s_H \circ g_E$, $g_V \circ t_G = t_H \circ g_E$, $l_G = l_H \circ g_V$ and $m_G = m_H \circ g_E$. We say g is *injective* (*surjective*, *bijective*) if both functions g_V and g_E are. A graph H is a *subgraph* of G , denoted by $H \subseteq G$, if there exists an *inclusion morphism* $i : H \rightarrow G$ with $i(x) = x$ for all items x .

It is well known that graphs and morphisms over Σ form a category. Graph morphisms are bijective if and only if they are isomorphisms in the categorical sense. Given a graph G , we write $[G]$ for the isomorphism class of G and call $[G]$ an *abstract graph*. We denote by $\mathcal{G}(\Sigma)$ the set of all abstract graphs over Σ .

2.3 Double-Pushout Graph Transformation

A *rule* is a pair of inclusions $r = \langle L \leftarrow K \rightarrow R \rangle$, where L is the left-hand side (LHS), K the interface, and R the right-hand side (RHS). A *match* of r in a graph G is an injective morphism $L \rightarrow G$. An application of rule r to G with match $g : L \rightarrow G$ requires to construct two pushouts as in Figure 1. We write $G \Rightarrow_{r,g} H$ for this application and call the diagram in Figure 1 a *direct derivation*.

$$\begin{array}{ccccc} L & \longleftarrow & K & \longrightarrow & R \\ \downarrow g & & \downarrow d & & \downarrow h \\ G & \longleftarrow & D & \longrightarrow & H \end{array}$$

Fig. 1. A direct derivation

Given r and the match $g : L \rightarrow G$, the direct derivation of Figure 1 exists if and only if the *dangling condition* is satisfied: nodes in $g(L - K)$ must not be incident to edges in $G - g(L)$. In this case the graphs D and H are determined uniquely up to isomorphism [10]. We call the injective morphism h the *comatch* of the rule application.

Given a set of rules \mathcal{R} , we write $G \Rightarrow_{\mathcal{R}} H$ if H is obtained from G by applying any of the rules from \mathcal{R} . We write $G \Rightarrow_{\mathcal{R}}^+ H$ if H is obtained from G by one or more rule applications, and $G \Rightarrow_{\mathcal{R}}^* H$ if $G \cong H$ or $G \Rightarrow_{\mathcal{R}}^+ H$.

By pushout properties, the relation $\Rightarrow_{\mathcal{R}}$ can be lifted to abstract graphs. Hence we have an ARS $(\mathcal{G}(\Sigma), \Rightarrow_{\mathcal{R}})$. This view gives us the definition of (local) confluence and termination for graph transformation systems.

2.4 Graph Languages

A graph language is simply a set of graphs, in the same way that a string language is a set of strings. Just like we can define string languages using string grammars, we can define graph languages using graph grammars, where we rewrite some start graph using a set of graph transformation rules. Derived graphs are then defined to be in the language exactly when they are terminally labelled.

Given a *graph transformation system* $T = (\Sigma, \mathcal{R})$, a subalphabet of *non-terminals* \mathcal{N} , and a *start graph* S over Σ , then a *graph grammar* is a tuple $\mathcal{G} = (\Sigma, \mathcal{N}, \mathcal{R}, S)$. We say that a graph G is *terminally labelled* if $l(V) \cap \mathcal{N}_V = \emptyset$ and $m(E) \cap \mathcal{N}_E = \emptyset$. Thus, we can define the *graph language* generated by \mathcal{G} :

$$L(\mathcal{G}) = \{[G] \mid S \Rightarrow_{\mathcal{R}}^* G, G \text{ terminally labelled}\}.$$

Given $\mathcal{G} = (\Sigma, \mathcal{N}, \mathcal{R}, S)$, we have $G \Rightarrow_r H$ if and only if $H \Rightarrow_{r-1} G$, for some $r \in \mathcal{R}$, by using the *comatch*. Moreover, $[G] \in L(\mathcal{G})$ if and only if $G \Rightarrow_{\mathcal{R}-1}^* S$ and G is terminally labelled. So we have a non-deterministic membership checking.

2.5 Confluence Checking

In 1970, Knuth and Bendix showed that confluence checking of terminating term rewriting systems is decidable [18]. Moreover, it suffices to compute all *critical pairs* and check their joinability [17, 1]. Unfortunately, for (terminating) graph transformation systems, confluence is not decidable in general, and joinability of critical pairs does not imply local confluence. In 1993, Plump showed that *strong joinability* of all critical pairs is sufficient but not necessary to show local confluence [26, 28].

The derivations $H_1 \xleftarrow{r_1, g_1} G \xrightarrow{r_2, g_2} H_2$ are *parallelly independent* if $(g_1(L_1) \cap g_2(L_2)) \subseteq (g_1(K_1) \cap g_2(K_2))$. We say two *parallelly independent* derivations are a *critical pair* if additionally $G = g_1(L_1) \cup g_2(L_2)$, and if $r_1 = r_2$ then $g_1 \neq g_2$. Every graph transformation system has only finitely many critical pairs.

Let $G \Rightarrow H$ be a *direct derivation*. Then the *track morphism* is defined to be the partial morphism $tr_{G \Rightarrow H} = in' \circ in^{-1}$, where in and in' are the bottom left and right morphisms in Figure 1, respectively. We define $tr_{G \Rightarrow^* H}$ inductively as the composition of track morphisms. The set of *persistent nodes* of a critical pair $\Phi : H_1 \leftarrow G \Rightarrow H_2$ is $Persist_{\Phi} = \{v \in G_V \mid tr_{G \Rightarrow H_1}(\{v\}), tr_{G \Rightarrow H_2}(\{v\}) \neq \emptyset\}$. That is, those nodes that are not deleted by the application of either rule.

A critical pair $\Phi : H_1 \leftarrow G \Rightarrow H_2$ is *strongly joinable* if it is *joinable* without deleting any of the persistent nodes, and the persistent nodes are identified when joining. That is, there exists a graph M and derivations $H_1 \Rightarrow_{\mathcal{R}}^* M \xleftarrow{\mathcal{R}}^* H_2$ such that $\forall v \in Persist_{\Phi}, tr_{G \Rightarrow H_1 \Rightarrow^* M}(\{v\}) = tr_{G \Rightarrow H_2 \Rightarrow^* M}(\{v\}) \neq \emptyset$.

Theorem 3 (Critical Pair Lemma [26, 28]). *A graph transformation system T is locally confluent if all its critical pairs are strongly joinable.*

The original proof of the Critical Pair Lemma needs the Commutativity, Clipping and Embedding Theorems, and some auxiliary definitions. We will need these intermediate results when we come to prove our generalised version.

Theorem 4 (Commutativity [11]). *If $H_1 \xleftarrow{r_1, g_1} G \xrightarrow{r_2, g_2} H_2$ are parallelly independent, then there is a graph G' and derivations $H_1 \Rightarrow_{r_2} G' \xleftarrow{r_1} H_2$.*

Let the derivation $\Delta : G_0 \Rightarrow^* G_n$ be given by pushouts $(1), (1'), \dots, (n), (n')$ and suppose there are pushouts $(\underline{1}), (\underline{1}'), \dots, (\underline{n}), (\underline{n}')$ whose vertical morphisms

are injective. Then, the derivation $\Delta' : G'_0 \Rightarrow^* G'_n$ consisting of the composed pushouts $(1 + \underline{1}), \dots, (n' + \underline{n}')$ is an instance of Δ based on the morphism $G_0 \rightarrow G'_0$. Moreover, we define the subgraph $\text{Use}\Delta$ to be all items x such that there is some $i \geq 0$ with $G_0 \Rightarrow^* G_i(x) \in \text{Match}(G_i \Rightarrow G_{i+1})$ where $\text{Match}(G_i \Rightarrow G_{i+1})$ is the image of the associated rule's left hand side graph under the match $L \rightarrow G_i$.

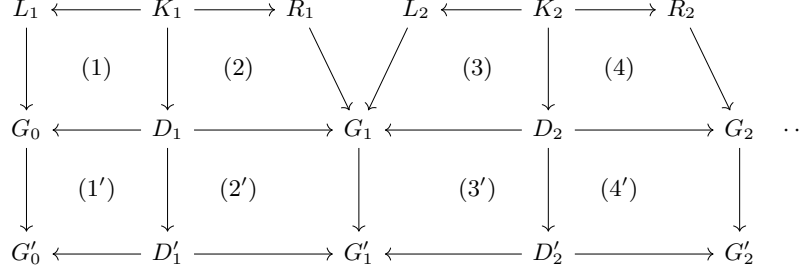


Fig. 2. Derivation instances

Theorem 5 (Clipping [27]). *Given a derivation $\Delta' : G' \Rightarrow^* H'$ and an injective morphism $h : G \rightarrow G'$ such that $\text{Use}\Delta' \subseteq h(G)$, there exists a derivation $\Delta : G \Rightarrow^* H$ such that Δ' is an instance of Δ based on h .*

Given a derivation $\Delta : G \Rightarrow^* H$ the subgraph of G , $\text{Persist}\Delta$, consists of all items x such that $\text{tr}_{G \Rightarrow^* H}(x)$ is defined.

Theorem 6 (Embedding [27]). *Let $\Delta : G \Rightarrow^* H$ be a derivation, $h : G \rightarrow G'$ an injective graph morphism, B_Δ be the discrete subgraph of G consisting of all nodes x such that $h(x)$ is incident to an edge in $G' \setminus h(G)$. If $B_\Delta \subseteq \text{Persist}\Delta$, then there exists a derivation $\Delta' : G' \Rightarrow^* H'$ such that Δ' is an instance of Δ based on h . Moreover, there exists a pushout of $t : B_\Delta \rightarrow H$ along $h' : B_\Delta \rightarrow C_\Delta$ where $C_\Delta = (G' \setminus h(G)) \cup h(B_\Delta)$ and t is the restriction of $\text{tr}_{G \Rightarrow^* H}$ to B_Δ .*

3 Closedness and Confluence up to Garbage

In this section, we introduce (local) confluence and termination up to garbage, and closedness. We show that if we have closedness and termination up to garbage, then local confluence up to garbage implies confluence up to garbage: the Generalised Newmann's lemma. Moreover, we recap that closedness is undecidable in general, in the context of DPO graph transformation.

3.1 Closedness and Garbage

Definition 1. Let $T = (\Sigma, \mathcal{R})$ be a GT system, and $\mathcal{D} \subseteq \mathcal{G}(\Sigma)$ be a set of abstract graphs. Then, a graph G is called *garbage* if $[G] \notin \mathcal{D}$ and \mathcal{D} is *closed* under T if for all G, H such that $G \Rightarrow_{\mathcal{R}} H$, if $[G] \in \mathcal{D}$ then $[H] \in \mathcal{D}$.

The idea is that a set of abstract graphs \mathcal{D} represents the *good input*, and the *garbage* is the graphs that are not in this set. \mathcal{D} need not be explicitly generated

by a graph grammar. For example, it could be defined by some (monadic second-order [8]) logical formula.

Example 1. Consider the reduction rules in Figure 3. The language of acyclic graphs is *closed* under the GT system $((\{\square\}, \{\square\}), \{r_1\})$, and the language of trees (forests) and its complement are both *closed* under $((\{\square\}, \{\square\}), \{r_2\})$.

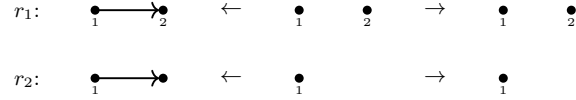


Fig. 3. Example Reduction Rules

Definition 2 (Closedness Problem).

Input: A GT system $T = (\Sigma, \mathcal{R})$ and a graph grammar \mathcal{G} over Σ .

Question: Is $L(\mathcal{G})$ *closed* under T ?

It turns out that closedness is undecidable in general, even if we restrict ourselves to recursive languages and terminating GT systems. In 1998, Fradet and Le Métayer showed the following result:

Theorem 7 (Undecidable Closedness [14]). *The closedness problem is undecidable in general, even for terminating GT systems T with only one rule, and \mathcal{G} an edge replacement grammar.*

3.2 Confluence up to Garbage

We can now define (*local*) *confluence* and *termination* up to garbage, allowing us to say that, ignoring the garbage graphs, a system is (*locally*) *confluent*.

Definition 3. Let $T = (\Sigma, \mathcal{R})$, $\mathcal{D} \subseteq \mathcal{G}(\Sigma)$. Then:

1. if for all graphs G, H_1, H_2 , such that $[G] \in \mathcal{D}$, $H_1 \leftarrow_{\mathcal{R}} G \Rightarrow_{\mathcal{R}} H_2$ implies that H_1, H_2 are *joinable*, then T is *locally confluent* (up to garbage) on \mathcal{D} ;
2. if for all graphs G, H_1, H_2 , such that $[G] \in \mathcal{D}$, $H_1 \leftarrow_{\mathcal{R}}^* G \Rightarrow_{\mathcal{R}}^* H_2$ implies that H_1, H_2 are *joinable*, then T is *confluent* (up to garbage) on \mathcal{D} ;
3. if there is no infinite derivation sequence $G_0 \Rightarrow_{\mathcal{R}} G_1 \Rightarrow_{\mathcal{R}} G_2 \Rightarrow_{\mathcal{R}} \dots$ such that $[G_0] \in \mathcal{D}$, then T is *terminating* (up to garbage) on \mathcal{D} .

The following is an immediate consequence of set inclusion:

Proposition 1. *Let $T = (\Sigma, \mathcal{R})$, $\mathcal{D} \subseteq \mathcal{G}(\Sigma)$, $\mathcal{E} \subseteq \mathcal{D}$. Then (*local*) *confluence* on \mathcal{D} implies (*local*) *confluence* on \mathcal{E} , and similarly for *termination*.*

Example 2. Looking again at r_1 and r_2 from our first example, it is easy to see that r_1 is *terminating* and *confluent up to garbage* on the language of acyclic graphs, but is not *confluent* on all graphs. Similarly, r_2 is *terminating* and *confluent up to garbage* on the language of trees.

Example 3. Consider the rules in Figure 4. Clearly they are terminating, since they are size reducing. Moreover, the language of all linked lists with edge labels a or b and its complement are closed under the rules. The rules are not locally confluent. To see this, consider the 3-cycle with edges labelled with a, a, b . It is possible for the cycle to be reduced to either the 2-cycle with edges a and a or the 2-cycle with edge a and b . Neither of these cycles can be reduced further, and so we have a counter example to confluence. These rules are locally confluent on linked lists. Moreover, an input graph G is a linked list if and only if it can be reduced using these rules to a length one linked list.



Fig. 4. List Reduction Rules

Theorem 8 (Generalised Newman’s Lemma). *Let $T = (\Sigma, \mathcal{R}), \mathcal{D} \subseteq \mathcal{G}(\Sigma)$. If T is terminating on \mathcal{D} and \mathcal{D} is closed under T , then T is confluent on \mathcal{D} if and only if it is locally confluent on \mathcal{D} .*

Proof. This can be seen by Noetherian Induction (Figure 5), due to the fact that closedness ensures applicability of the induction hypothesis. \square

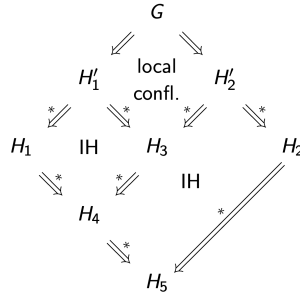


Fig. 5. Induction Step Diagram

4 Generalised Critical Pair Lemma

In this section, we generalise Plump’s Critical Pair Lemma, providing a machine checkable sufficient condition for local confluence up to garbage. For this, we need to define a notion of subgraph closure and non-garbage critical pairs.

4.1 Subgraph Closure

In the proof of the traditional critical pair lemma for (hyper)graphs, the argument is that if a pair of derivations is not parallelly independent, then it must be the case that a critical pair can be embedded within it. In our new setting,

the possible start graphs will be restricted, since some of the graphs will be *garbage*. We are only interested in those critical pairs with start graphs that can be embedded in non-garbage graphs. This is exactly the statement that the start graph of the critical pair is in the subgraph closure of the non-garbage graphs.

Definition 4. Let $\mathcal{D} \subseteq \mathcal{G}(\Sigma)$ be a set of abstract graphs. Then \mathcal{D} is *subgraph closed* if for all graphs G, H , such that $H \subseteq G$, if $[G] \in \mathcal{D}$, then $[H] \in \mathcal{D}$. The *subgraph closure* of \mathcal{D} , denoted $\widehat{\mathcal{D}}$, is the smallest set containing \mathcal{D} that is *subgraph closed*.

Proposition 2. *Given $\mathcal{D} \subseteq \mathcal{G}(\Sigma)$, $\widehat{\mathcal{D}}$ always exists, and is unique. Moreover, $\mathcal{D} = \widehat{\mathcal{D}}$ if and only if \mathcal{D} is subgraph closed.*

Proof. The key observations are that the subgraph relation is transitive, and each graph has only finitely many subgraphs. Clearly, the smallest possible set containing \mathcal{D} is just the union of all subgraphs of the elements of \mathcal{D} , up to isomorphism. This is the unique subgraph closure of \mathcal{D} . \square

$\widehat{\mathcal{D}}$ always exists, however it need not be decidable, even when \mathcal{D} is! It is not obvious what conditions on \mathcal{D} ensure that $\widehat{\mathcal{D}}$ is decidable. Interestingly, the classes of regular and context-free string languages are actually closed under substring closure [4].

Example 4. \emptyset , $\mathcal{G}(\Sigma)$, and the language of discrete graphs are subgraph closed.

Example 5. The subgraph closure of the language of trees is the language of forests. The subgraph closure of the language of connected graphs is the language of all graphs.

4.2 Non-Garbage Critical Pairs

We now define non-garbage critical pairs, which allow us to ignore certain pairs, which if all are strongly joinable, will allow us to conclude local confluence up to garbage, even in the presence of (local) non-confluence on all graphs.

Definition 5. Let $T = (\Sigma, \mathcal{R})$, $\mathcal{D} \subseteq \mathcal{G}(\Sigma)$. A *critical pair* $H_1 \leftarrow G \Rightarrow H_2$ is *non-garbage* if $[G] \in \widehat{\mathcal{D}}$.

Lemma 1. *Given a GT system $T = (\Sigma, \mathcal{R})$ and $\mathcal{D} \subseteq \mathcal{G}(\Sigma)$, then there are only finitely many non-garbage critical pairs up to isomorphism. Moreover, if $\widehat{\mathcal{D}}$ is decidable, then one can find them in finite time.*

Proof. There are only finitely many critical pairs for T , up to isomorphism, and there exists a terminating procedure for generating them. It then remains to filter out the garbage pairs, which can always be done if $\widehat{\mathcal{D}}$ is decidable. \square

Corollary 1. *Let $T = (\Sigma, \mathcal{R})$, $\mathcal{D} \subseteq \mathcal{G}(\Sigma)$ be such that T is terminating on \mathcal{D} and $\widehat{\mathcal{D}}$ is decidable. Then, one can decide if all the non-garbage critical pairs are strongly joinable.*

Proof. By Lemma 1, we can generate all the pairs, but then since T is terminating on D , there are only finitely many successor graphs to be generated. We can then test each for strong joinability in finite time. \square

Theorem 9 (Generalised Critical Pair Lemma). *Let $T = (\Sigma, \mathcal{R})$, $\mathcal{D} \subseteq \mathcal{G}(\Sigma)$. If all its non-garbage critical pairs are strongly joinable, then T is locally confluent on \mathcal{D} .*

Proof. Our proof is a generalisation of Plump’s original proof of the Critical Pair Lemma for (hyper)graphs (Theorem 3) [26, 28]. We need to show that every pair of derivations $H_1 \leftarrow_{r_1, g_1} G \Rightarrow_{r_2, g_2} H_2$ such that G is non-garbage can be joined. There are two cases to consider. Firstly, if the derivations are parallelly independent, then by Theorem 4, the result is immediate. Otherwise, we must consider the case that they are not parallelly independent.

By Theorem 5, we can factor out a pair $T_1 \leftarrow S \Rightarrow T_2$. Since critical pairs are, by construction, the overlaps of rule left hand sides, it must be the case that this pair is actually a critical pair. Moreover, since $[G] \in \mathcal{D}$, then $[S] \in \widehat{\mathcal{D}}$ and so the critical pair must be non-garbage, and must be strongly joinable to U . We can now apply Theorem 6 to $T_1 \Rightarrow^* U$ and $T_2 \Rightarrow^* U$, separately, giving result graphs M_1 and M_2 (applicability of the theorem is a consequence of strong joinability). To see that M_1 and M_2 are isomorphic follows from elementary properties of pushouts along monomorphisms [28]. \square

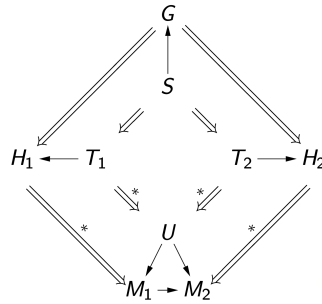


Fig. 6. Generalised Critical Pair Lemma Diagram

Corollary 2. *Let $T = (\Sigma, \mathcal{R})$, $\mathcal{D} \subseteq \mathcal{G}(\Sigma)$. If T is terminating on \mathcal{D} , \mathcal{D} is closed under T , and all T ’s non-garbage critical pairs are strongly joinable, then T is confluent on \mathcal{D} .*

Proof. By the above theorem, T is locally confluent up to garbage, so by the Generalised Newman’s Lemma (Theorem 8), T is confluent up to garbage. \square

Obviously, checking for local confluence up to garbage is undecidable in general, even when $\widehat{\mathcal{D}}$ is decidable and the system is terminating and closed. What is remarkable though, is that local confluence up to garbage is actually undecidable in general for a terminating non-length-increasing string rewriting systems and \mathcal{D} a regular string language [7]!

4.3 Checking for Confluence up to Garbage

Given a GT system T and a language \mathcal{D} (possibly specified by a grammar), the process is to:

1. Establish (by means of direct proof) that T is terminating on \mathcal{D} and \mathcal{D} is closed under T . If this is not true, one may want to restart with some language containing \mathcal{D} to try to establish closedness.
2. Generate the finitely many non-garbage critical pairs of T .
3. Check if each generated pair is (strongly) joinable.

If all the pairs are strongly joinable, then we have confluence up to garbage due to Corollary 2. If all the pairs are joinable, but not all strongly, then we cannot draw any conclusions, but one may be able to construct a counter example to confluence by attaching context to nodes. Finally, if one of the pairs is not joinable at all, then we have a direct counter example to confluence, and we can conclude non-confluence up to garbage.

5 Language Recognition

In this section, we introduce a general notion of what it means to recognise a language, and what it means to be a confluent decider. We then demonstrate the applicability of our earlier results by showing that there are confluent deciders for Extended Flow Diagrams and Labelled Series-Parallel Graphs, even in the absence of confluence. We thus have algorithms, specified by reduction rules, that can check membership of these languages without needing to backtrack.

5.1 Confluent Recognition

One can think of graph transformation systems in terms of grammars that define languages. If they are terminating, then membership testing is decidable, but in general, non-deterministic in the sense that a deterministic algorithm must backtrack if it produces a normal form not equal to the start graph, to determine if another derivation sequence could have reached it. If the system is confluent too, then the algorithm becomes deterministic.

In general, the requirement of confluence is too strong, and one only requires confluence on the language we are recognising. Using the results from the last section, it is often possible to prove local confluence up to garbage using the Generalised Critical Pair Lemma, and then, in the presence of termination and closure, use the Generalised Newman's Lemma to show confluence up to garbage. Closedness and language recognition has actually been considered before by Bakewell, Plump, and Runciman, in the context of languages specified by reduction systems without non-terminals [3], but without the development of the theory we have provided.

Before continuing, we must provide a formal definition of what it means to recognise a language, and that grammars satisfy our definition by considering

their rules in reverse, abstracting away from grammars, with a more general definition that accounts for the fact that reduction systems may need auxiliary symbols, not in the input, in the same way grammars can use non-terminals.

Definition 6 (Language Recognition). Let $T = (\Sigma, \mathcal{R})$ be a GT system, $\mathcal{I} \subseteq \Sigma$ an input alphabet, and \mathcal{S} a finite set of graphs over Σ . Then we say that (T, \mathcal{S}) *recognises* a language \mathcal{L} over \mathcal{I} if for all graphs G over \mathcal{I} , $[G] \in \mathcal{L}$ if and only if $G \Rightarrow_{\mathcal{R}}^* S$ for some $S \in \mathcal{S}$.

Theorem 10 (Membership Checking). *Given a grammar $\mathcal{G} = (\Sigma, \mathcal{N}, \mathcal{R}, S)$, $[G] \in L(\mathcal{G})$ if and only if $G \Rightarrow_{\mathcal{R}^{-1}}^* S$ and G is terminally labelled. That is, $((\Sigma, \mathcal{R}^{-1}), \{S\})$ recognises $L(\mathcal{G})$ over $\Sigma \setminus \mathcal{N}$.*

Proof. The key is that rules and derivations are invertible, which means that if S can be derived from G using the reverse rules, then G can be derived from S using the original rules so is in the language. If S cannot be derived from G , then G cannot be in the language since that would imply there was a derivation sequence from S to G which we could invert to give a contradiction. \square

We are now ready to define *confluent deciders*, and show that such systems can test for language membership without backtracking.

Definition 7 (Confluent Decider). Let $T = (\Sigma, \mathcal{R})$ be a GT system, $\mathcal{I} \subseteq \Sigma$ an input alphabet, and \mathcal{S} a finite set of graphs over Σ . Then we say that (T, \mathcal{S}) is a *confluent decider* for a language \mathcal{L} over \mathcal{I} if (T, \mathcal{S}) recognises \mathcal{L} over \mathcal{I} , T is terminating on $\mathcal{G}(\mathcal{I})$, and T is confluent on \mathcal{L} .

Theorem 11 (Confluent Decider Correctness). *Given a confluent decider (T, \mathcal{S}) for a language \mathcal{L} over $\mathcal{I} \subseteq \Sigma$ and an input graph G over \mathcal{I} , the following algorithm is correct: Compute a normal form of G by deriving successor graphs using T as long as possible. If the result graph is isomorphic to S , the input graph is in the language. Otherwise, the graph is not in the language.*

Proof. Suppose G is not in \mathcal{L} . Then, since T is terminating on $\mathcal{G}(\mathcal{I})$ our algorithm must be able to find a normal form of G , say H , and because T recognises \mathcal{L} , it must be the case that H is not isomorphic to S , and so the algorithm correctly decides that G is not in \mathcal{L} .

Now, suppose that G is in \mathcal{L} . Then, because T is terminating, as before, we must be able to derive some normal form, H . But then, since T is both confluent on \mathcal{L} and recognises \mathcal{L} , it must be the case that H is isomorphic to S , and so the algorithm correctly decides that G is in \mathcal{L} . \square

What we really want is a version of Theorem 10 for instantiating confluent deciders. We really want is a Since both termination and confluence testing is undecidable in general, we cannot hope for an effective procedure, even for a terminating system, however the theory we introduced in the previous sections will help by automating local confluence checking. It just remains for us to choose a suitable set \mathcal{D} , and proceed in a similar way to as described in Subsection 4.3.

For the remainder of this section, we will look at two examples that demonstrate how we can use the Generalised Newman's Lemma and Generalised Critical Pair Lemma to show that we have a confluent decider for a language, given a grammar that generates the language.

5.2 Extended Flow Diagrams

In 1976, Farrow, Kennedy and Zucconi presented *semi-structured flow graphs*, defining a grammar with confluent reduction rules [13]. Plump has considered a restricted version of this language: *extended flow diagrams* (EFDs) [28]. The reduction rules for *extended flow diagrams* are a confluent decider for the EFDs, despite not being confluent.

Definition 8. The language of *extended flow diagrams* is generated by $\text{EFD} = (\Sigma, \mathcal{N}, \mathcal{R}, S)$ where $\Sigma_V = \{\bullet, \square, \diamond\}$, $\Sigma_E = \{t, f, \square\}$, $\mathcal{N}_V = \mathcal{N}_E = \emptyset$, $\mathcal{R} = \{\text{seq}, \text{while}, \text{ddec}, \text{dec1}, \text{dec2}\}$, and $S = \bullet \rightarrow \square \rightarrow \bullet$.

In the next figure, the shorthand notation with the numbers under the nodes places such nodes in the interface graph of the rules. We assume that the interface graphs are discrete (have no edges).

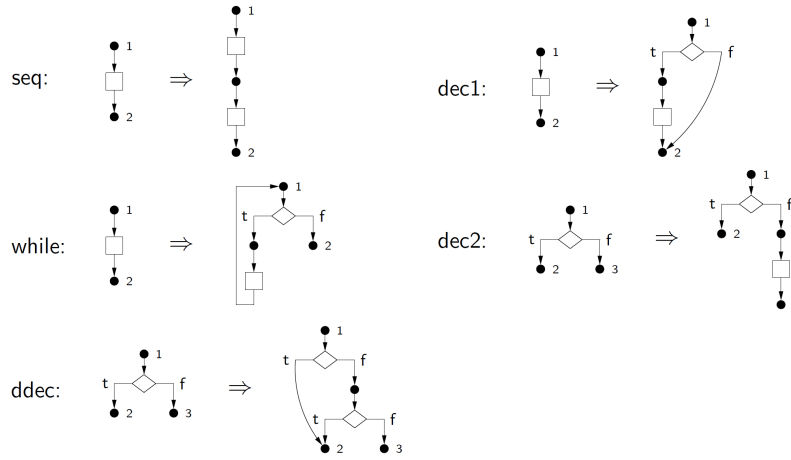


Fig. 7. EFD Grammar Rules

Lemma 2. *Every directed cycle in an EFD contains a t -labelled edge*

Proof. By induction. □

Theorem 12 (Confluent EFD Decider). *Let $T = (\Sigma, \mathcal{R}^{-1})$. Then $(T, \{S\})$ is a confluent decider for $L(\text{EFD})$ over Σ .*

Proof. By Theorem 10, T recognises $L(\text{EFD})$ over Σ , and one can see that it is terminating since each rule is size reducing.

We now proceed by performing critical pair analysis on T . There are ten critical pairs, all but one of which are strongly joinable apart from one (Figure 8). Now observe that Lemma 2 tells us that EFDs cannot contain such cycles. With this knowledge, we define \mathcal{D} to be all graphs such that directed cycles contain at least one t -labelled edge. Clearly, \mathcal{D} is subgraph closed, and then by our Generalised Critical Pair Lemma (Theorem 9), we have that T is locally confluent on \mathcal{D} .

Next, it is easy to see that \mathcal{D} is closed under T , so we can use Generalised Newman’s Lemma (Theorem 8) to conclude confluence on \mathcal{D} and thus, by Proposition 1, T is confluent on $L(\text{EFD})$.

Thus, T is a confluent decider for $L(\text{EFD})$ over Σ , as required. \square

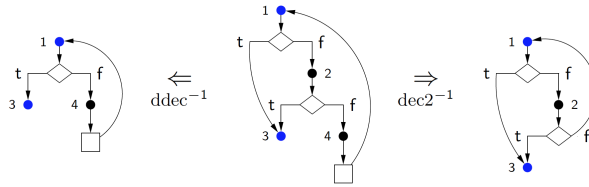


Fig. 8. Non-Joinable EFD Critical Pair

5.3 Series-Parallel Graphs

Series-parallel graphs were introduced by Duffin [9] as a model of electrical networks. A more general version of the class was introduced by Lawler [23] and Monma and Sidney [24] as a model for scheduling problems.

Definition 9. *Series-parallel* graphs are inductively defined:

1. P is a series-parallel graph where s is the *source* and t the *sink*.
2. The class of series-parallel graphs is closed under *parallel composition* and *sequential composition*.

where $P = \bullet \xrightarrow{s} \bullet \xrightarrow{t}$, parallel composition identifies the two sources and the two sinks, and sequential composition identifies the sink of one with the source of another.

Duffin showed that a graph is series-parallel if and only if it can be reduced to P by a sequence of series and parallel reductions. We can rephrase this in terms of a graph grammar.

Theorem 13 (SP Recognition [29]). *The class of series-parallel graphs is the language generated by grammar $SP = ((\{\square\}, \{\square\}), (\emptyset, \emptyset), \{s, p\}, P)$.*



Fig. 9. Series-Parallel Grammar Rules

By traditional critical pair analysis, one can establish that the reversed rules are confluent, however, we run into a problem if we want to consider arbitrarily labelled graphs. Consider the case where the edge alphabet is of size 2, rather than size 1. The obvious modification to the rules is to use all combinations of labels in LHS graphs (Figure 10), however Hristakiev and Plump [16] observed that when doing (the equivalent of) this in GP 2, we no longer have confluence. We exhibit a counter example to confluence in Figure 11.

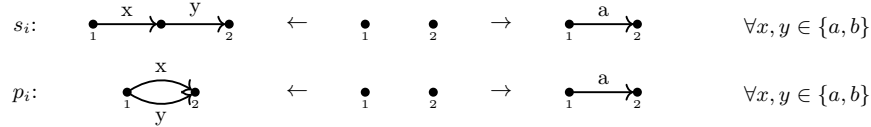


Fig. 10. Labelled Series-Parallel Reduction Rules

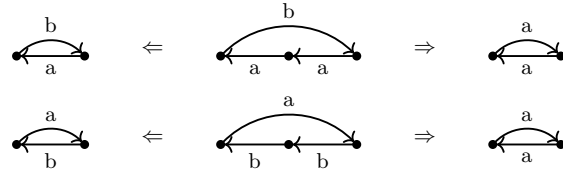


Fig. 11. Non-Joinable Labelled Series-Parallel Pairs

All is not lost, however, because we can use our new theory to show that, via non-garbage critical pair analysis, the new system is confluence up to garbage, and so we can show that we have a confluent decider. Moreover, our system for two edge labels can be easily generalised for any finite edge alphabet.

Definition 10. The class of *labelled series-parallel graphs* (LSPs) is all series-parallel graphs, but with arbitrary edge labels chosen from $\Sigma_E = \{a, b\}$.

Theorem 14 (Confluent LSP Decider). Let $\Sigma = (\{\square\}, \{a, b\})$, $T = (\Sigma, \{s_i, p_i \mid i \in I\})$ (where I indexes label choice), $P_a = \bullet \xrightarrow{a} \bullet$ and $P_b = \bullet \xrightarrow{b} \bullet$. Then $(T, \{P_a, P_b\})$ is a confluent decider for the labelled series-parallel graphs over Σ .

Proof. We denote by \mathcal{L} the language of all labelled series-parallel graphs.

Our rules are structurally the same as the unlabelled rules, so because our LHS graphs are arbitrarily labelled, language recognition of \mathcal{L} over Σ follows from Theorem 13. Termination follows from the fact that the combined metric of graph size plus number of b -labelled edges strictly decreases with each derivation.

We now proceed by performing critical pair analysis on T . We find that we have two non-isomorphic critical pairs that are not joinable (Figure 11). These pairs have a cyclic start graph, but the series-parallel graphs are acyclic, so we can define \mathcal{D} to be the language of acyclic graphs over Σ , thus classifying these two pairs as garbage. The remaining critical pairs are strongly joinable, so by Theorem 9, we have that T is locally confluent on \mathcal{D} .

We find that all the non-garbage critical pairs are strongly joinable. We have, up to isomorphism, two garbage critical pairs. These are not even joinable, which give us a counter example to (local) confluence, but since all our non-garbage pairs are strongly joinable, we can claim local confluence up to garbage.

Next, it is easy to see that \mathcal{D} is closed under T , so we can use Theorem 8 to conclude confluence on \mathcal{D} and thus, by Proposition 1, T is confluent on \mathcal{L} . \square

6 Conclusion and Future Work

In this paper we have introduced (local) confluence and termination up to garbage for DPO graph transformation systems, and shown that Newmann’s Lemma and Plump’s Critical Pair Lemma can be generalised, providing us with machine checkable conditions for confluence up to garbage, using only critical pairs. Of course, confluence up to garbage of terminating graph transformation systems is undecidable in general, however, now we can detect more positive cases of confluence up to garbage using non-garbage critical pair analysis, where we previously would have been unable to draw a conclusion due to non-strong joinability of some critical pairs.

In particular, our results can be directly applied to recognition of languages, which we have demonstrated with Extended Flow Diagrams and Labelled Series-Parallel Graphs. We have backtracking-free algorithms that apply reduction rules as long as possible, with correctness established via non-garbage critical pair analysis. We also anticipate there to be other applications, since there are many other reasons one would want to show confluence up to garbage, such as considering GT systems as computing functions where we restrict [15]. Indeed, one might only be interested in the non-garbage critical pairs themselves, and classification of conflicts [20, 22].

Confluence analysis of GT systems (and related systems) still remains a generally under-explored area. One obvious piece of future work is to investigate the connection to the work by Lambers, Ehrig and Orejas on *essential critical pairs* [21] and the continued work by others including Born and Taentzer [20]. It is also not obvious if there is a relation between confluence up to garbage and graphs satisfying negative constraints [19]. Moreover, developing a stronger version of the Generalised Critical Pair Lemma that allows for the detection of persistent nodes that need not be identified in the joined graph would allow conclusions of confluence up to garbage where it was previously not determined.

Future work also includes developing checkable sufficient conditions under which one can decide if a graph is in the subgraph closure of a language. Finally, applying our theory in a rooted context and to GP 2 is future work [2]. It is likely that the theory will be applicable there, since program preconditions correspond exactly to non-garbage input, and so it is only natural to be interested in confluence up to garbage, rather than confluence. We would also expect there to be analogues of our results for other kinds of rewriting systems such as string and term rewriting.

References

1. Baader, F., Nipkow, T.: *Term Rewriting and All That*. Cambridge University Press (1998)
2. Bak, C.: *GP 2: Efficient Implementation of a Graph Programming Language*. Ph.D. thesis, Department of Computer Science, University of York, UK (2015), <https://etheses.whiterose.ac.uk/12586/>
3. Bakewell, A., Plump, D., Runciman, C.: Specifying pointer structures by graph reduction. In: *Proc. Second International Workshop on Applications of Graph Transformations with Industrial Relevance (AGTIVE 2003)*. Lecture Notes in Computer Science, vol. 3062, pp. 30–44. Springer (2004). https://doi.org/10.1007/978-3-540-25959-6_3
4. Berstel, J.: *Transductions and Context-Free Languages*. Vieweg+Teubner (1979). <https://doi.org/10.1007/978-3-663-09367-1>
5. Campbell, G.: *Efficient Graph Rewriting*. BSc thesis, Department of Computer Science, University of York, UK (2019), <https://arxiv.org/abs/1906.05170>
6. Campbell, G., Plump, D.: *Efficient recognition of graph languages*. Tech. rep., Department of Computer Science, University of York, UK (2019), <https://arxiv.org/abs/1911.12884>
7. Caron, A.C.: Linear bounded automata and rewrite systems: Influence of initial configurations on decision properties. In: *Proc. International Joint Conference on Theory and Practice of Software Development (TAPSOFT '91)*. CAAP 1991. Lecture Notes in Computer Science, vol. 493, pp. 74–89. Springer (1991). https://doi.org/10.1007/3-540-53982-4_5
8. Courcelle, B.: The monadic second-order logic of graphs: Definable sets of finite graphs. In: *Proc. 14th International Workshop on Graph-Theoretic Concepts in Computer Science (WG '88)*. Lecture Notes in Computer Science, vol. 344, pp. 30–53. Springer (1989). https://doi.org/10.1007/3-540-50728-0_34
9. Duffin, R.J.: Topology of series-parallel networks. *Journal of Mathematical Analysis and Applications* **10**(2), 303–318 (1965). [https://doi.org/10.1016/0022-247X\(65\)90125-3](https://doi.org/10.1016/0022-247X(65)90125-3)
10. Ehrig, H., Ehrig, K., Prange, U., Taentzer, G.: *Fundamentals of Algebraic Graph Transformation*. Monographs in Theoretical Computer Science. An EATCS Series, Springer (2006). <https://doi.org/10.1007/3-540-31188-2>
11. Ehrig, H., Kreowski, H.J.: Parallelism of manipulations in multidimensional information structures. In: *Proc. 5th Symposium on Mathematical Foundations of Computer Science (MFCS 1976)*. Lecture Notes in Computer Science, vol. 45, pp. 284–293. Springer (1976). https://doi.org/10.1007/3-540-07854-1_188
12. Ehrig, H., Pfender, M., Schneider, H.: Graph-grammars: An algebraic approach. In: *Proc. 14th Annual Symposium on Switching and Automata Theory (SWAT 1973)*. pp. 167–180. IEEE (1973). <https://doi.org/10.1109/SWAT.1973.11>
13. Farrow, R., Kennedy, K., Zucconi, L.: Graph grammars and global program data flow analysis. In: *Proc. 17th Annual Symposium on Foundations of Computer Science (SFCS 1976)*. pp. 42–56. IEEE (1976). <https://doi.org/10.1109/SFCS.1976.17>
14. Fradet, P., Métayer, D.L.: Structured Gamma. *Science of Computer Programming* **31**(2–3), 263–289 (1998). [https://doi.org/10.1016/S0167-6423\(97\)00023-3](https://doi.org/10.1016/S0167-6423(97)00023-3)
15. 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>

16. Hristakiev, I., Plump, D.: Checking graph programs for confluence. In: Software Technologies: Applications and Foundations – STAF 2017 Collocated Workshops, Revised Selected Papers. Lecture Notes in Computer Science, vol. 10748, pp. 92–108. Springer (2018). https://doi.org/10.1007/978-3-319-74730-9_8
17. Huet, G.: Confluent reductions: Abstract properties and applications to term rewriting systems. *Journal of the ACM* **27**(4), 797–821 (1980). <https://doi.org/10.1145/322217.322230>
18. Knuth, D., Bendix, P.: Simple word problems in universal algebras. In: Computational Problems in Abstract Algebras. pp. 263–297. Pergamon Press (1970). <https://doi.org/10.1016/B978-0-08-012975-4.50028-X>
19. Lambers, L.: Certifying Rule-Based Models using Graph Transformation. Ph.D. thesis, Technical University of Berlin, Elektrotechnik und Informatik (2009), <https://dx.doi.org/10.14279/depositonce-2348>
20. Lambers, L., Born, K., Orejas, F., Strüber, D., Taentzer, G.: Initial Conflicts and Dependencies: Critical Pairs Revisited, Lecture Notes in Computer Science, vol. 10800, pp. 105–123. Springer (2018). https://doi.org/10.1007/978-3-319-75396-6_6
21. Lambers, L., Ehrig, H., Orejas, F.: Efficient conflict detection in graph transformation systems by essential critical pairs. In: Proc. Fifth International Workshop on Graph Transformation and Visual Modeling Techniques (GT-VMT 2006). Electronic Notes in Theoretical Computer Science, vol. 211, pp. 17–26. Elsevier (2008). <https://doi.org/10.1016/j.entcs.2008.04.026>
22. Lambers, L., Kosiol, J., Strüber, D., Taentzer, G.: Exploring conflict reasons for graph transformation systems. In: Proc. 12th International Conference on Graph Transformation (ICGT 2019). Lecture Notes in Computer Science, vol. 11629, pp. 75–92. Springer (2019). https://doi.org/10.1007/978-3-030-23611-3_5
23. Lawler, E.: Sequencing jobs to minimize total weighted completion time subject to precedence constraints. *Annals of Discrete Mathematics* **2**, 75–90 (1978). [https://doi.org/10.1016/S0167-5060\(08\)70323-6](https://doi.org/10.1016/S0167-5060(08)70323-6)
24. Monma, C., Sidney, J.: Sequencing with series-parallel precedence constraints. *Mathematics of Operations Research* **4**(3), 215–224 (1979). <https://doi.org/10.1287/moor.4.3.215>
25. Newman, M.: On theories with a combinatorial definition of "equivalence". *Annals of Mathematics* **43**(2), 223–243 (1942). <https://doi.org/10.2307/1968867>
26. Plump, D.: Hypergraph rewriting: Critical pairs and undecidability of confluence. In: Term Graph Rewriting. pp. 201–213. John Wiley and Sons (1993)
27. Plump, D.: Computing by Graph Rewriting. Habilitation thesis, Universität Bremen, Fachbereich Mathematik und Informatik (1999)
28. Plump, D.: Confluence of Graph Transformation Revisited, Lecture Notes in Computer Science, vol. 3838, pp. 280–308. Springer (2005). https://doi.org/10.1007/11601548_16
29. Plump, D.: Reasoning about graph programs. In: Proc. 9th International Workshop on Computing with Terms and Graphs (TERMGRAPH 2016). Electronic Proceedings in Theoretical Computer Science, vol. 225, pp. 35–44. Open Publishing Association (2016). <https://doi.org/10.4204/EPTCS.225.6>