Theoretical Computer Science ••• (••••) •••–•••

Contents lists available at ScienceDirect

Theoretical Computer Science

www.elsevier.com/locate/tcs

Confluence up to garbage in graph transformation

Graham Campbell [a,*,1], Detlef Plump [b]

[a] School of Mathematics, Statistics and Physics, Newcastle University, Newcastle upon Tyne, United Kingdom
[b] Department of Computer Science, University of York, York, United Kingdom

ARTICLE INFO

Article history:
Received 4 January 2021
Received in revised form 13 May 2021
Accepted 10 June 2021
Available online xxxx

Keywords:
Graph transformation
Confluence
Subcommutativity
Critical pair analysis
Graph languages

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 in two case studies about efficient language recognition: we present 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. We also give a critical pair condition for subcommutativity up to garbage which, together with closedness, implies confluence up to garbage even in non-terminating systems.

1. Introduction

Rule-based graph transformation and graph grammars date back to the late 1960s. The most developed theoretical framework is the so-called double-pushout (DPO) approach to graph transformation [1,2]. 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. For example, [3] contains 23 case studies of confluent and terminating graph reductions systems which specify pointer structures such as cyclic lists, balanced binary trees and red-black trees. Confluence is also important in the context of evaluating functional expressions by graph reduction, see for example [4].

However, there are application scenarios where the rules are not 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". An example is the class of so-called extended flow diagrams discussed in Subsection 5.3. The reduction rules for these graphs give rise to ten critical pairs, nine of which are strongly joinable. But a single pair is not joinable and hence the rules are not confluent. The non-joinable pair represents a conflict in graphs containing a type of cycle that cannot occur in extended flow

* Corresponding author.
E-mail addresses: g.j.campbell2@newcastle.ac.uk (G. Campbell), detlef.plump@york.ac.uk (D. Plump).

[1] 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.

https://doi.org/10.1016/j.tcs.2021.06.010

diagrams. Hence these graphs can be considered as garbage which in this case consists of all graphs that are not extended flow diagrams.

In this paper, we introduce the notions of confluence up to garbage and termination up to garbage in graph transformation. We generalise Plump's Critical Pair Lemma [5,6] and Newmann's Lemma [7] 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 labelled series-parallel graphs and a class of flow diagrams, respectively. Both systems are non-confluent but confluent up to garbage.

The rest of this paper is organised as follows. Section 2 reviews abstract reduction systems, graphs and morphisms, the double-pushout approach to graph transformation, graph languages, and confluence checking by critical pair analysis. In Section 3, confluence up to garbage and closedness properties are introduced, and a generalisation of Newman's Lemma is presented. We also compare confluence up to garbage with confluence modulo an equivalence relation. In Section 4, we generalise the Critical Pair Lemma to confluence up to garbage and discuss the generation of non-garbage critical pairs. This includes sufficient conditions for automating the generation process. Section 5 is devoted to the main application of confluence up to garbage, the recognition of graph languages by backtracking-free graph reduction. We present two case studies which illustrate this application. In Section 6, we show that subcommutativity up to garbage together with closedness implies confluence up to garbage, even in the absence of termination. Finally, in Section 7, we briefly mention related work, discuss potential generalisations and future work, and conclude.

This paper is an extended version of our ICGT 2020 paper [8], which was in turn partly developed from Campbell's BSc Thesis [9]. Throughout this paper, we provide more detail and examples than previously. We have included a new subsection on the relationship between confluence up to garbage and confluence modulo an equivalence relation. Section 5 has been revised with more succinct terminology and more details of the critical pair analyses included. Section 6 on subcommutativity up to garbage is entirely new. We give a second version of our generalised critical pair lemma in this setting, showing how critical pair analysis can be used to check for subcommutativity up to garbage. This property implies confluence up to garbage even in non-terminating systems, provided that non-garbage is closed under reduction. This is relevant for applications because confluence up to garbage in such systems implies that non-garbage graphs can be reduced to at most one irreducible graph.

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 $(\mathcal{A}, \to)$ where $\mathcal{A}$ is a *class* and $\to$ a *binary relation* on $\mathcal{A}$. Write $\xrightarrow{=}$ for the reflexive closure of $\to$, $\xrightarrow{+}$ for the transitive closure, and $\xrightarrow{*}$ for the reflexive transitive closure. Given $x, x_i, y, y_1, y_2 \in \mathcal{A}$ $(i \geq 0)$, we say that:

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

The principle of *Noetherian Induction* is:

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

Theorem 2.1 *(Noetherian induction [10]). Given an ARS $(\mathcal{A}, \to)$, the principle of Noetherian induction holds if and only if $\to$ is terminating.*

Lemma 2.2. *Subcommutativity implies confluence, and confluence implies local confluence.*

Theorem 2.3 *(Newman's Lemma [7]). Let $\to$ be a terminating relation. Then $\to$ is confluent if and only if it is locally confluent.*

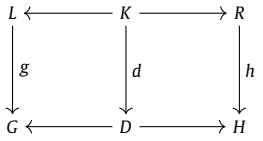


Fig. 1. A direct derivation.

2.2. Labelled graphs and morphisms

We will be working with *directed labelled graphs* [11]. A *signature* 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, $s\colon E \to V$ is the source function, $t\colon E \to V$ is the target function, $l\colon V \to \Sigma_V$ is the node labelling function, and $m\colon E \to \Sigma_E$ is the edge labelling function. We may write the components of G as V_G, E_G, s_G, t_G, l_G, and m_G.

A *graph morphism* $g\colon G \to H$ is a pair $g = (g_V, g_E)$ of functions $g_V\colon V_G \to V_H$ and $g_E\colon E_G \to E_H$ 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* $i\colon H \to G$ with $i(x) = x$ for all items x. Graphs G and H are *isomorphic*, denoted by $G \cong H$, if there exists a bijective graph morphism $G \to H$.

It is well known that graphs and morphisms over a fixed signature Σ form a category. Graph morphisms are injective (surjective, bijective) if and only if they are monomorphisms (epimorphisms, isomorphisms) in the categorical sense. We denote by $\mathcal{G}(\Sigma)$ the class of all 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 $L \to G$. An application of rule r to G with match $g\colon L \to G$ requires to construct two pushouts as in Fig. 1. We write $G \Rightarrow_{r,g} H$ for this application and call the diagram in Fig. 1 a *direct derivation*.

Given r and the match $g\colon L \to G$, the direct derivation of Fig. 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 [2]. 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 $\mathcal{R}$. Note that $\Rightarrow_{\mathcal{R}}$ is isomorphism-compatible in that $G' \cong G \Rightarrow_{\mathcal{R}} H \cong H'$ implies $G' \Rightarrow_{\mathcal{R}} H'$. 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 $G \Rightarrow^{+}_{\mathcal{R}} H$. We can view a graph transformation system as ARS $(\mathcal{G}(\Sigma), \Rightarrow_{\mathcal{R}})$, giving us the definition of local confluence, confluence, subcommutativity, and termination for graph transformation systems.

2.4. Graph languages

A *graph language* is an isomorphism-closed class of graphs, and the size of a graph language is defined to be the number of non-isomorphic graphs in the language. 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.

A *graph transformation system* $T = (\Sigma, \mathcal{R})$ consists of a signature and a finite set of rules with graphs over Σ. By adding a subsignature of *non-terminals* N and a *start graph* S over Σ, we obtain a *graph grammar* $\mathcal{G} = (\Sigma, N, \mathcal{R}, S)$. We say that a graph G is *terminally labelled* if $l(V) \cap N_V = \emptyset$ and $m(E) \cap N_E = \emptyset$. Thus, we can define the *graph language* generated by $\mathcal{G}$:

$$\mathrm{L}(\mathcal{G}) = \{G \mid S \Rightarrow^{*}_{\mathcal{R}} G, G \text{ terminally labelled}\}.$$

Given $\mathcal{G} = (\Sigma, N, \mathcal{R}, S)$, 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, $G \in \mathrm{L}(\mathcal{G})$ if and only if $G \Rightarrow^{*}_{\mathcal{R}^{-1}} S$ and G is terminally labelled. So we have a non-deterministic membership checking algorithm, by running the rules in reverse.

2.5. Confluence checking

In 1970, Knuth and Bendix showed that confluence checking of terminating term rewriting systems is decidable [12]. Moreover, it suffices to compute all *critical pairs* and check their joinability [13,10]. 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 [5,6].

In order to define critical pairs and critical pair isomorphism, we first must define what we mean by an instance of a derivation based on a morphism and what it means for two derivations to be parallelly independent.

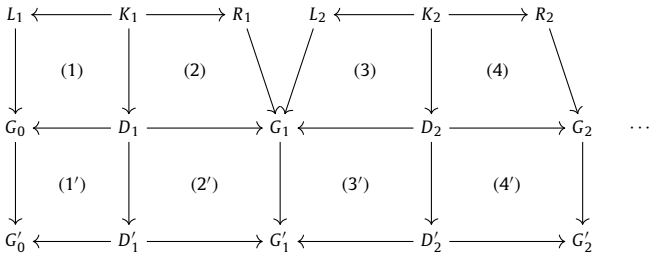


Fig. 2. Derivation instances.

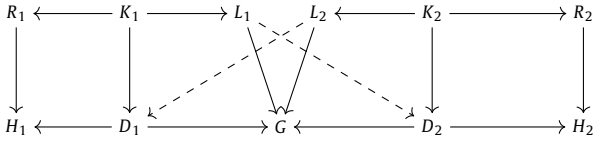


Fig. 3. Parallelly independent direct derivations.

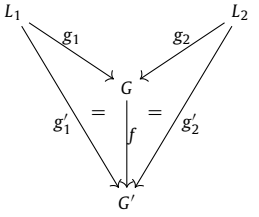


Fig. 4. Isomorphism of critical pairs.

Let the derivation $\Delta : G_0 \Rightarrow^* G_n$ be given by pushouts $(1), \ldots, (n)$ and suppose there are pushouts $(1'), \ldots, (n')$ whose vertical morphisms are injective (Fig. 2). Then, the derivation $\Delta' : G'_0 \Rightarrow^* G'_n$ consisting of the composed pushouts $(1 + 1'), \ldots, (n + n')$ is an *instance* of Δ based on the morphism $G_0 \to G'_0$. Moreover, we define the subgraph Use_Δ to be all items x such that there is some $i \geq 0$ with $G_0 \Rightarrow^* G_i(x) \in \mathrm{Match}(G_i \Rightarrow G_{i+1})$ where $\mathrm{Match}(G_i \Rightarrow G_{i+1})$ is the image of the associated rule's left hand side graph under the match $L \to G_i$.

We say two direct derivations $H_1 \Leftarrow_{r_1,g_1} G \Rightarrow_{r_2,g_2} H_2$ are *parallelly independent* if $(g_1(L_1) \cap g_2(L_2)) \subseteq (g_1(K_1) \cap g_2(K_2))$, or equivalently, if there are morphisms $L_1 \to D_2$ and $L_2 \to D_1$ such that $L_1 \to D_2 \to G = L_1 \to G$ and $L_2 \to D_1 \to G = L_2 \to G$ (Fig. 3). Otherwise, we say they are *parallelly dependent*.

We say two *parallelly dependent* direct derivations $H_1 \Leftarrow_{r_1,g_1} G \Rightarrow_{r_2,g_2} H_2$ are a *critical pair* if additionally $G = g_1(L_1) \cup g_2(L_2)$, and if $r_1 = r_2$ then $g_1 \neq g_2$. We call two critical pairs $H_1 \Leftarrow_{r_1,g_1} G \Rightarrow_{r_2,g_2} H_2$ and $H'_1 \Leftarrow_{r_1,g'_1} G' \Rightarrow_{r_2,g'_2} H'_2$ *isomorphic* if there is a isomorphism $f : G \to G'$ such that $G' \Rightarrow H'_1$ is an instance of $G \Rightarrow H_1$ based on f and $G' \Rightarrow H'_2$ is an instance of $G \Rightarrow H_2$ based on f. Equivalently, the critical pairs $H_1 \Leftarrow_{r_1,g_1} G \Rightarrow_{r_2,g_2} H_2$ and $H'_1 \Leftarrow_{r_1,g'_1} G' \Rightarrow_{r_2,g'_2} H'_2$ are isomorphic if there is an isomorphism $f : G \to G'$ such that $g'_1 = f \circ g_1$ and $g'_2 = f \circ g_2$ (Fig. 4). It is easy to see that every graph transformation system has, up to isomorphism, only finitely many critical pairs.

The *track morphism* of a direct derivation $G \Rightarrow H$ 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 Fig. 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* (*strongly subcommutative*) if it is *joinable* (*subcommutative*) without deleting any of the persistent nodes, and the persistent nodes are identified when joining. That is, there exists a graph M and derivations $H_1 \Rightarrow^*_{\mathcal{R}} M \Leftarrow^*_{\mathcal{R}} H_2$ ($H_1 \Rightarrow^=_{\mathcal{R}} M \Leftarrow^=_{\mathcal{R}} H_2$) such that $\forall v \in Persist_\Phi, tr_{G \Rightarrow H_1 \Rightarrow^* M}(\{v\}) = tr_{G \Rightarrow H_2 \Rightarrow^* M}(\{v\}) \neq \emptyset$.

Example 2.4. Consider the rules s_2 and s_4 from Example 3.8. Fig. 5 shows a strongly joinable critical pair and its joining derivations.

Theorem 2.5 *(Critical pair lemma [5,6]). A graph transformation system T is locally confluent if all its critical pairs are strongly joinable.*

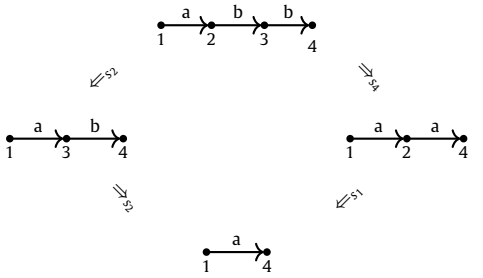


Fig. 5. Example critical pair and joining derivations.

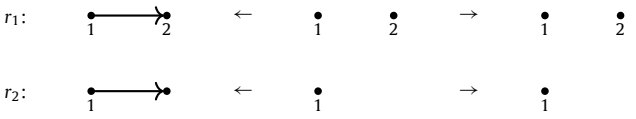


Fig. 6. Reduction rules for Example 3.2.

It's easy to see that the result also holds if one only considers non-isomorphic critical pairs, which can result in a large speedup in practice, since for large rules, there can often be many isomorphic critical pairs.

The original proof of the Critical Pair Lemma needs the Commutativity, Clipping and Embedding Theorems, which we shall now provide, and use in the proof of our Generalised Critical Pair Lemma (Theorem 4.8).

Theorem 2.6 *(Commutativity [14]). If* $H_1 \Leftarrow_{r_1,g_1} G \Rightarrow_{r_2,g_2} H_2$ *are parallelly independent, then there is a graph* G' *and derivations* $H_1 \Rightarrow_{r_2} G' \Leftarrow_{r_1} H_2$.

Theorem 2.7 *(Clipping [15]). Given a derivation* $\Delta' : G' \Rightarrow^* H'$ *and an injective morphism* $h : G \to G'$ *such that* $\mathrm{Use}\Delta' \subseteq h(G)$*, 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, $\mathrm{Persist}_\Delta$, consists of all items x such that $tr_{G \Rightarrow^* H}(x)$ is defined.

Theorem 2.8 *(Embedding [15]). Let* $\Delta : G \Rightarrow^* H$ *be a derivation,* $h : G \to G'$ *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* $G' - h(G)$*. If* $B_\Delta \subseteq \mathrm{Persist}_\Delta$*, 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* $t : B_\Delta \to H$ *along* $h' : B_\Delta \to C_\Delta$ *where* $C_\Delta = (G' - h(G)) \cup h(B_\Delta)$ *and* t *is the restriction of* $tr_{G \Rightarrow^* H}$ *to* B_Δ.

3. Closedness and confluence up to garbage

The purpose of this section is to introduce the notion of "up to garbage" and lay some foundations that we can use in the remainder of the paper. We divide the section into three subsections, finishing by relating "up to garbage" with the existing notion of "modulo an equivalence relation".

3.1. Closedness and garbage

We start with the definition of closedness, and what it means for an item to be considered garbage.

Definition 3.1. Let $T = (\mathcal{A}, \to)$ be an ARS and $\mathcal{D} \subseteq \mathcal{A}$. Then an object $x \in \mathcal{A}$ is called *garbage* if $x \notin \mathcal{D}$ and $\mathcal{D}$ is *closed* under T if for all $x, y \in \mathcal{A}$ such that $x \to y$, if $x \in \mathcal{D}$ then $y \in \mathcal{D}$.

The idea is that $\mathcal{D}$ represents the *good input*, and the *garbage* is the objects that are not in this class. In the context of graph transformation, $\mathcal{D}$ will be a graph language, but need not be explicitly generated by a graph grammar. For example, it could be defined by some (monadic second-order [16]) logical formula, a finite listing of graphs, or a type graph language (Subsection 4.3). Finite languages and type graph languages will be of particular interest to us due to the fact they have decidable subgraph membership problem (Subsection 4.3).

Example 3.2. Consider the reduction rules in Fig. 6. 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 $((\{\square\}, \{\square\}), \{r_2\})$.

The closedness problem is defined in the obvious way:

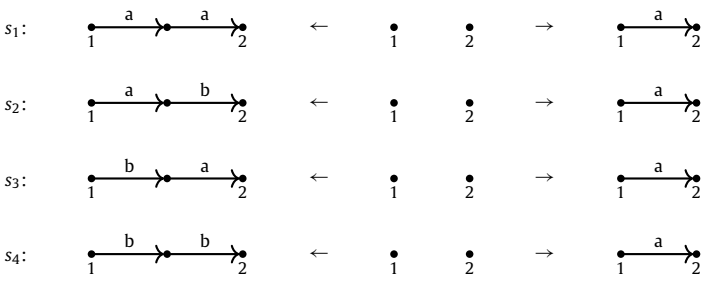


Fig. 7. Reduction rules for Example 3.8.

Definition 3.3 *(Closedness problem).*

Instance: A GT system $T = (\Sigma, \mathcal{R})$ and a graph grammar $\mathcal{G}$ over Σ.
Question: Is $\mathrm{L}(\mathcal{G})$ closed under T?

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

Theorem 3.4 *(Undecidable closedness [17]). 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 and subcommutativity up to garbage

In this subsection, we generalise the familiar definitions of local confluence, confluence, subcommutativity, and termination to permit ignoring *garbage*.

Definition 3.5. Given an ARS $(\mathcal{A}, \to)$, $\mathcal{D} \subseteq \mathcal{A}$, $x, x_0 \in \mathcal{D}$, and $x_i, y_1, y_2 \in \mathcal{A}$ $(i \geq 1)$, we say that:

1. $\to$ is *confluent up to garbage* on $\mathcal{D}$ if $y_1 \overset{*}{\leftarrow} x \overset{*}{\to} y_2$ implies y_1, y_2 are joinable;
2. $\to$ is *locally confluent up to garbage* on $\mathcal{D}$ if $y_1 \leftarrow x \to y_2$ implies y_1, y_2 are joinable;
3. $\to$ is *subcommutative up to garbage* on $\mathcal{D}$ if $y_1 \leftarrow x \to y_2$ implies y_1, y_2 are subcommutative;
4. $\to$ is *terminating up to garbage* on $\mathcal{D}$ if there is no infinite sequence $x_0 \to x_1 \to \ldots$.

The following is an immediate consequence of inclusion:

Lemma 3.6. *Let $(\mathcal{A}, \to)$ be an ARS, $\mathcal{D} \subseteq \mathcal{A}$, $\mathcal{E} \subseteq \mathcal{D}$, and $\mathcal{P}$ be the property of confluence up to garbage, local confluence up to garbage, subcommutativity up to garbage, or termination up to garbage. Then $\mathcal{P}$ on $\mathcal{D}$ implies $\mathcal{P}$ on $\mathcal{E}$.*

Our next two examples show that confluence up to garbage need not correspond to confluence. That is, a system can be non-confluent, but confluent up to garbage.

Example 3.7. Consider again the rules in Fig. 6. It is easy to see that the GT system containing both rules is terminating, but not confluent. It is, however, both confluent up to garbage on the language of unlabelled discrete graphs and subcommutative up to garbage on the language of unlabelled discrete graphs.

Example 3.8. Consider the rules in Fig. 7. They are terminating, since they are size reducing. Moreover, the language of all linked lists with edge labels a or b and its complement (over the same signature) are closed under the rules. These rules are confluent up to garbage on linked lists, since any non-trivial linked list is necessarily reduced to the length one linked list labelled by a, and the length zero and one linked lists are already in normal form. By comparison, the rules are not locally confluent due to the counterexample in Fig. 8 (recall that we assume injective matching).

It is easy to see that confluence up to garbage always implies local confluence up to garbage, and subcommutativity up to garbage implies local confluence up to garbage, however subcommutativity up to garbage need not imply confluence up to garbage. Similarly, in the presence of termination, local confluence up to garbage need not imply confluence up to garbage. Our next example demonstrates this.

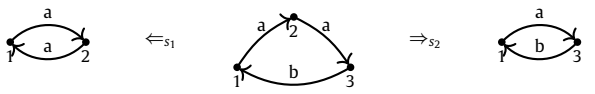


Fig. 8. Non-joinable derivations for Example 3.8.

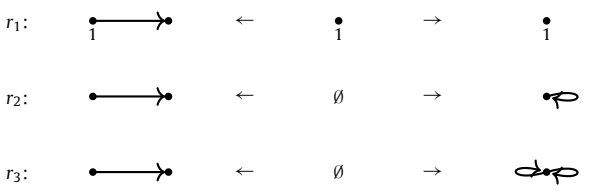


Fig. 9. Rules for Example 3.9.

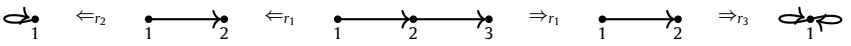


Fig. 10. Non-joinable derivations for Example 3.9.

Example 3.9. Let $\mathcal{D}$ be the language of linked lists containing at least two edges and T be the GT system with rules from Fig. 9. Then r_2 and r_3 cannot be applied to any graph in $\mathcal{D}$, and r_1 will always be applicable in a unique way, with the effect of deleting the last node and its edge. It is thus immediate that T is subcommutative up to garbage on $\mathcal{D}$, and thus also locally confluent up to garbage on $\mathcal{D}$. T is not, however, confluent up to garbage on $\mathcal{D}$ due to the following two-step counterexample in Fig. 10.

Our next two results show that closedness is the missing ingredient to recovering the familiar relationships between local confluence, confluence, subcommutativity, and termination, in our generalised setting of "up to garbage".

Lemma 3.10. *Let $(\mathcal{A}, \to)$ be an ARS and $\mathcal{D} \subseteq \mathcal{A}$.*

1. *If $\mathcal{D}$ is closed under $\to$ and $\to$ is subcommutative up to garbage on $\mathcal{D}$, then $\to$ is confluent up to garbage on $\mathcal{D}$;*
2. *If $\to$ is confluent up to garbage on $\mathcal{D}$, then $\to$ is locally confluent up to garbage on $\mathcal{D}$.*

Proof. The first part can be seen by Noetherian Induction, due to the fact that closedness ensures applicability of the induction hypothesis, and the second part follows immediately from the definitions. □

Theorem 3.11 *(Generalised Newman's Lemma). Let $(\mathcal{A}, \to)$ be an ARS and $\mathcal{D} \subseteq \mathcal{A}$. If $\to$ is terminating up to garbage on $\mathcal{D}$ and $\mathcal{D}$ is closed under $\to$, then $\to$ is confluent up to garbage on $\mathcal{D}$ if and only if $\to$ it is locally confluent up to garbage on $\mathcal{D}$.*

Proof. One direction can be seen by Noetherian Induction (Fig. 11), since closedness ensures applicability of the induction hypothesis, and the other follows from the second part of Lemma 3.10. □

3.3. Confluence and subcommutativity modulo garbage

In this subsection, we show that our notion of confluence up to garbage can be related to the existing notion of confluence modulo.

First, we recall the definition of local confluence (confluence, subcommutativity) modulo an equivalence relation. If the relation is the identity relation, then we recover the standard definitions of local confluence (confluence, subcommutativity).

Definition 3.12 *([13]).* Given an ARS $(\mathcal{A}, \to)$, an equivalence $\sim$ on $\mathcal{A}$, and $x, x_i, y, y_1, y_2 \in \mathcal{A}$ $(i \geq 0)$, we say that:

1. x and y are *$\sim$-joinable* if there is are $z_1, z_2 \in \mathcal{A}$ such that $x \overset{*}{\to} z_1 \sim z_2 \overset{*}{\leftarrow} y$;
2. x and y are *$\sim$-subcommutative* if there are $z_1, z_2 \in \mathcal{A}$ such that $x \overset{=}{\to} z_1 \sim z_2 \overset{=}{\leftarrow} y$;
3. $\to$ is *confluent modulo* $\sim$ if $y_1 \overset{*}{\leftarrow} x \overset{*}{\to} y_2$ implies y_1, y_2 are $\sim$-joinable;
4. $\to$ is *locally confluent modulo* $\sim$ if $y_1 \leftarrow x \to y_2$ implies y_1, y_2 are $\sim$-joinable;
5. $\to$ is *subcommutative modulo* $\sim$ if $y_1 \leftarrow x \to y_2$ implies y_1, y_2 are $\sim$-subcommutative.

We now show that confluence up to garbage can be encoded as confluence modulo an equivalence, and vice versa.

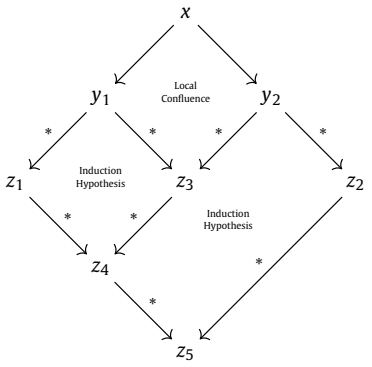


Fig. 11. Diagram for the proof of Theorem 3.11.

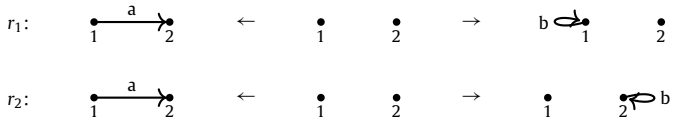


Fig. 12. Rules for Example 4.1.

Theorem 3.13 *(Encoding confluence up to garbage). Let $(\mathcal{A}, \to)$ be an ARS, $\mathcal{D} \subseteq \mathcal{A}$, and define the equivalence $\sim$ on $\mathcal{A}$ by $x \sim y$ exactly when $x = y$ or $x, y \in \mathcal{A} - \mathcal{D}$. Then:*

1. *if $\mathcal{A} - \mathcal{D}$ is closed under $\to$ and $\to$ is $\mathcal{P}$ up to garbage on $\mathcal{D}$, then $\to$ is $\mathcal{P}$ modulo $\sim$;*
2. *if $\mathcal{D}$ is closed under $\to$ and $\to$ is $\mathcal{P}$ modulo $\sim$, then $\to$ is $\mathcal{P}$ up to garbage on $\mathcal{D}$,*

where $\mathcal{P}$ is the property confluence, local confluence, or subcommutativity.

Proof. We deal only with confluence. Local confluence and subcommutativity are trivial modifications of the same argument.

Suppose $\mathcal{A} - \mathcal{D}$ is closed under $\to$ and $\to$ is confluent up to garbage on $\mathcal{D}$. Then, for all derivations $x \xrightarrow{*} y$, $x \xrightarrow{*} z$ such that $x \in \mathcal{D}$, there is a $t \in \mathcal{A}$ such that $y \xrightarrow{*} t$ and $z \xrightarrow{*} t$. Clearly $t \sim t$, so all such derivations are $\sim$-joinable. Suppose now that $x \notin \mathcal{D}$. Then by closedness, $y, z \notin \mathcal{D}$ too, so $y \sim z$, so all such derivations are $\sim$-joinable. Thus $\to$ is confluent modulo $\sim$, as required.

Suppose $\mathcal{D}$ is closed under $\to$ and $\to$ is confluent modulo $\sim$. Then, for all derivations $x \xrightarrow{*} y$, $x \xrightarrow{*} z$ such that $x \in \mathcal{D}$, there are $t_1, t_2 \in \mathcal{A}$ such that $y \xrightarrow{*} t_1$, $z \xrightarrow{*} t_2$, and $t_1 \sim t_2$. Due to closedness, we have $y, t_1, z, t_2 \in \mathcal{D}$. Putting this together with the fact that $t_1 \sim t_2$ tells us that in fact $t_1 = t_2$. Thus all such derivations are joinable. Finally, if $x \notin \mathcal{D}$, we don't need to consider joinability. Thus, $\to$ is confluent up to garbage on $\mathcal{D}$, as required. □

Thus, if both $\mathcal{A} - \mathcal{D}$ and $\mathcal{D}$ are closed under $\to$, then the notions of confluence (local confluence, subcommutativity) up to garbage and modulo garbage exactly correspond:

Corollary 3.14. *Let $(\mathcal{A}, \to)$ be an ARS, $\mathcal{D} \subseteq \mathcal{A}$, and define the equivalence $\sim$ on $\mathcal{A}$ by $x \sim y$ exactly when $x = y$ or $x, y \in \mathcal{A} - \mathcal{D}$. If $\mathcal{A} - \mathcal{D}$ and $\mathcal{D}$ are closed under $\to$, then $\to$ is $\mathcal{P}$ up to garbage on $\mathcal{D}$ if and only if $\to$ is $\mathcal{P}$ modulo $\sim$, where $\mathcal{P}$ is the property confluence, local confluence, or subcommutativity.*

4. Generalised critical pair lemma

Recall that strong joinability of all critical pairs is a sufficient condition for local confluence (Theorem 2.5). This was first shown by Plump in 1993 [5]. Combining this with Newman's Lemma (Theorem 2.3), we have a checkable condition for confluence of a GT system. Unlike for string and term rewriting, joinability is not sufficient to show local confluence, as demonstrated by the following example due to Plump [6]:

Example 4.1. Consider the terminating GT system with the rules from Fig. 12. The only critical pair (Fig. 13) is joinable, but not strongly, and the system is not locally confluent due to the counterexample in Fig. 14.

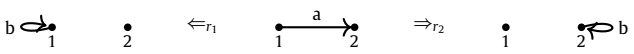


Fig. 13. Critical pair for Example 4.1.

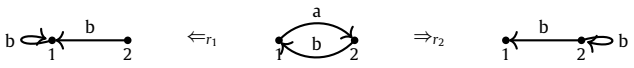


Fig. 14. Non-joinable instance of Fig. 13.

In this section, we generalise Plump's critical pair analysis for confluence up to garbage. We delay the treatment of subcommutativity up to garbage to Section 6, for ease of reading.

4.1. Subgraph closure and subgraph closed languages

In the original proof of the Critical Pair Lemma for (hyper)graphs [5], 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. We start this subsection by defining subgraph closure.

Definition 4.2. Let $\mathcal{D} \subseteq \mathcal{G}(\Sigma)$ be a language over some signature Σ. 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 language (with respect to inclusion) containing $\mathcal{D}$ that is *subgraph closed*.

Lemma 4.3. *Given a language $\mathcal{D} \subseteq \mathcal{G}(\Sigma)$, $\widehat{\mathcal{D}}$ 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 $\mathcal{D}$. □

$\widehat{\mathcal{D}}$ always exists, however it needs not be decidable, even when $\mathcal{D}$ is! It is not obvious what conditions on $\mathcal{D}$ ensure that $\widehat{\mathcal{D}}$ is decidable. If we move to the setting of string rewriting, there are some known cases where this can be solved. The classes of regular and context-free string languages are closed under substring closure, and the substring membership problem is decidable for context-free grammars [18] due to the fact that showing closure is constructive. We return to the issue of deciding subgraph membership in Subsection 4.3.

Example 4.4. The following graph languages are subgraph closed, over any signature Σ:

1. the empty language $\emptyset$ and the language of all graphs $\mathcal{G}(\Sigma)$;
2. the language of discrete graphs;
3. the language of acyclic graphs;
4. the language of planar graphs;
5. the language of k-colourable graphs for any fixed $k \geq 2$;
6. the language of bounded degree graphs for any fixed bound;
7. the language of bounded treewidth graphs for any fixed bound.

Example 4.5. 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. Generalising the critical pair lemma

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 4.6. Let $T = (\Sigma, \mathcal{R})$ be a GT system and $\mathcal{D} \subseteq \mathcal{G}(\Sigma)$ a language. A *critical pair* $H_1 \Leftarrow G \Rightarrow H_2$ is *$\mathcal{D}$-non-garbage* if $G \in \widehat{\mathcal{D}}$.

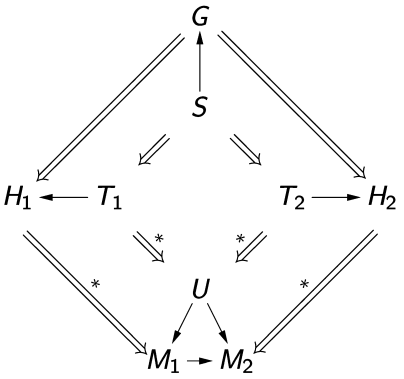


Fig. 15. Diagram for the proof of Theorem 4.8.

Lemma 4.7. *Given a GT system $T = (\Sigma, \mathcal{R})$ and a language $\mathcal{D} \subseteq \mathcal{G}(\Sigma)$, then there are only finitely many $\mathcal{D}$-non-garbage critical pairs (up to isomorphism).*

Proof. Recall from Subsection 2.5 that any GT system has only finitely many critical pairs. Filtering out those that are garbage or isomorphic is certainly only going to leave us with a finite number of critical pairs. □

Of course, just because there are only finitely many non-garbage critical pairs, it doesn't mean that generation of them is effective, in general. In order to avoid interrupting the flow, we will discuss this further in Subsection 4.3. We now proceed to present the main result:

Theorem 4.8 *(Generalised critical pair lemma). Let $T = (\Sigma, \mathcal{R})$ be a GT system and $\mathcal{D} \subseteq \mathcal{G}(\Sigma)$ a language. If all T's $\mathcal{D}$-non-garbage critical pairs are strongly joinable, then T is locally confluent up to garbage on $\mathcal{D}$.*

Proof. Our proof is a generalisation of Plump's original proof of the Critical Pair Lemma for (hyper)graph transformation systems (Theorem 2.5). 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 2.6, the result is immediate. Otherwise, we must consider the case that they are not parallelly independent.

By Theorem 2.7, 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 2.8 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) (Fig. 15). To see that M_1 and M_2 are isomorphic follows from elementary properties of pushouts along monomorphisms [6]. □

Just as with the original Critical Pair Lemma in Subsection 2.5, it is sufficient to only check the non-isomorphic critical pairs for strong joinability due to the fact that derivations based on a critical pair can be reset as derivations based on any other isomorphic critical pair simply by passing through the isomorphism. Thus, if all T's non-isomorphic $\mathcal{D}$-non-garbage critical pairs are strongly joinable, then T is locally confluent up to garbage on $\mathcal{D}$.

The most common use case of this generalised critical pair will be the following corollary, where the aim is not to show local confluence up to garbage, but confluence up to garbage, given termination up to garbage:

Corollary 4.9. *Let $T = (\Sigma, \mathcal{R})$ be a GT system and $\mathcal{D} \subseteq \mathcal{G}(\Sigma)$ a language. If $\mathcal{D}$ is closed under T, T is terminating up to garbage on $\mathcal{D}$, and all T's non-isomorphic $\mathcal{D}$-non-garbage critical pairs are strongly joinable, then T is confluent up to garbage on $\mathcal{D}$.*

Proof. By the above theorem, T is locally confluent up to garbage, so by the Generalised Newman's Lemma (Theorem 3.11), T is confluent up to garbage. □

Example 4.10. Recall from Subsection 3.2, the non-confluent GT system from Example 3.8 (Fig. 7). We can use Corollary 4.9 to show that this system is confluent up to garbage on the language of acyclic graphs with edge labels a and b, $\mathcal{D}$. First, we observe that the rules (Example 3.8) are terminating, and that $\mathcal{D}$ is closed under the rules. Next, we observe that the system has 16 non-isomorphic critical pairs. (Here and in later examples we used a tool of the first author to find all critical

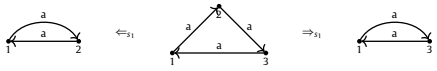
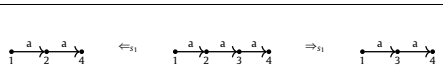
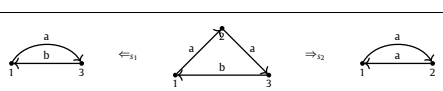
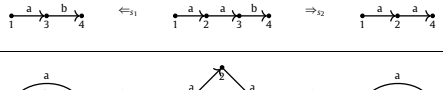
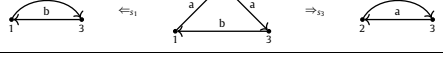
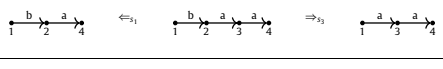
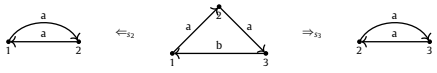
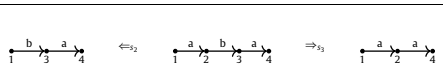
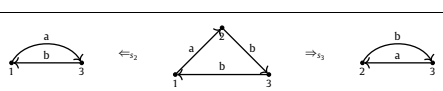
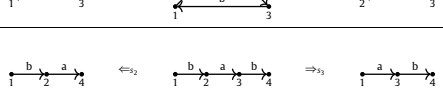
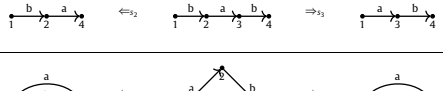
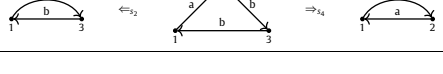
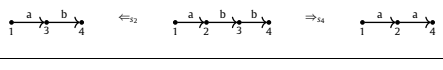
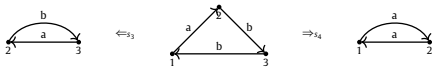
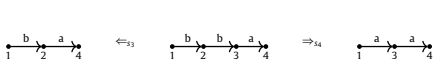
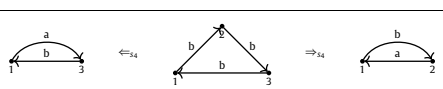
Pair/Property	Joinable	Strongly Joinable	Non-Garbage
	✓	✓	✗
	✓	✓	✓
	✗	✗	✗
	✓	✓	✓
	✗	✗	✗
	✓	✓	✓
	✓	✓	✗
	✓	✓	✓
	✓	✗	✗
	✓	✓	✓
	✗	✗	✗
	✓	✓	✓
	✗	✗	✗
	✓	✗	✗
	✓	✓	✓
	✓	✗	✗
	✓	✓	✓

Fig. 16. Critical pair analysis for Example 4.10.

pairs.[2]) Fig. 16 shows, for each of the pairs, if they are joinable, strongly joinable, or $\mathcal{D}$-non-garbage. From this, we can see that every non-garbage critical pair is strongly joinable, and so the system is confluent up to garbage on $\mathcal{D}$.

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. Moreover, local confluence up to garbage is actually undecidable in general for a terminating non-length-increasing string rewriting system and $\mathcal{D}$ a regular string language [19]. The following (corrected) example due to

[2] https://gitlab.com/YorkCS/Hyperspeed/GT-Tool.

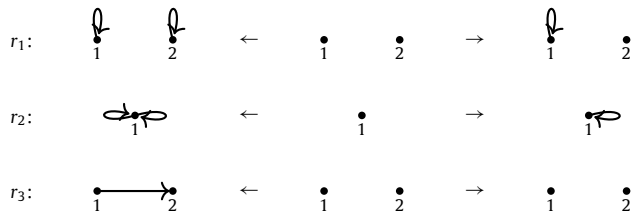


Fig. 17. Rules for Example 4.11.

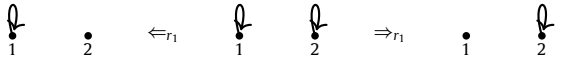


Fig. 18. Non-strongly joinable pair for Example 4.11.

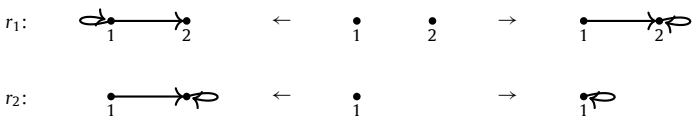


Fig. 19. Rules for Example 4.12.

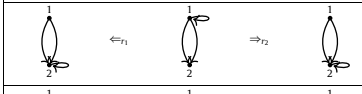
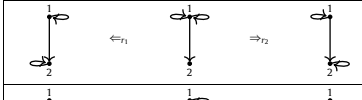
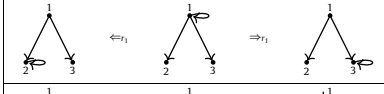
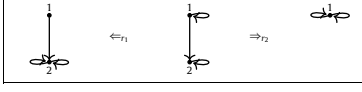
Pair/Property	Joinable	Strongly Joinable	Non-Garbage
	✓	✓	✗
	✓	✓	✗
	✓	✗	✓
	✗	✗	✗

Fig. 20. Critical pair analysis for Example 4.12.

Plump [6] demonstrates that a GT system can be confluent and terminating, with all critical pairs joinable, and at least one not strongly joinable:

Example 4.11. The GT system with rules in Fig. 17 is terminating and confluent, and all its critical pairs are strongly joinable apart from the pair in Fig. 18 which is only joinable.

We now extend this even further, showing that there is a GT system T that is not confluent and an infinite language of graphs $\mathcal{D}$ such that $\mathcal{D}$ is closed under T, T is terminating on $\mathcal{D}$, T is confluent on $\mathcal{D}$, all T's $\mathcal{D}$-non-garbage critical pairs are joinable, and at least one of them is not strongly joinable:

Example 4.12. Let $\mathcal{D}$ be the language of all graphs that are trees with exactly one looped edge added to one of the nodes, and T be the GT system with rules in Fig. 19. Fig. 20 shows the four non-isomorphic critical pairs of the system. We can see there is a garbage pair which is non-joinable, which tells us that T is not locally confluent. By direct argument, one can see that $\mathcal{D}$ is closed under T, T is terminating on $\mathcal{D}$, and T is confluent on $\mathcal{D}$, however there is a non-strongly joinable non-garbage critical pair.

In Subsection 4.3, we will discuss generation of the set of non-isomorphic non-garbage critical pairs of a given GT system, discussing sufficient conditions on $\mathcal{D}$ for this process to be effective. In Subsection 4.4 we discuss testing for strong joinability of a given non-garbage critical pair.

Fig. 21. Type graph for Example 4.16.

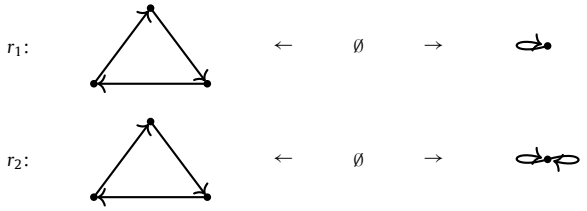


Fig. 22. Rules for Example 4.16.

4.3. Generation of non-garbage critical pairs

In general, there is no algorithm that, when given a DPO grammar and a graph, can decide if the graph is contained in the language generated by the grammar. That is, the universal membership problem is undecidable. It is easy to see that the similar problem of whether a graph is in the subgraph closure of the language generated by a DPO grammar is undecidable in general too.

This means that, unlike for critical pairs, generation of all the non-garbage critical pairs is not possible in general, due the impossibility of deciding subgraph membership. Though, if we are provided with an algorithm for testing if a graph is a subgraph of a graph of D, then we can generate the set of non-isomorphic $\mathcal{D}$-non-garbage critical pairs.

Definition 4.13 *(Universal subgraph membership problem).*

Instance: A graph grammar $\mathcal{G}$ over Σ and a graph G over Σ.
Question: Is $G \in \widehat{\mathrm{L}(\mathcal{G})}$?

Lemma 4.14. *The universal subgraph membership problem is undecidable.*

Proof. By reduction of undecidability of the emptiness problem, since $\emptyset \notin \widehat{\mathrm{L}(\mathcal{G})}$ if and only if $\mathrm{L}(\mathcal{G}) = \emptyset$. □

In practice, it is often the case that one can determine if a graph is contained in the subgraph closure of a language, and so undecidability is not too much of a concern. For example, if a graph language is known to only contain acyclic graphs, critical pairs with start graphs containing a cycle can be discarded as garbage. Moreover, it may not even be necessary to decide if a critical pair is garbage, if one can show that it is strongly joinable instead.

Here are some types of graph languages for which membership in the subgraph closure is decidable:

1. If D is finite, then membership in the subgraph closure can be decided simply by checking if the given graph is a subgraph of any graph in the language.
2. If D is subgraph-closed, then membership in the subgraph closure is the same as membership in D. Hence, membership is decidable if D is the class of discrete graphs, bounded degree graphs for some fixed bound, acyclic graphs, k-colourable graphs or planar graphs (see [20] for how to decide membership in the latter three classes). If D is the class of bounded treewidth graphs, for a fixed bound, membership can be decided by the algorithm in [21].
3. If D is specified by a so-called type graph (see below), then D is also subgraph-closed and membership is decidable. Type graph languages are studied by Corradini, König, and Nolte in [22].

Definition 4.15 *(Type graph language).* Given a signature Σ and a graph $G \in \mathcal{G}(\Sigma)$, define the *type graph language* $\mathrm{L}_\Sigma(G) = \{H \in \mathcal{G}(\Sigma) \mid \text{ there exists a graph morphism } H \to G\}$.

Example 4.16. It is easy to see that the language of 2-colourable unlabelled graphs, $\mathcal{D}$, can be specified by the type graph in Fig. 21. Consider the GT system with the two rules in Fig. 22. It is easy to see that $\mathcal{D}$ is closed under these rules (due to the fact that they are never applicable), that they are terminating (due to the fact that they are size reducing), and that there are five non-isomorphic critical pairs (Fig. 23, where the third pair is repeated twice more, formally with different matches), all of which are garbage (which we can machine check because $\mathcal{D}$ is specified by a type graph). Thus, by Corollary 4.9, the rules are confluent up to garbage on $\mathcal{D}$.

We are not aware of any other general families of grammars, or otherwise, for which we can solve the subgraph membership problem. We do not believe this problem is even decidable for hyperedge replacement grammars. This conjecture

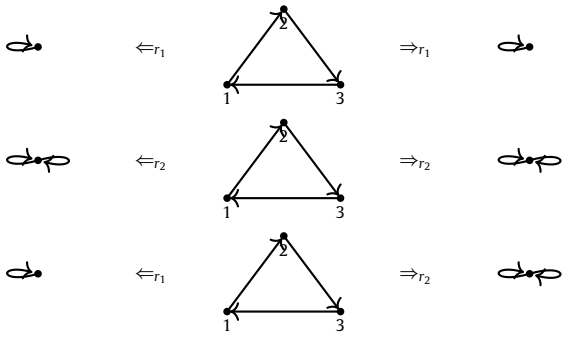


Fig. 23. Critical pairs for Example 4.16.

is not incompatible with this problem being easy for type graph languages, since the class of graph languages generated by hyperedge replacement grammars is incomparable with the class of graph languages specified by type graphs.

4.4. Checking for strong joinability

We briefly, explicitly discuss the process for checking if a pair of direct derivations is strongly joinable in a given GT system.

If a GT system is terminating, then it is decidable whether a pair of direct derivations is joinable or strongly joinable due to the fact that GT systems are finitely branching up to isomorphism, so there can only be finitely many successor graphs, up to isomorphism. It is then simply a matter of checking if there is an isomorphism between any of the successor graphs which behaves correctly with respect to the preserved nodes, as in the definition of strong joinability.

Alternatively, if a GT system is only terminating up to garbage on some language $\mathcal{D}$ and $\mathcal{D}$ is closed under T, then similarly, one can test joinability and strong joinability due to the fact that closedness ensures only finitely many successor graphs, as above.

4.5. Summary

We have presented our Generalised Critical Pair Lemma and Generalised Newman's Lemma, which together allow one to check for confluence up to garbage on some language $\mathcal{D}$ in the presence of termination and closedness. If there is an algorithm for solving the subgraph membership problem of $\mathcal{D}$, then we can effectively generate the set of non-isomorphic $\mathcal{D}$-non-garbage critical pairs and effectively test each of them for strong joinability. This process will always terminate, however may not provide a conclusive answer.

If the analysis completes with all the non-isomorphic $\mathcal{D}$-non-garbage critical pairs being strongly joinable, then we can conclude the system is confluent up to garbage on $\mathcal{D}$. If the analysis completes with a non-joinable $\mathcal{D}$-non-garbage critical pair that has its start graph in $\mathcal{D}$, then we can conclude the system is not confluent up to garbage on $\mathcal{D}$. In any other scenario, we cannot directly make a conclusion.

Finally, sometimes one might want to show confluence of a GT system T on a language $\mathcal{D}$ which is not necessarily closed under T. That is, either $\mathcal{D}$ is not closed under T, or indeed closure is simply unknown. In this scenario, one should attempt to show confluence on some larger language $\mathcal{E}$ containing $\mathcal{D}$. For example, if $\mathcal{D}$ contains only acyclic graphs, a good choice for $\mathcal{E}$ could be the language of acyclic graphs over the same signature. Transitivity of confluence up to garbage (Lemma 3.6) tells us that if we establish that T is confluent up to garbage on $\mathcal{E}$, then it is also confluent up to garbage on $\mathcal{D}$.

5. Backtracking-free 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 backtracking-free specification. We then demonstrate the applicability of our earlier results by showing that there are backtracking-free specifications for the languages of labelled series-parallel graphs and extended flow diagrams, 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. Backtracking-free specifications

Given a graph transformation system and a start graph, we can think of the pair as a graph grammar, generating a graph language. If the reversed system is 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. It is easy to see that confluence is a sufficient condition to

give determinism, however confluence is often not easily obtainable in practice. For this reason, we will consider the weaker property of confluence up to garbage on the generated language.

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. Language recognition by confluent graph reduction has been considered before by Bakewell, Plump, and Runciman, in the context of pointer structures [3,23], but without the concept of confluence up to garbage.

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

Theorem 5.2 *(Membership checking). Given a grammar $\mathcal{G} = (\Sigma, N, \mathcal{R}, S)$, $[G] \in \mathrm{L}(\mathcal{G})$ if and only if $G \Rightarrow^*_{\mathcal{R}^{-1}} S$ and G is terminally labelled. That is, $((\Sigma, \mathcal{R}^{-1}), \{S\})$ recognises $\mathrm{L}(\mathcal{G})$ over $\Sigma - N$.*

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. □

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

Definition 5.3 *(Backtracking-free specification).* Let $T = (\Sigma, \mathcal{R})$ be a GT system, $A \subseteq \Sigma$ an input signature, and $\mathcal{S}$ a finite set of graphs over Σ. Then we say that $(T, \mathcal{S})$ is a *backtracking-free specification* for a language $\mathcal{L}$ over A if $(T, \mathcal{S})$ recognises $\mathcal{L}$ over A, T is terminating on $\mathcal{G}(A)$, and T is confluent on $\mathcal{L}$.

Theorem 5.4. *Given a backtracking-free specification $(T, \mathcal{S})$ for a language $\mathcal{L}$ over $A \subseteq \Sigma$ and an input graph G over A, 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 some $S \in \mathcal{S}$, 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 $\mathcal{G}(A)$ 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 $\mathcal{L}$. □

For the remainder of this section, we look at two examples that demonstrate how we can use our Generalised Newman's Lemma and Generalised Critical Pair Lemma to verify if we have a backtracking-free specification for a language, given a grammar that generates the language.

5.2. Backtracking-free specification of series-parallel graphs

Series-parallel graphs were introduced by Duffin [24] as a model of electrical networks. A more general version of the class was introduced by Lawler [25] and Monma and Sidney [26] as a model for scheduling problems.

Definition 5.5. *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 parallel composition identifies the two sources and the two sinks, and sequential composition identifies the sink of one graph with the source of the other graph.

Fig. 24 shows an example series-parallel graph.

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, giving a graph grammar that generates the language:

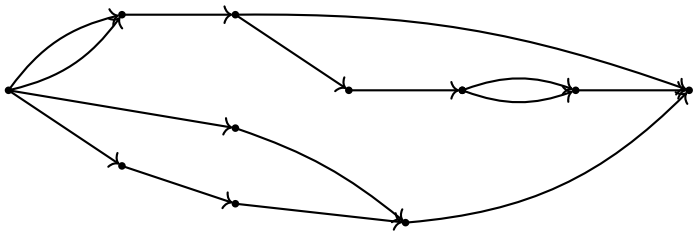


Fig. 24. Example series-parallel graph.

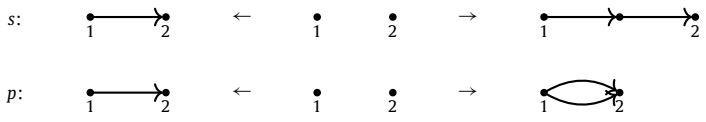


Fig. 25. Series-parallel graph generation rules.

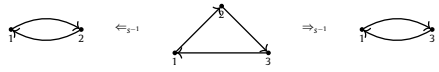
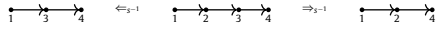
Pair/Property	Joinable	Strongly Joinable
	✓	✓
	✓	✓
	✓	✓
	✓	✓

Fig. 26. Series-parallel critical pair analysis.

Theorem 5.6 *(SP recognition [27]). The class of series-parallel graphs is the language generated by grammar* $SP = (\{\{\square\}, \{\square\}), (\emptyset, \emptyset), \{s, p\}, P)$ *(Fig. 25).*

By traditional critical pair analysis, one can establish that the reversed rules are confluent (Fig. 26), 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 (Figs. 7 and 27), however Hristakiev and Plump [28] observed that when doing the equivalent of this in GP 2, we no longer have confluence.

Definition 5.7. The class of *labelled series-parallel graphs* (LSPs) is all series-parallel graphs, but with arbitrary edge labels chosen from $\Sigma_E = \{a, b\}$.

The GT system with the 7 rules from Figs. 7 and 27 has 26 non-isomorphic critical pairs. 16 of the critical pairs are conflicts between the sequential reduction rules (Fig. 16) and the remaining 10 are conflicts between the parallel reduction rules (Fig. 28). There are no critical pairs between the series rules and the parallel rules. The non-joinable pairs confirm we no longer have confluence, however the fact that the language of labelled series-parallel graphs is closed under the rules, the rules are terminating, and all the non-garbage critical pairs are strongly joinable, allows us to conclude the rules are confluent up to garbage on the language of labelled series-parallel graphs.

Theorem 5.8 *(Backtracking-free LSP specification). Let* $\Sigma = (\{\square\}, \{a, b\})$, $T = (\Sigma, \{s_1, s_2, s_3, s_4, p_1, p_2, p_3\})$, $P_a = \bullet \xrightarrow{a} \bullet$ *and* $P_b = \bullet \xrightarrow{b} \bullet$. *Then* $(T, \{P_a, P_b\})$ *is a backtracking-free specification 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 5.6. Formally, our above discussion used Corollary 4.9 to establish that T is confluent up to garbage on $\mathcal{L}$, as required. □

Finally, we remark that this construction generalises for arbitrary edge label alphabets, and not just those of size 2. The number of conflicts is simply much larger, however the critical pair analysis will always conclude in the same way. Thus, we

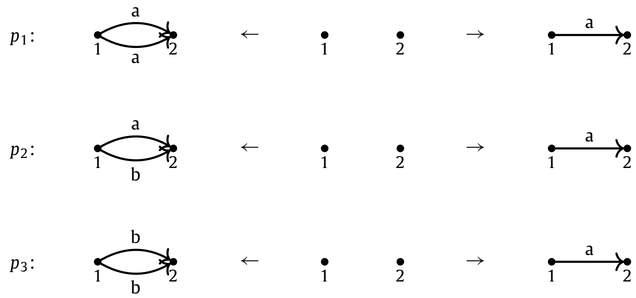


Fig. 27. Parallel LSP reduction rules.

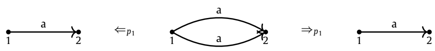
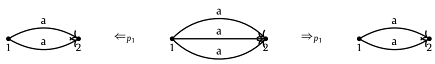
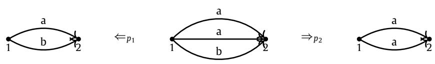
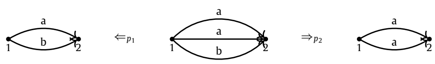
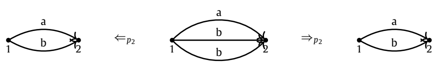
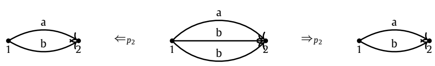
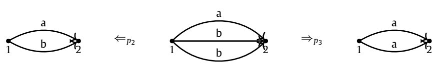
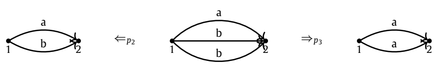
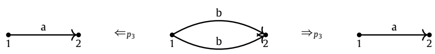
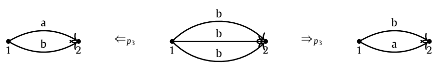
Pair/Property	Joinable	Strongly Joinable	Non-Garbage
	✓	✓	✓
	✓	✓	✓
	✓	✓	✓
	✓	✓	✓
	✓	✓	✓
	✓	✓	✓
	✓	✓	✓
	✓	✓	✓
	✓	✓	✓
	✓	✓	✓

Fig. 28. Critical pair analysis of parallel rules.

have shown that the obvious generalisation of the series-parallel reduction rules to a non-trivial edge labelling set admits a back-tracking free specification, even though the system is not confluent.

5.3. *Backtracking-free specification of extended flow diagrams*

In 1976, Farrow, Kennedy and Zucconi presented *semi-structured flow graphs*, defining a grammar with confluent reduction rules [29]. Plump has considered a restricted version of this language: *extended flow diagrams* (EFDs) [6]. The reduction rules for *extended flow diagrams* are a backtracking-free specification for the EFDs, despite not being confluent.

Throughout this subsection, we will use a shorthand notation for rules, where we assume all interface graphs contain no edges, and any node that appears in the interface graph will be labelled by a subscript number on both sides of the rule, writing only the left-hand side and right-hand side graphs. We also highlight persistent nodes within critical pairs in blue, for ease of reading. The colouring has no special meaning, other than that.

We now define extended flow diagrams using a grammar:

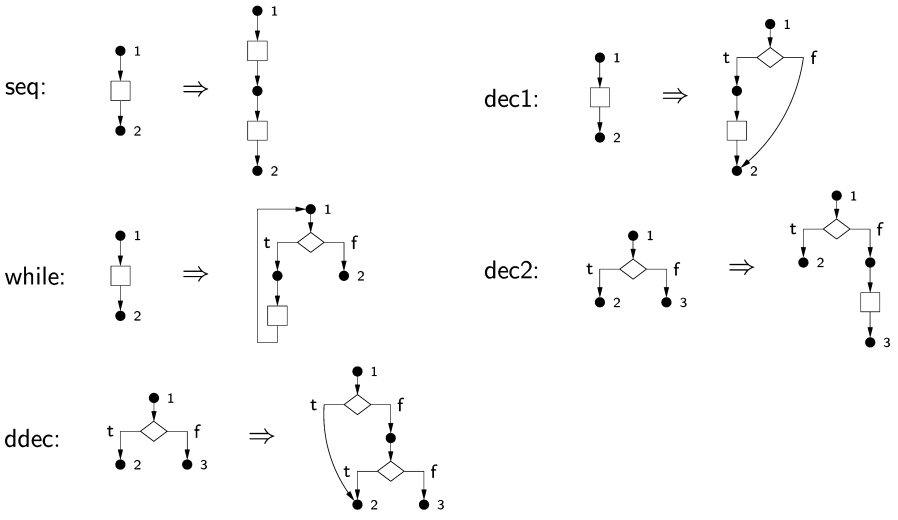


Fig. 29. EFD grammar rules.

Definition 5.9. The language of *extended flow diagrams* is generated by the grammar $\mathrm{EFD} = (\Sigma, N, \mathcal{R}, S)$ where $\Sigma_V = \{\bullet, \square, \diamond\}$, $\Sigma_E = \{t, f, \square\}$, $N_V = N_E = \emptyset$ (Fig. 29), $\mathcal{R} = \{seq, while, ddec, dec1, dec2\}$, and $S = \bullet\!\!\rightarrow\!\!\square\!\!\rightarrow\!\!\bullet$.

Before we show that reversing these rules admits a backtracking-free specification, we first need the following fact:

Lemma 5.10. *Every directed cycle in an EFD contains a t-labelled edge*

Proof. By induction. □

Theorem 5.11 *(Backtracking-free EFD specification). Let $T = (\Sigma, \mathcal{R}^{-1})$. Then $(T, \{S\})$ is a backtracking-free specification for* L(EFD) *over Σ.*

Proof. By Theorem 5.2, T recognises L(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 non-isomorphic critical pairs:

1. The pair exactly as in Fig. 30;
2. The pair in Fig. 30 with nodes 1 and 4 identified;
3. The pair exactly as in Fig. 31;
4. The pair in Fig. 31 with nodes 1 and 5 identified;
5. The pair in Fig. 31 with nodes 2 and 5 identified;
6. The pair exactly as in Fig. 32;
7. The pair exactly as in Fig. 33;
8. The pair in Fig. 33 with nodes 1 and 5 identified;
9. The pair exactly as in Fig. 34;
10. The pair exactly as in Fig. 35;

Pairs 1 through 9 are strongly joinable, and pair 10 is not joinable. Now observe that Lemma 5.10 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 (over Σ).

Clearly, $\mathcal{D}$ is subgraph closed, and then by our Generalised Critical Pair Lemma (Theorem 4.8), 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 3.11) to conclude confluence on $\mathcal{D}$ and thus, by Lemma 3.6, T is confluent on L(EFD).

Thus, T is a backtracking-free specification for L(EFD) over Σ, as required. □

6. Subcommutativity

In this section, we study critical pair analysis with a view to establish subcommutativity up to garbage, rather than confluence up to garbage, as previously in Section 3. We have already introduced subcommutativity in Section 2 and subcommutativity up to garbage in Section 3.

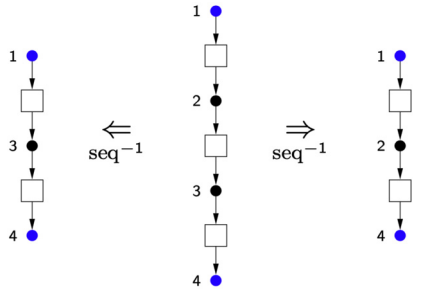


Fig. 30. EFD critical pair 1.

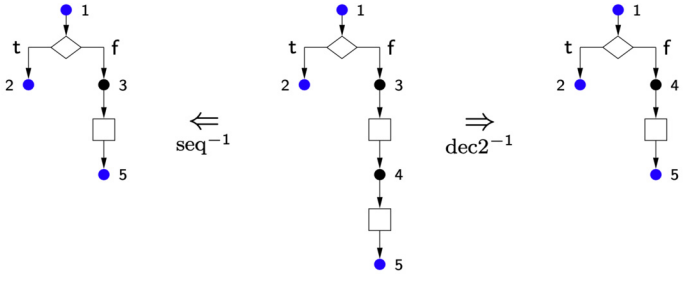


Fig. 31. EFD critical pair 3.

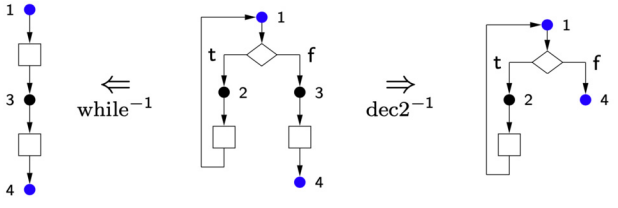


Fig. 32. EFD critical pair 6.

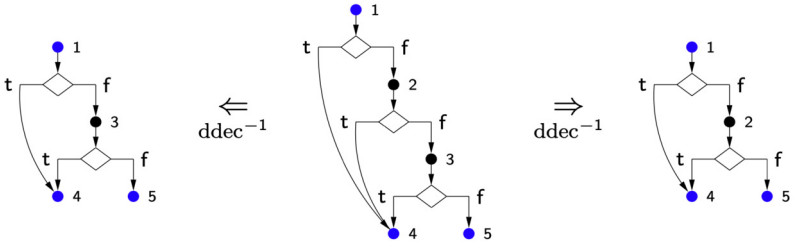


Fig. 33. EFD critical pair 7.

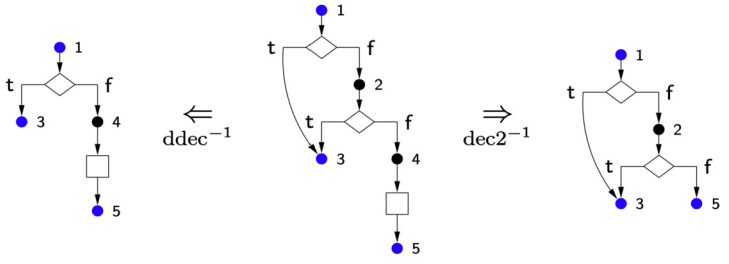


Fig. 34. EFD critical pair 9.

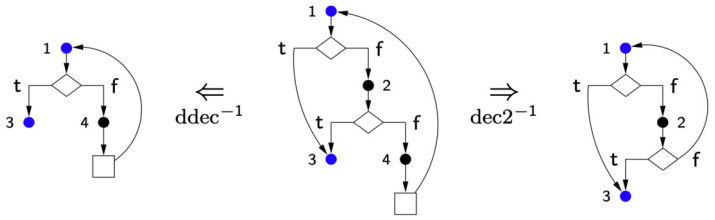


Fig. 35. EFD critical pair 10.

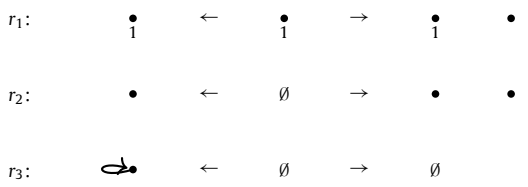


Fig. 36. Rules for Example 6.3.

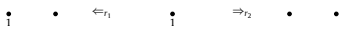
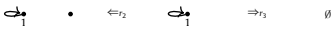
Pair/Property	Strongly Subcommutative	Non-Garbage
	✓	✓
	✗	✗

Fig. 37. Critical pair analysis for Example 6.3.

We start by giving the main result of this section:

Theorem 6.1. *Let $T = (\Sigma, \mathcal{R})$ and $\mathcal{D} \subseteq \mathcal{G}(\Sigma)$. If all T's $\mathcal{D}$-non-garbage critical pairs are strongly subcommutative, then T is subcommutative up to garbage on $\mathcal{D}$.*

Proof. Easy modification of the proof of the original theorem (Theorem 4.8). □

Corollary 6.2. *Let $T = (\Sigma, \mathcal{R})$ and $\mathcal{D} \subseteq \mathcal{G}(\Sigma)$. If all T has no $\mathcal{D}$-non-garbage critical pairs, then T is subcommutative up to garbage on $\mathcal{D}$.*

Just as before, it suffices to only analyse the non-isomorphic critical pairs. A notable difference is that closure and termination are no longer needed to check for joinability, since we are only looking for strong subcommutativity of critical pairs. However, as noted in Subsection 3.2, closedness is required in order for subcommutativity up to garbage to imply confluence up to garbage (see Example 3.9 and Lemma 3.10).

Revisiting our examples in Section 5, all 4 of the critical pairs of the series-parallel reduction system are actually strongly subcommutative, all 18 non-garbage critical pairs of the labelled series-parallel reduction rules are strongly subcommutative, and so are the 9 non-garbage critical pairs of the extended flow diagram reduction system.

We finish this subsection with a simple example demonstrating termination is not a requirement to establish subcommutativity up to garbage.

Example 6.3. Let $\mathcal{D}$ be the language of discrete graphs and T be the GT system with the three rules in Fig. 36. There are two non-isomorphic critical pairs, all of which are garbage (Fig. 37), which allow us to immediately conclude subcommutativity of T up to garbage on $\mathcal{D}$ (Theorem 6.1). Notice $\mathcal{D}$ is closed under T, which means T is also confluent to garbage on $\mathcal{D}$ (Lemma 3.10).

Notice these rules aren't terminating, even up to garbage on discrete graphs, thus naive machine checking for strong joinability of these pairs would not terminate, however we only need to check for subcommutativity.

7. Conclusion, related and future work

In this paper we have introduced local confluence, confluence, subcommutativity, 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 checkable conditions for confluence and subcommutativity 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. We have directly applied our results to recognition of languages, looking specifically at the class of extended flow diagrams and the class of 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 the domain [11]. Indeed, one might only be interested in the non-garbage critical pairs themselves, and classification of conflicts [30,31].

7.1. Generalisations

Our results also work if we relax the injectivity requirement of the $K \to R$ morphism in rules. One should note that the two equivalent definitions of parallel independence we have in Subsection 2.5 were specialised for injective rules only. More generally, two direct derivations $H_1 \Leftarrow_{r_1,g_1} G \Rightarrow_{r_2,g_2} H_2$ are parallelly independent if there are morphisms $L_1 \to D_2$ and $L_2 \to D_1$ such that $L_1 \to D_2 \to G = L_1 \to G$, $L_2 \to D_1 \to G = L_2 \to G$, $L_1 \to D_1 \to H_2$ is injective, and $L_2 \to D_1 \to H_1$ is injective [11]. Our results also work in the setting of hypergraph transformation, as well as just graph transformation, with almost identical proofs. We think it is extremely likely our results hold for any $\mathcal{M}$-adhesive system with the usual restrictions [32], by modification of the original proof of the Generalised Critical Lemma from Campbell's BSc Thesis [9], which operates by showing completeness of the non-garbage critical pairs. In an unpublished report, we have also shown that these results hold for graph transformation with relabelling [33], which is important, since the setting is not $\mathcal{M}$-adhesive [34] and is the graph transformation framework used by GP 2 [35,36].

7.2. Related work

Lambers, Ehrig and Orejas investigated essential critical pairs [37] and the continued work by others including Born and Taentzer [30]. That said, all of our examples exhibit only essential critical pairs, so checking if a critical pair is essential only has the effect of slowing down the analysis in our examples. Please note that in this paper we are not interested in the actual critical pairs or even the property of confluence, but the weaker property of confluence up to garbage. This property still gives a lot of the benefits of confluence such as backtracking-free reduction. Blakewell, Plump and Runciman investigate the of graph reduction for specifying pointer structures [3]. Of particular interest in this context are confluent (up to garbage) reduction systems. Confluence modulo is also studied in other areas of computer science such as constraint handling rules (CHR), see for example [38].

7.3. Future work

Confluence analysis of GT systems (and related systems) still remains a generally under-explored area. Future work includes the investigation of any relation with essential critical pairs and developing further checkable sufficient conditions under which one can decide if a graph is in the subgraph closure of a language, beyond those in Subsection 4.3. Finally, applying our theory in a rooted context and to GP 2 is future work [36]. 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.

It is also not obvious what the relation is between confluence up to garbage and graphs satisfying negative constraints or nested application conditions [39,32]. 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.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

[1] H. Ehrig, M. Pfender, H. Schneider Graph-grammars, An algebraic approach, in: Proc. 14th Annual Symposium on Switching and Automata Theory, SWAT 1973, IEEE, 1973, pp. 167–180, https://doi.org/10.1109/SWAT.1973.11.

[2] H. Ehrig, K. Ehrig, U. Prange, G. Taentzer, Fundamentals of Algebraic Graph Transformation, Monographs in Theoretical Computer Science, An EATCS Series, Springer, 2006, https://doi.org/10.1007/3-540-31188-2.

[3] A. Bakewell, D. Plump, C. Runciman, Specifying pointer structures by graph reduction, Tech. rep, Department of Computer Science, University of York, UK, 2003, https://www.cs.york.ac.uk/plasma/publications/pdf/BakewellPlumpRuncimanReport.03.pdf.

[4] D. Plump, Term Graph Rewriting, World Scientific, 1999, pp. 3–61, https://doi.org/10.1142/9789812815149_0001.
[5] D. Plump, Hypergraph rewriting: critical pairs and undecidability of confluence, in: Term Graph Rewriting, John Wiley and Sons, 1993, pp. 201–213.
[6] D. Plump, Confluence of Graph Transformation Revisited, Lecture Notes in Computer Science, vol. 3838, Springer, 2005, pp. 280–308, https://doi.org/10.1007/11601548_16.
[7] M. Newman, On theories with a combinatorial definition of "equivalence", Ann. Math. 43 (2) (1942) 223–243, https://doi.org/10.2307/1968867.
[8] G. Campbell, D. Plump, Confluence up to garbage, in: F. Gadducci, T. Kehrer (Eds.), Proc. 13th International Conference on Graph Transformation, ICGT 2020, in: Lecture Notes in Computer Science, vol. 12150, Springer, 2020, pp. 20–37, https://doi.org/10.1007/978-3-030-51372-6_2.
[9] G. Campbell, Efficient graph rewriting, BSc thesis, Department of Computer Science, University of York, UK, 2019, https://arxiv.org/abs/1906.05170.
[10] F. Baader, T. Nipkow, Term Rewriting and All That, Cambridge University Press, 1998.
[11] A. Habel, J. Müller, D. Plump, Double-pushout graph transformation revisited, Math. Struct. Comput. Sci. 11 (5) (2001) 637–688, https://doi.org/10.1017/S0960129501003425.
[12] D. Knuth, P. Bendix, Simple word problems in universal algebras, in: Computational Problems in Abstract Algebras, Pergamon Press, 1970, pp. 263–297, https://doi.org/10.1016/B978-0-08-012975-4.50028-X.
[13] G. Huet, Confluent reductions: abstract properties and applications to term rewriting systems, J. ACM 27 (4) (1980) 797–821, https://doi.org/10.1145/322217.322230.
[14] H. Ehrig, H.-J. Kreowski, Parallelism of manipulations in multidimensional information structures, in: Proc. 5th Symposium on Mathematical Foundations of Computer Science, MFCS 1976, in: Lecture Notes in Computer Science, vol. 45, Springer, 1976, pp. 284–293, https://doi.org/10.1007/3-540-07854-1_188.
[15] D. Plump, Computing by graph rewriting, Habilitation thesis, Universität Bremen, Fachbereich Mathematik und Informatik, 1999.
[16] B. Courcelle, 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, in: Lecture Notes in Computer Science, vol. 344, Springer, 1989, pp. 30–53, https://doi.org/10.1007/3-540-50728-0_34.
[17] P. Fradet, D.L. Métayer, Structured gamma, Sci. Comput. Program. 31 (2–3) (1998) 263–289, https://doi.org/10.1016/S0167-6423(97)00023-3.
[18] J. Berstel, Transductions and Context-Free Languages, Vieweg+Teubner, 1979, https://doi.org/10.1007/978-3-663-09367-1.
[19] A.-C. Caron, 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, in: Lecture Notes in Computer Science, vol. 493, Springer, 1991, pp. 74–89, https://doi.org/10.1007/3-540-53982-4_5.
[20] S. Skiena, The Algorithm Design Manual, 2nd edition, Springer, 2008, https://doi.org/10.1007/978-1-84800-070-4.
[21] H. Bodlaender, A linear-time algorithm for finding tree-decompositions of small treewidth, SIAM J. Comput. 25 (6) (1996) 1305–1317, https://doi.org/10.1137/S0097539793251219.
[22] A. Corradini, B. König, D. Nolte, Specifying graph languages with type graphs, J. Log. Algebraic Methods Program. 104 (2019) 176–200, https://doi.org/10.1016/j.jlamp.2019.01.005.
[23] A. Bakewell, D. Plump, C. Runciman, Specifying pointer structures by graph reduction, in: Proc. Second International Workshop on Applications of Graph Transformations with Industrial Relevance, AGTIVE 2003, in: Lecture Notes in Computer Science, vol. 3062, Springer, 2004, pp. 30–44, https://doi.org/10.1007/978-3-540-25959-6_3.
[24] R.J. Duffin, Topology of series-parallel networks, J. Math. Anal. Appl. 10 (2) (1965) 303–318, https://doi.org/10.1016/0022-247X(65)90125-3.
[25] E. Lawler, Sequencing jobs to minimize total weighted completion time subject to precedence constraints, Ann. Discrete Math. 2 (1978) 75–90, https://doi.org/10.1016/S0167-5060(08)70323-6.
[26] C. Monma, J. Sidney, Sequencing with series-parallel precedence constraints, Math. Oper. Res. 4 (3) (1979) 215–224, https://doi.org/10.1287/moor.4.3.215.
[27] D. Plump, Reasoning about graph programs, in: Proc. 9th International Workshop on Computing with Terms and Graphs, TERMGRAPH 2016, in: Electronic Proceedings in Theoretical Computer Science, vol. 225, Open Publishing Association, 2016, pp. 35–44, https://doi.org/10.4204/EPTCS.225.6.
[28] I. Hristakiev, D. Plump, Checking graph programs for confluence, in: Software Technologies: Applications and Foundations – STAF 2017 Collocated Workshops, Revised Selected Papers, in: Lecture Notes in Computer Science, vol. 10748, Springer, 2018, pp. 92–108, https://doi.org/10.1007/978-3-319-74730-9_8.
[29] R. Farrow, K. Kennedy, L. Zucconi, Graph grammars and global program data flow analysis, in: Proc. 17th Annual Symposium on Foundations of Computer Science, SFCS 1976, IEEE, 1976, pp. 42–56, https://doi.org/10.1109/SFCS.1976.17.
[30] L. Lambers, K. Born, F. Orejas, D. Strüber, G. Taentzer, Initial Conflicts and Dependencies: Critical Pairs Revisited, Lecture Notes in Computer Science, vol. 10800, Springer, 2018, pp. 105–123, https://doi.org/10.1007/978-3-319-75396-6_6.
[31] L. Lambers, J. Kosiol, D. Strüber, G. Taentzer, Exploring conflict reasons for graph transformation systems, in: Proc. 12th International Conference on Graph Transformation, ICGT 2019, in: Lecture Notes in Computer Science, vol. 11629, Springer, 2019, pp. 75–92, https://doi.org/10.1007/978-3-030-23611-3_5.
[32] H. Ehrig, U. Golas, A. Habel, L. Lambers, F. Orejas, $\mathcal{M}$-adhesive transformation systems with nested application conditions, Part 2: embedding, critical pairs and local confluence, Fundam. Inform. 118 (1–2) (2012) 35–63, https://doi.org/10.3233/FI-2012-705.
[33] G. Campbell, D. Plump, Efficient recognition of graph languages, Tech. rep, Department of Computer Science, University of York, UK, 2019, https://arxiv.org/abs/1911.12884.
[34] A. Habel, D. Plump, $\mathcal{M},\mathcal{N}$-adhesive transformation systems, in: H. Ehrig, G. Engels, H.-J. Kreowski, G. Rozenberg (Eds.), Proc. 6th International Conference on Graph Transformation, ICGT 2012, in: Lecture Notes in Computer Science, vol. 7562, Springer, 2012, pp. 218–233, https://doi.org/10.1007/978-3-642-33654-6_15.
[35] D. Plump, The design of GP 2, in: S. Escobar (Ed.), Proc. 10th International Workshop on Reduction Strategies in Rewriting and Programming, WRS 2011, in: Electronic Proceedings in Theoretical Computer Science, vol. 82, Open Publishing Association, 2012, pp. 1–16, https://doi.org/10.4204/EPTCS.82.1.
[36] C. Bak, 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/.
[37] L. Lambers, H. Ehrig, F. Orejas, 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, in: Electronic Notes in Theoretical Computer Science, vol. 211, Elsevier, 2008, pp. 17–26, https://doi.org/10.1016/j.entcs.2008.04.026.
[38] D. Gall, T. Frühwirth, Confluence modulo equivalence with invariants in constraint handling rules, in: J. Gallagher, M. Sulzmann (Eds.), Proc. 14th International Symposium on Functional and Logic Programming, FLOPS 2018, in: Lecture Notes in Computer Science, vol. 10818, Springer, 2018, pp. 116–131, https://doi.org/10.1007/978-3-319-90686-7_8.
[39] L. Lambers, 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.