Confluence in Data Reduction: Bridging Graph Transformation and Kernelization*

Hartmut Ehrig Claudia Ermel Falk Hüffner†
Rolf Niedermeier Olga Runge

Institut für Softwaretechnik und Theoretische Informatik,
Technische Universität Berlin, Germany

January 17, 2013

Abstract

Kernelization is a core tool of parameterized algorithmics for coping with computationally intractable problems. A *kernelization* reduces in polynomial time an input instance to an equivalent instance whose size is bounded by a function only depending on some problem-specific parameter k; this new instance is called problem kernel. Typically, problem kernels are achieved by performing efficient data reduction rules. So far, there was little systematic study in the literature concerning the mutual interaction of data reduction rules, in particular whether data reduction rules for a specific problem always lead to the same reduced instance, no matter in which order the rules are applied. This corresponds to the concept of confluence from the theory of rewriting systems. We argue that it is valuable to study whether a kernelization is confluent, using the NP-hard graph problems (Edge) Clique Cover and Partial Clique Cover as running examples. We apply the concept of critical pair analysis from graph transformation theory, supported by the AGG software tool. These results support the main goal of our work, namely, to establish a fruitful link between (parameterized) algorithmics and graph transformation theory, two so far unrelated fields.

1 Introduction

Theoretical Computer Science is usually divided into algorithm-oriented research and description-oriented research (as witnessed by the two volumes "Algorithms and Complexity" and "Formal Methods and Semantics" of the Handbook of

*An extended abstract of this paper appears in *Proceedings of the 8th Conference on Computability in Europe (CiE 2012), LNCS 7318, Springer.* The present long version contains additional details in Section 4 and all proofs.

†Supported by DFG project PABI (NI 369/7-2).

Theoretical Computer Science [22]). Unfortunately, the corresponding research communities typically work in two "parallel worlds" with relatively little interaction. In this work, we propose a new link between algorithmics and formal methods that may lead to a fruitful "interdisciplinary" field of research. More specifically, we develop a connection between efficient preprocessing of NP-hard (graph) problems by kernelization [2, 15] and the theory of graph transformations [10, 30]: We employ the concept of confluence of rewriting systems to show "uniqueness results" for problem kernels. This leads to the concept of confluent data reduction rules, having a number of both theoretical and practical benefits as discussed in the following.

1.1 Confluence in Kernelization

Data reduction, also known as polynomial-time preprocessing, is a classic approach for dealing with NP-hard combinatorial optimization problems (see [2, 15] for surveys). The idea is to remove redundant parts of the input, thereby obtaining a hard "core" of the instance. Costly algorithms need then only be applied to this core. Data reduction is thus useful in virtually any approach to solving computationally hard problems, whether heuristic, approximative, or exact. Formally, we consider only decision problems, where (*data*) *reduction rules* replace in polynomial time a given problem instance I by an instance I' with $|I'| < |I|$. We say that the data reduction rule is *correct* when I is a yes-instance iff I' is a yes-instance. An instance to which none of a given set of reduction rules applies is called *reduced* with respect to these rules.

While they are a standard technique for practitioners, only fairly recently have data reduction rules been the subject of wider theoretical analyses, using the concept of a *problem kernel* [2, 15]. This notion comes from the field of *parameterized complexity* [7, 11, 26], where the performance of algorithms is analyzed not just in terms of the problem size n, but also in terms of a parameter k, for example the solution size. A *kernelization* is a data reduction that creates an equivalent instance whose size depends only on the parameter k, and not on the original input size n anymore (see Section 2.1 for a more formal definition).

Note that applying data reduction rules is nondeterministic in general and thus may lead to different reduced instances. In fact, the result depends not only on the order in which the rules are applied, but also on the occurrence where each rule is applied to the current instance. Thus, we have two sources of nondeterminism, which in general may lead to different results. A trivial way to avoid different results is to forbid nondeterminism by fixing an execution order of the rules and occurrences. This, however, may cause a smaller reduction power since with a different execution order of the data reduction rules further reductions might be possible. In this paper we do not restrict the nondeterminism of rule applications, but we analyze confluence of the rule set to obtain unique instances (see Section 1.2).

We call a terminating set of data reduction rules *confluent* if any order of application of the rules yields a unique reduced instance, up to isomorphism.

Confluence is a standard concept from graph transformation theory (see below). There are a number of reasons why it seems useful to investigate whether data reduction rules are confluent: If they are, then the rules are robust in a sense; we obtain a unique starting point for further processing after the data reduction has been performed. In an implementation of the rules, we can apply the rules in any order without worrying about the result, and can thus optimize for the speed of their application. If the rules are not confluent, this might indicate some "slack" in the rules: some orders of application might lead to worse results, that is, larger kernels. Investigating all this might lead to improved data reduction rules. Moreover, insight on the interaction of data reduction rules can lead to faster kernelizations.

Confluence was also exploited by Kneis et al. [20]. They present a set of reduction rules for a graph with m edges and show that a particular order of execution yields a tree decomposition of width $m/5.769 + O(\log n)$. Then, they show that a different order of application yields a path decomposition. Thus, a proof of confluence implies that any order of application of data reduction yields a path decomposition of width $m/5.769 + O(\log n)$.

Finally, proving confluence is also a good way to check for possible conflicts between data reduction rules, since all possible interactions need to be taken into account. It might also give an incentive to create "minimal" kernelization rules in order to make confluence proofs easier, which could give a sharper picture of what exactly is needed to achieve a problem kernel.

1.2 Confluence of Graph Transformation Systems

The theory of graph grammars and graph transformation systems has been started in the early 1970s [8] as a generalization of Chomsky grammars and term rewriting systems, which are based on strings and trees, respectively. The main idea is the rule-based modification of graphs. Graph transformations are most suitable to model the operational semantics of visual languages and also to define model transformations between different kinds of models. Several approaches for graph transformations are known [30], including logical and algebraic approaches. A graph transformation system consists of a set of graph rules. They are applied in a non-deterministic way, leading to graph transformation steps $G \Longrightarrow H$ and sequences $G \overset{*}{\Longrightarrow} H$. A single rule consists of a left-hand side graph *LHS*, a right-hand side graph *RHS*, and their intersection graph. A subgraph of the input graph that is isomorphic to *LHS* of a rule is called *match* or *occurrence* of the rule. To apply a rule, first a match has to be found. In a second step, the corresponding graph without the intersection graph has to be deleted. The result is a context graph, which is glued together with *RHS* at the vertices and edges of the intersection graph. A graph transformation system is called *confluent* if for each pair of graph transformation sequences $G \overset{*}{\Longrightarrow} G_1$, $G \overset{*}{\Longrightarrow} G_2$ with the same domain, there is a graph G_3 together with sequences $G_1 \overset{*}{\Longrightarrow} G_3$ and $G_2 \overset{*}{\Longrightarrow} G_3$, as shown in Fig. 1 (a).

There are numerous applications in software engineering, concurrency, and distributed systems, where confluence of graph transformations plays an impor-

Figure 1: Confluence (a) and local confluence (b) for graph transformations

tant role [9]. Confluence together with termination, i. e., non-existence of infinite transformation sequences, implies that any order of applying the rules as long as possible yields a unique graph, up to isomorphism. Moreover, we obtain for isomorphic input graphs isomorphic reduced graphs [10]. Altogether, this means that in case of confluence and termination, the graph transformation system has *functional behavior*. This property is especially important to obtain unique normal forms for the operational semantics of rewriting systems.

In order to show confluence it is sufficient to show local confluence and termination [18, 25]. Local confluence means confluence for the special case that the given sequences from G to G_1 and G_2 are transformation steps $G \Longrightarrow G_1$ and $G \Longrightarrow G_2$, where in each step only one transformation rule is applied (as indicated in Fig. 1 (b)). Data reduction rules for kernelization for graph problems define graph transformation systems based on undirected graphs, such that the general concepts of (local) confluence and termination are applicable. An important technique to analyze local confluence, called *critical pair analysis*, will be discussed below.

Structure of the paper. Basic notions about kernelization and critical pair analysis are presented in Section 2. In Section 3, we examine a kernelization (Rules 1 to 3) for Clique Cover from the literature [14, 16] and show that it is confluent by a direct proof. This proof also yields a linear-time algorithm to compute the kernel. We then demonstrate how confluence can be shown alternatively via critical pair analysis (Section 3.3). This analysis is supported by the software tool AGG [1], which allows automated detection of critical pairs. Partial Clique Cover is introduced in Section 4 together with Rules 4 to 6 extending Rules 1 to 3. In Theorem 4, we show that Rules 4 to 6 yield a kernel for Partial Clique Cover with 2^{k+c} vertices, where c is the number of covered edges in the given graph. In Section 4.2, we show confluence of Rules 4 to 6, where the explicit proof is quite complex; again an alternative via critical pair analysis is discussed (Section 4.3). We emphasize that the primary interest in this paper is to demonstrate the benefits of applying graph transformation techniques to the field of kernelization in general; the concrete new results in edge clique covering thus are of secondary interest only. Hence, our contribution is of mostly conceptual nature.

2 Basic Concepts of Kernelization and Critical Pair Analysis

Kernelization is a core concept of parameterized algorithmics, while critical pair analysis is of central importance in analyzing confluence of (graph) transformation systems.

2.1 Kernelizations

Data reduction for NP-hard problems is often seen as a heuristic task because in classic complexity analysis, nothing can be proved about the quality of data reduction; this is because even the smallest provable data reduction would, by repetition, imply polynomial-time solvability of the instance and thus P = NP [2, 15]. Parameterized complexity, however, provides a useful notion of the power of data reduction with the concept of a *problem kernel*.

A *parameterized problem* can be defined by a set of instances (x, k), where k is called the *parameter* [7, 11, 26].

Definition 1. *Let L be a parameterized problem. A* reduction to a problem kernel *or* kernelization *is a transformation via data reduction rules of an instance (x, k) to an instance (x', k') (the* problem kernel *of instance (x, k)), such that*

- $(x, k) \in L \iff (x', k') \in L$,
- $|x'| \leq g(k)$ *for some arbitrary computable function g depending only on k,*
- $k' \leq k$, *and*
- *the transformation runs in polynomial time.*

We call $g(k)$ the size of the problem kernel of the parameterized problem L.

In other words, a kernelization is a data reduction that creates an equivalent instance whose size depends only on the parameter k and not on the original input size n anymore.

2.2 Critical pair analysis in graph transformation theory

The algebraic theory of graph transformations [10] provides a specific technique known from term rewriting systems [18], called *critical pair analysis*, which has been generalized to graph transformation systems by Plump [29]. Critical pair analysis supports the verification of local confluence using the software system AGG [1]. The main idea is to show local confluence not for all pairs of (a possibly infinite number of) transformation steps $G \Longrightarrow G_1$ and $G \Longrightarrow G_2$ via rules r_1 resp. r_2, but only for all *critical pairs*. A pair of transformation steps is called a *critical pair* if it is *conflicting in a minimal context* in the following sense: The pair $G \Longrightarrow G_1$, $G \Longrightarrow G_2$ via r_1, r_2 is called *parallel independent* if there are transformation steps $G_1 \Longrightarrow G_3$, $G_2 \Longrightarrow G_3$ via r_2, r_1. A pair is called *conflicting*

if it is not parallel independent. It has *minimal context* if each vertex and edge in G belongs to the match of r_1 or r_2 in G of the transformation steps $G \Longrightarrow G_1$ and $G \Longrightarrow G_2$ via r_1, r_2. For a graph transformation system with a finite number of rules based on finite graphs, there is only a finite number of critical pairs. All of them can be computed automatically by the graph transformation analysis tool AGG [1]. The Local Confluence Theorem for algebraic graph transformations [10] implies local confluence of a graph transformation system provided that all critical pairs are *strictly confluent*, where "strictness" is an additional technical condition for the transformations. The verification of strict confluence for critical pairs can also be supported by AGG and is applied to data reduction rules in Sections 3 and 4.

As pointed out in Section 1.2, local confluence and termination implies confluence [18, 25]. Hence, for terminating data reduction rules it is sufficient to show strict confluence for all critical pairs, in order to obtain confluence of data reduction.

The application of critical pair analysis to data reduction rules, however, is not yet fully automated. The first reason is that the Local Confluence Theorem [10] based on critical pairs is valid for directed graphs (with parallel edges and loops) and several other kinds of graphs (such as typed, attributed, or hypergraphs), but not yet formally proved for undirected graphs as considered for data reduction in this paper. The second reason is that data reduction rules in general are *rule schemes* in the sense of graph transformation theory, where rule schemes can be applied to an unbounded number of vertices, and rules are applied to a constant-size subgraph, depending on the size of the left-hand side of the rule. Each rule schema corresponds to a—possibly infinite—set of rules in the sense of Ehrig et al. [10]. For these reasons, we prove local confluence also directly in Sections 3 and 4; in the case of Partial Clique Cover, the proof is quite complex, based on a large number of case distinctions. These proofs depend strictly on the specific rules. Altogether, we study two approaches for local confluence, the direct proof and the proof based on critical pairs. It is an interesting challenge for future work to extend the theory of graph transformations [10]—and the corresponding tool AGG—to handle also data reduction rules in a more general way.

3 Case Study Clique Cover

We use the well-known NP-hard Clique Cover problem for our first case study.

Clique Cover
Instance: An undirected graph $G = (V, E)$ and an integer $k \geq 0$.
Question: Is there a set of at most k cliques in G such that each edge in E has both its endpoints in at least one of the selected cliques?

For an instance (G, k), we call a set of at most k cliques that covers all edges a *solution*. Choosing Clique Cover[1] has several reasons: It is a conceptually simple graph problem, and the best known (theoretical) data reduction rules so far are easy to understand and also applied in practice [13, 14, 27, 31]. Moreover, Clique Cover has a kernelization with a size bound of 2^k vertices [14, 16], and it was recently shown that under standard complexity-theoretic assumptions, this cannot be improved to a polynomial bound [5].

Clique Cover occurs in many contexts and applications (see e. g. [6, 13, 14, 19, 21] and references therein). It is also known as Keyword Conflict problem [19] or Covering by Cliques [12] or Intersection Graph Basis [12]. Clique Cover is NP-hard [21, 28], even when restricted to planar graphs [4] or graphs with maximum degree 6 [17]. Further, Clique Cover is not approximable within a factor of n^ϵ for some $\epsilon > 0$ unless P = NP [23]. Gramm et al. [14] report on experiments with an exact algorithm based on data reduction and a search tree. They also prove that their data reduction rules yield a kernel. These data reduction rules are the basis of our investigations on confluence. The rules have also recently found applications in optimizing compilers [31] and in computational biology [27].

3.1 Kernelization for Clique Cover.

For the currently only known kernelization for Clique Cover with parameter k, the following data reduction rules are used [14, 16].[2]

Rule 1. *Remove isolated vertices, that is, vertices with no neighbors.*

Rule 2. *If there is an isolated edge, then delete it and decrease k by one.*

Two vertices $u, v \in V$ are called *twins* if $\{u, v\} \in E$ and u and v have exactly the same neighbors (except for v and u, respectively).

Rule 3. *If $\{u, v\}$ are twins and $\{u, v\}$ is not an isolated edge, then delete u (that is, remove it from the vertex set and all incident edges from the edge set).*

Theorem 1 ([14, 16])**.** *Rules 1 to 3 are correct and yield a problem kernel for* Clique Cover *with at most 2^k vertices.*

Note that a rule R is correct if for each transformation of instances from I to I' via R, we have that I is a yes-instance iff I' is a yes-instance. For technical correctness of the kernel (as defined in Section 2), we additionally need to check whether after exhaustive application of Rules 1 to 3 there are more than 2^k vertices left, and if so, the instance is replaced with a small "no"-instance (for instance, $k + 1$ disjoint edges). We omit such trivial checks in the following.

[1] Note that in the literature sometimes also covering vertices instead of edges by cliques is called Clique Cover.

[2] We note that Gramm et al. [14] used different rules involving "covered edges", which are equivalent to the rules presented here if the initial instance does not have covered edges (except that Rule 3' from Gramm et al. [14] does not treat isolated edges correctly; as already noted by Gyárfás [16], they require a special case).

3.2 Confluence of Data Reduction for Clique Cover

We now show that the kernelization rules from Theorem 1 are confluent.

Theorem 2. *The set of Rules 1 to 3 for* Clique Cover *is confluent.*

Proof. Clearly, the order of application for Rule 1 and Rule 2 with respect to any of the three rules is not relevant, since their application does not affect the applicability of other rules. It remains to show that the relative order of applications of Rule 3 does not matter.

If we consider two vertices as equivalent when they are twins, then we obtain an equivalence relation on the vertex set. Thus, we can partition the vertex set into the equivalence classes of this relation, called *twin classes*. Note that every twin class forms a clique in the graph. Let the *twin graph*[3] of a graph be a graph with the twin classes as vertices and an edge between two twin classes if there is an edge between one vertex from one class and one vertex from the other class.

The twin graph does not change (up to isomorphism) when Rule 3 is applied, since u and v must be from the same twin class and the rule thus always leaves at least one vertex in any twin class. Further, Rule 3 is applicable until a twin class contains exactly one vertex (if it is connected to vertices outside the twin class) or two vertices (if it is an isolated clique). Since the twin graph and the number of vertices per twin class uniquely represent a graph up to isomorphism, we obtain confluence. □

This proof also yields a shortcut to calculate the result of the kernelization, whose naive calculation would require $O(|E| \cdot |V|^2)$ time (Gramm et al. [14] only state the running time of $O(|V|^4)$ for Rules 1 to 3 plus another rule).

Corollary 1. *A 2^k-vertex kernel for* Clique Cover *can be found in linear time.*

Proof. From the proof of Theorem 2, we can see that it is sufficient to calculate the twin graph, contract each twin class to a single vertex, and then delete isolated vertices and edges. Finding the twin graph can be done in linear time [24, Corollary 7.4], so the kernelization can be done in linear time, too. □

3.3 Confluence via Critical Pair Analysis

As pointed out in Section 2.2, the standard way to show confluence of a rule set in graph transformation theory [10] is to construct all critical pairs and to show for each critical pair that it is strictly confluent. The approach has been shown for directed graphs [10], and we are confident that it can also be extended to undirected graphs as considered in this paper, in particular to data reduction for Clique Cover and Partial Clique Cover. Note that data reduction rules, like Rule 2, may also change the parameter k, but this is not essential for confluence and will be disregarded in this section.

[3] Twin classes and the twin graph have been used before for data reduction under the names *critical cliques* and *critical clique graph* (see e. g. [15]).

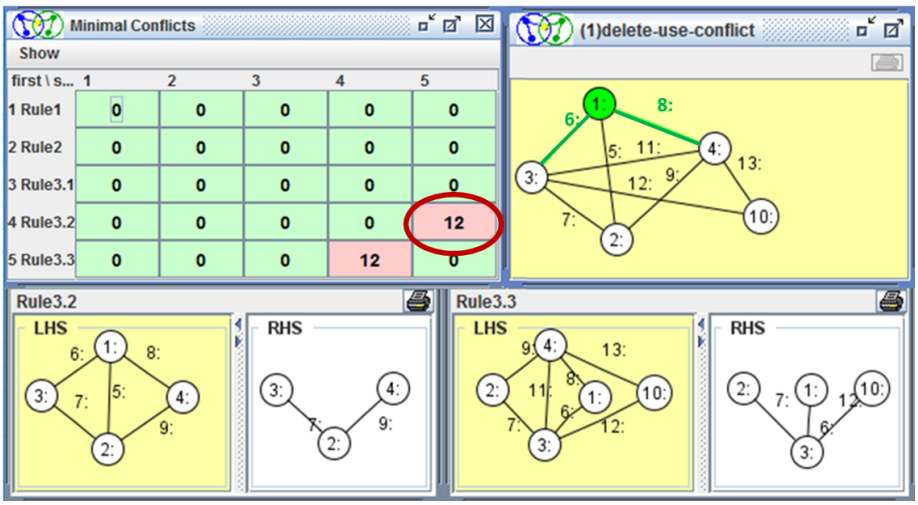
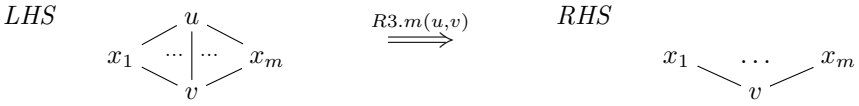


Figure 2: CP table for Clique Cover Rule 1 to Rule 3.3, and one critical pair in detail

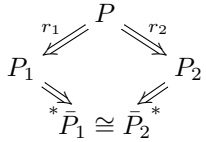
Actually, Rule 3 is a rule scheme in the sense of graph transformation theory, which can be represented by the following family of rules $R3.m$ for $m \geq 1$:



The rule describes the deletion of u. Applying the rule to a graph G means to find an occurrence of the left-hand side LHS in G satisfying $N[u] = N[v] = \{u, v, x_1, \ldots, x_m\}$, and to replace this occurrence by the right-hand side RHS.

For graphs with n vertices, we only have to consider rules Rule 1, Rule 2, Rule 3.1, ..., Rule 3.r with $r = n - 2$, because rules with $r > n - 2$ cannot be applied. Fig. 2 shows the table computed by the AGG tool [1] giving the number of critical pairs (CP) for each pair of rules and $r = 3$. In the AGG tool, when clicking on an entry in the CP table (e. g. the highlighted field showing twelve minimal conflicts for Rule 3.2 and Rule 3.3 where rule Rule 3.2 is applied first), the twelve conflicting situations of these two rules are shown in detailed graphical views. Vertices and edges in the rules (in the bottom of Fig. 2) are numbered to define their conflicting overlapping situation. We can see one of the twelve conflicts in the overlapping graph P in the upper right part of Fig. 2, where vertex 1 and edges 5, 6 and 8 shall be deleted by Rule 3.2, but vertex 1 and edges 6 and 8 are also needed for the application of Rule 3.3, which is supposed to delete vertex 4 and its incident edges.

For each critical pair $P_1 \overset{r_1}{\Longleftarrow} P \overset{r_2}{\Longrightarrow} P_2$ of the rule set in Fig. 2, we have shown strict confluence using AGG, essentially by applying the rules from the rule set as long as possible



to P_1 and to P_2, leading to reduced graphs $\bar{P}_1$ and $\bar{P}_2$, and showing that they are isomorphic, as indicated in the diagram.

The critical pairs can be computed automatically, and the reduction sequences $P_1 \overset{*}{\Longrightarrow} \bar{P}_1, P_2 \overset{*}{\Longrightarrow} \bar{P}_2$, and the isomorphism for $\bar{P}_1$ and $\bar{P}_2$ can be checked interactively using the tool AGG.

4 Case Study Partial Clique Cover

We provide a second, more demanding case study: Partial Clique Cover, a generalization of Clique Cover where some edges C are annotated as already covered, and only uncovered edges need to be covered by cliques.

> Partial Clique Cover
> **Instance:** An undirected graph $G = (V, E)$, a set $C \subseteq E$ of *covered* edges, and an integer $k \geq 0$.
> **Question:** Is there a set of at most k cliques in G such that each edge in $E \setminus C$ has both its endpoints in at least one of the selected cliques?

Partial Clique Cover was considered by Orlin [28]. A possible application is in pairwise testing for software features [6], that is, finding a set of configurations (tests) such that any two combinable features occur together in a test. This can be modeled as a clique cover problem where vertices are features, and an edge indicates that the two features are compatible. A covered edge here would indicate a combination of features that is legal, but does not need to be tested, for instance because it has been tested before or because it is not officially supported.

4.1 Kernelization for Partial Clique Cover

We now examine a more complex set of data reduction rules that can deal with Partial Clique Cover instances, where some edges are marked as covered. Note that while Gramm et al. [14] also deal with annotated graphs, some of their reduction rules (e. g. Rule 3') are only correct if the original input does not have any annotations, whereas here we allow any set of initially covered edges.

We generalize Rules 1 and 2 in a canonical way.

Rule 4 ([14, Rule 1])**.** *Remove isolated vertices and vertices that are only incident to covered edges.*

Rule 5. *If there is an isolated edge, then delete it and, if the edge was not covered, then decrease the parameter by one.*

We then adapt Rule 3 as follows:

Rule 6. *Let u, v be twins. Mark all edges incident to u as covered if the following covering conditions hold:*

$$\forall x \in V \setminus \{u, v\} : \{u, x\} \in C \iff \{v, x\} \in C, \tag{CC1}$$

$$\{u, v\} \notin C \Rightarrow \exists x \in V \setminus \{u, v\} : \{v, x\} \notin C. \tag{CC2}$$

The correctness of Rules 4 and 5 is easy to see; we now prove correctness of Rule 6. For Partial Clique Cover, a yes-instance $I = (G, C, k)$ means that there is a set of at most k cliques in the graph $G = (V, E)$ with covered edges C such that each edge in $E \setminus C$ has both its endpoints in at least one of the selected cliques.

Proposition 1. *Rule 6 is correct.*

Proof. Let $I = (G, C, k)$ be the original instance and $I' = (G, C', k)$ the instance after one application of Rule 6. If there is a clique cover $\mathcal{K} = (K_1, \ldots, K_k)$ for I, then clearly $\mathcal{K}$ is also a solution for I'. Conversely, if there is a clique cover $\mathcal{K}' = (K'_1, \ldots, K'_k)$ for I', we can construct a clique cover $\mathcal{K} = (K_1, \ldots, K_k)$ for I by setting $K_i = K'_i \cup \{u\}$ if $v \in K'_i$ and $K_i = K'_i$ otherwise, for $1 \leq i \leq k$. Since u and v are twins, each K_i is in fact a clique. It remains to show that each edge in $E \setminus C$ is covered by $\mathcal{K}$. All uncovered edges except those incident to u are already covered by $\mathcal{K}'$. For each uncovered edge $\{u, x\} \in E, \{u, x\} \notin C$ and $x \neq v$, by (CC1), we have $\{v, x\} \notin C$. Hence also $\{v, x\} \notin C'$, thus there is a clique $K'_i \in \mathcal{K}'$ with $\{v, x\} \subseteq K'_i$. This implies $v \in K'_i$ and using $K_i = K'_i \cup \{u\}$ also $\{u, x\} \subseteq K_i$. Finally, if $\{u, v\} \notin C$, then by (CC2) there is some $x \in V, x \neq u, v$ with $\{v, x\} \notin C$ and hence also $\{v, x\} \notin C'$, thus $\{v, x\} \subseteq K'_i$ for some $K'_i \in \mathcal{K}'$, and we have $\{u, v\} \subseteq K_i$ for similar reasons. □

However, if we drop either (CC1) or (CC2), then the rule is not correct, as we show by counter-examples.

Proposition 2. *Rule 6 without* (CC1) *or* (CC2) *is not correct.*

Proof. We give counter-examples: Consider a graph with three vertices u, v, w which form a clique. If $\{u, v\}$ and $\{v, w\}$ are covered and $\{u, w\}$ is uncovered, then Rule 6 without (CC1) would allow to mark $\{u, w\}$ as covered. However, the resulting instance is a yes-instance for $k = 0$, while the original instance is not. Similarly, consider the case that $\{u, w\}$ and $\{v, w\}$ are marked covered, but $\{u, v\}$ is not. Then, Rule 6 without (CC2) would allow to mark $\{u, v\}$ as covered. Again, the resulting instance is a yes-instance for $k = 0$, while the original instance is not. □

It is not hard to see that Rules 4 to 6 yield a problem kernel for Clique Cover, that is, for instances without covered edges.

Proposition 3. *Rules 4 to 6 yield a problem kernel for* Clique Cover *with at most* 2^k *vertices.*

Proof. If the original instance has no covered edges, new ones are only introduced by Rule 6. In this rule, all edges incident to one vertex are marked as covered. We can assume that this vertex then gets immediately deleted by Rule 4, since no other change can destroy this reduction opportunity. Thus, Rule 4 is equivalent to Rule 1, and Rule 5 is equivalent to Rule 2. Furthermore, (CC1) is always satisfied, and (CC2) is satisfied unless u and v form an isolated edge. Thus, Rule 6 is equivalent to Rule 3. The kernel bound now follows from Theorem 1. □

Unfortunately, the new rules do not yield a problem kernel for Partial Clique Cover with respect to the parameter k. In fact, we can show that Partial Clique Cover is already NP-complete for $k = 3$, and thus cannot have a problem kernel unless P = NP.

Theorem 3. Partial Clique Cover *is NP-complete for* $k \geq 3$.

Proof. We first describe a polynomial-time many-to-one reduction from 3-Coloring to the problem of covering vertices by disjoint cliques, and then reduce this problem to Partial Clique Cover. 3-Coloring is a well-known NP-hard problem [12], which asks for a coloring of the vertices of a graph with three colors such that no two adjacent vertices have the same color. From an instance $G = (V, E)$ of 3-Coloring, first construct the complement graph $\bar{G} = (V, \bar{E})$ with $\bar{E} = \{\{u, v\} \mid u, v \in V, \{u, v\} \notin E\}$. The task is now to find three vertex-disjoint cliques in $\bar{G}$ that cover all vertices. We then expand each vertex to an edge, that is, we construct a graph $G' = (V', E')$ with $V' = \{v_1 \mid v \in V\} \cup \{v_2 \mid v \in V\}$ and

$$E' = \{\{v_1, v_2\} \mid v \in V\} \cup \{\{v_i, w_j\} \mid \{v, w\} \in \bar{E}, i, j \in \{1, 2\}\}.$$

Our Partial Clique Cover instance then consists of G' with $C = E' \setminus \{\{v_1, v_2\} \mid v \in V\}$ and $k = 3$, that is, only the edges corresponding to a vertex in G need to be covered.

We now need to show that G has a 3-coloring iff there are three cliques in G' that cover all edges in C. If there is a 3-coloring of G, then we can cover the vertices of $\bar{G}$ with three cliques K_1, K_2, K_3. From a clique $K_i, 1 \leq i \leq 3$, we can construct a clique $K'_i = \{v_1 \mid v \in K_i\} \cup \{v_2 \mid v \in K_i\}$ in G'. It is easy to see that K'_1, K'_2, K'_3 is indeed a clique cover of size 3 for G' that covers every edge in $E' \setminus C$. Conversely, assume that we have three cliques K'_1, K'_2, K'_3 in G' that cover all edges in $E' \setminus C$. First, if only one vertex of v_1 and v_2 for some $v \in V$ is contained in some $K'_i, 1 \leq i \leq 3$, then we omit this vertex from K'_i. Then, if the two vertices of an edge of $E' \setminus C$ are contained in more than one clique, omit them from all but one of these cliques. Clearly, after these operations we still have a clique cover for $E' \setminus C$, and each edge in $E' \setminus C$ occurs in exactly one of the three cliques. We can then construct three cliques $K_i := \{v \mid v_1 \in K'_i\}$, $1 \leq i \leq 3$, which are disjoint and cover all vertices in $\bar{G}$. This yields a 3-coloring of G. The construction easily generalizes to $k > 3$. □

Adding the further parameter number $c = |C|$ of covered edges, we can show a kernel for Partial Clique Cover with respect to the combined parameter (k, c).

Theorem 4. *Rules 4 to 6 yield a problem kernel for* Partial Clique Cover *with at most* 2^{k+c} *vertices.*

Proof. The idea of the proof is to show that if a Partial Clique Cover instance has more than 2^{k+c} vertices, then we can construct a Clique Cover instance, find a data reduction opportunity there using Rules 1 to 3, and using

this also find a data reduction for the Partial Clique Cover instance; in this way, we can use the bounds from Theorem 1.

Given an instance $I = (G = (V, E), C, k)$ of Partial Clique Cover, we construct a graph $\bar{G} = (\bar{V}, \bar{E})$ for Clique Cover with $\bar{V} = V \cup \{x_e \mid e \in C\}$ and $\bar{E} = E \cup \bigcup_{e=\{u,v\} \in C} \{\{u, x_e\}, \{x_e, v\}\}$. That is, for each covered edge we add a new vertex and connect it to the endpoints of the edge to form a triangle.

We claim that if I is a yes-instance for Partial Clique Cover, then $\bar{I} = (\bar{G}, k + c)$ is a yes-instance for Clique Cover. To see this, consider a solution $C_1, \ldots, C_k$ for I. It covers all edges of $\bar{G}$ except for the newly added edges. These can clearly be covered by adding c triangles, each containing an edge in C and the two corresponding new edges. Hence, we get a clique cover of size $k + c$.

Thus, if I is a yes-instance with more than 2^{k+c} vertices, then $\bar{I}$ is also a yes-instance with more than 2^{k+c} vertices, and thus by Theorem 1, at least one of Rules 1 to 3 applies to $\bar{I}$. If Rule 1 or Rule 2 applies, then it is easy to see that Rule 4 or Rule 5 applies to I, respectively. We claim further that if Rule 3 applies to $\bar{I}$, then also one of Rules 4 to 6 applies to I. Showing this claim completes the proof of the theorem.

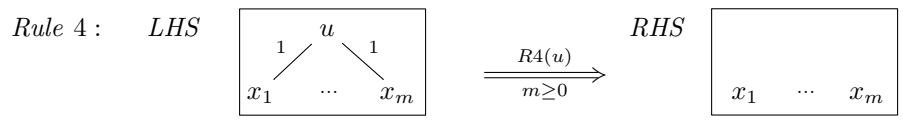
To see the claim, note that if Rule 3 applies to $\bar{I}$, then there is an edge $\{u, v\}$ which is not isolated such that $N_{\bar{G}}[u] = N_{\bar{G}}[v]$. If one of u and v, say u, is a new vertex $x_{\{v,w\}}$ added for a covered edge $\{v, w\} \in C$, then we have $N_{\bar{G}}[u] = \{u, v, w\} = N_{\bar{G}}[v]$ and thus $N_G[v] = \{v, w\}$, that is, in G vertex v is only incident to the covered edge $\{v, w\}$, and Rule 4 applies.

It remains to consider the case that $u, v \in V$. Clearly $N_{\bar{G}}[u] = N_{\bar{G}}[v]$ implies $N_G[u] = N_G[v]$, since G is an induced subgraph of $\bar{G}$. If there is no $w \in V, w \neq v$ with $\{u, w\} \in E \setminus C$, then $\{u, v\}$ is an isolated edge and Rule 5 applies. Otherwise we have $w \in V$, $w \neq v$ with $\{u, w\} \in E \setminus C$, but for no such w can $\{u, w\}$ be covered, since $x_{\{u,w\}}$ would be in $N_{\bar{G}}[u]$ but not in $N_{\bar{G}}[v]$. Thus, the covering conditions (CC1) and (CC2) are fulfilled and Rule 6 applies. □

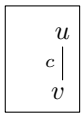
4.2 Confluence of Data Reduction for Partial Clique Cover

We give a direct proof for the confluence of the Partial Clique Cover data reduction rules, using local confluence and a somewhat involved case distinction. The challenge is that we cannot use the twin graph anymore as in Theorem 2, since it might be required for an optimal solution to cover twins with different cliques.

In the following we use the graphical notation for Rules 4 to 6 as usual in graph transformation theory [10]. An edge marked by "1" means that this edge is covered.

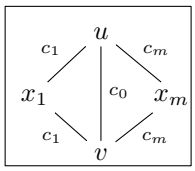


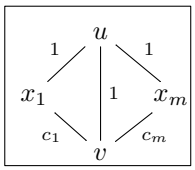
A match of Rule 4 in G is valid if $x_1, \ldots, x_m$ are exactly all neighbors of u in G.

Rule 5 : *LHS* 

$\xRightarrow[c\in\{0,1\}]{R5(u,v)}$ *RHS* 

A match of Rule 5 in G is valid if $\{u, v\}$ is an isolated edge in G, where $c = 0$ means that $\{u, v\}$ is uncovered, and $c = 1$ means that it is covered.

Rule 6 : *LHS* 

$\xRightarrow[m\geq 1,\ c_i\in\{0,1\}]{R6(u,v)}$ *RHS* 

A match of Rule 6 is valid if $\{u, v\}$ are twins, but $\{u, v\}$ is not an isolated edge ($m \geq 1$), and the match of *LHS* of Rule 6 in G contains exactly all vertices in $N[u] = N[v]$. Moreover, $c_0 = 0$ implies $c_i = 0$ for some $i \leq m$ (see covering conditions (CC1) and (CC2) of Rule 6).

We note that a confluent kernelization for Partial Clique Cover can also be obtained by taking Rules 1 to 3, restricting their applicability to instances without covered edges, and adding a rule that replaces covered edges by a triangle consisting of uncovered edges, as in the proof of Theorem 4. However, this does not serve our main goal of illustrating the use of tools from graph transformation theory in concrete rule analysis.

Theorem 5. *Rules 4 to 6 for* Partial Clique Cover *are confluent.*

Proof. According to a general result for rewriting systems [18, 25], confluence follows from local confluence and termination. Rule 4 and Rule 5 are terminating, because the number of vertices is strictly reduced for each rule application. For Rule 6, the number of uncovered edges is strictly reduced, unless all edges incident to u are covered already. In this case, the application of Rule 6 leads to an identical transformation which can be neglected for termination. It remains to show that Rules 4 to 6 ($R4$ to $R6$ for short) are locally confluent.

This requires to show local confluence of $G_1 \overset{r_1}{\Longleftarrow} G \overset{r_2}{\Longrightarrow} G_2$, where $r_1, r_2 \in \{R4, R5, R6\}$. Local confluence of $(R4, R4)$, $(R5, R5)$, $(R4, R5)$, and $(R5, R6)$ is easy to see, and for symmetry reasons also for $(R5, R4)$ and $(R6, R5)$.

1. $(R4, R4)$**: Local confluence of Rule 4 with itself.**

   Local confluence of $G_1 \overset{R4(u)}{\Longleftarrow} G \overset{R4(u')}{\Longrightarrow} G_2$ with $u \neq u'$ is obvious for $u' \notin N[u]$ and hence, $u \notin N[u']$ because both rule applications are parallel independent. For $u' \in N[u]$ we have w.l.o.g. $u' = x_1$, leading to the local confluence diagram in Fig. 3, where we only show the essential vertices and edges of G, G_1, G_2, and G_3.

   Note that $N[u]$ in the match of $R4(u)$ to G is different from $N[u]$ for $R4[u]$ applied to G_2 and similarly for $N[u']$.

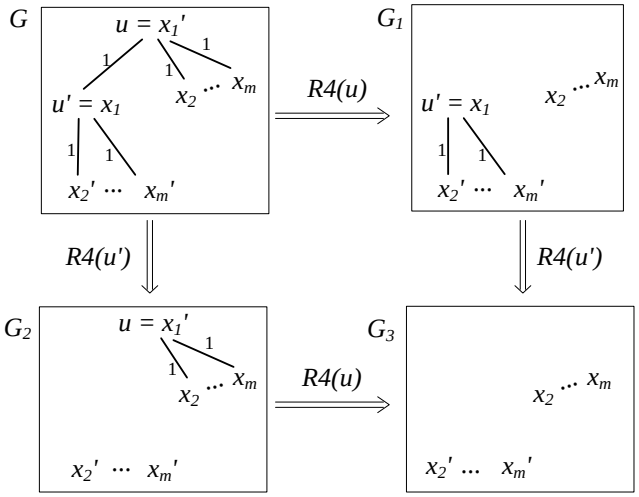


Figure 3: Local confluence of Rule 4 with itself

2. $(R5, R5)$**: Local confluence of Rule 5 with itself.**

   Local confluence of $G_1 \overset{R5(u,v)}{\Longleftarrow} G \overset{R5(u',v')}{\Longrightarrow} G_2$ for $\{u, v\} \neq \{u', v'\}$ is obvious because the isolated edges $\{u, v\}$ and $\{u', v'\}$ cannot overlap.

3. $(R4, R5)$**: Local confluence of Rule 4 with Rule 5.**

   Local confluence of $G_1 \overset{R4(u)}{\Longleftarrow} G \overset{R5(u',v')}{\Longrightarrow} G_2$ is obvious for $u \neq u', v'$ because the matches are disjoint. For $u = u'$ (and similarly for $u = v'$), we have $x_1 = v'$ leading to the local confluence in Fig. 4.

4. $(R6, R5)$**: Local confluence of Rule 6 with Rule 5.**
   Local confluence of $R6(u, v)$ with $R5(u', v')$ is obvious because the matches

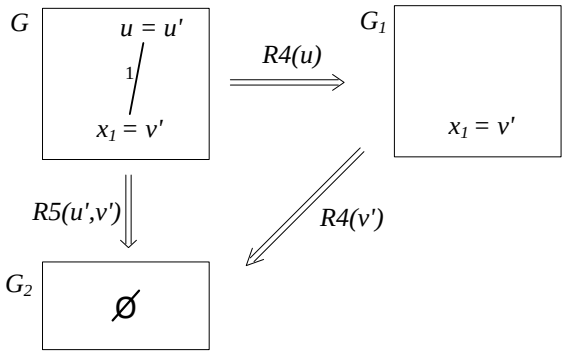


Figure 4: Local confluence of Rule 4 and Rule 5

are disjoint, i. e., $u', v' \notin N[u] = N[v]$.

It remains to show local confluence for the cases $(R6, R4)$ and $(R6, R6)$, more precisely, $(R6(u,v), R4(w))$ and $(R6(u,v), R6(u',v'))$, where (w), (u,v), and (u',v') are arbitrary matches for $R6$ resp. $R4$, but these cases are quite involved.

5. $(R6, R4)$: **Local confluence of Rule 6 with Rule 4** For local confluence $G_2 \overset{R6(u,v)}{\Longleftarrow} G \overset{R4(w)}{\Longrightarrow} G_1$ we have to consider the following four different subcases: (5.1) $w \notin N[u]$ and (5.2) $w \in N[u]$ with (5.2.1) $w = x_1$, (5.2.2) $w = u$, and (5.2.3) $w = v$.

   5.1 $w \notin N[u] = N[v]$.

   In this case we have parallel independence:

$$\begin{array}{ccc} G & \overset{R4(w)}{\Longrightarrow} & G_1 \\ {\scriptstyle R6(u,v)}\Downarrow & & \Downarrow{\scriptstyle R6(u,v)} \\ G_2 & \underset{R4(w)}{\Longrightarrow} & G_3 \end{array}$$

   $R4(w)$ is applicable to G_2 because covered edges $\{w, x\}$ in G are preserved as covered edges in G_2. The match of $R6(u,v)$ in G is given by $\{u,v\}$, $\{u,x_i\}$, $\{v,x_i\}$, $i = 1, \ldots, m$ and preserved in G_1 because $w \notin N[u] = N[v]$. This means $R6(u,v)$ is applicable to G_1. We obtain in both direction the same graph G_3, which is structurally equal to G_1, but with $\{u,x_i\} \in C$ for $i = 1, \ldots, m$.

   5.2 $w \in N[u] = N[v]$.

      5.2.1 $w = x_1$.

      For $m = |N[u]| = |N[v]| = 1$, we have local confluence by diagram (1) in Fig. 5 where $\{u,v\}$ is covered in G by condition (CC2) of Rule 6.

      For $m = |N[u]| = |N[v]| \geq 2$, we consider the diagram (2) in Fig. 5. Rule $R4(w)$ can be applied to G_2 because the match of $R4(w)$ in G given by $\{w,u\}$, $\{w,v\}$, $\{w,y_1\}$, $\ldots$, $\{w,y_j\}$ is still available in G_2. Moreover, $R6(u,v)$ can be applied to G_1 because we can show properties (CC1) and (CC2) of Rule 6, where (CC1) splits into the two parts for "$\Rightarrow$" and for "$\Leftarrow$".

      (CC1),$\Rightarrow$ For all $x \neq u, v$ and $\{u,x\} \notin C$ in G_1, we have $x \neq x_1 = w$ and $\{u,x\} \notin C$ in G. Applying $R6(u,v)$ to G, we have $\{v,x\} \notin C$ in G and thus also $\{v,x\} \notin C$ in G_1 obtained by applying $R4(w)$ to G.

      (CC1),$\Leftarrow$ For all $x \neq u, v$ and $\{u,x\} \in C$ in G_1, we have $x \neq x_1$ because $x_1 \notin G_1$. This implies $\{u,x\} \in C$ in G, and we have $\{v,x\} \in C$ in G, and therefore $\{v,x\} \in C$ in G_1.

      (CC2) Let $\{u,v\} \notin C$ in G_1. Then $\{u,v\} \notin C$ in G. By (CC2) valid for the match of $R6(u,v)$ in G (which is essentially the same

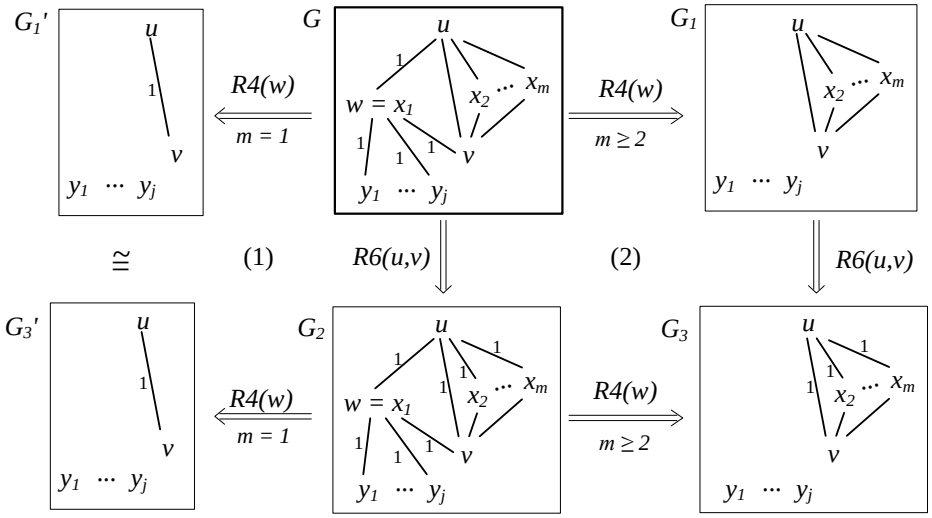


Figure 5: Local confluence of Rule 6 and Rule 4 for Case 5.2.1: $w = x_1$

match now, reduced only by node x_1 and its incident edges), we have $\{v, x\} \notin C$ for some $x \neq x_1$ because $\{v, x_1\} \in C$. This implies $\{v, x\} \notin C$ in G_1.

5.2.2 $w = u$.
In this case we have the situation depicted in Fig. 6, which shows local confluence. Note that all neighbors of w in G are also neighbors of v because $u = w$ and $N[u] = N[v]$.

5.2.3 $w = v$.
In Fig. 7, we have a similar picture as above showing local confluence for $w = v$. Note that we have $c_1 = \cdots = c_m = 1$, leading to the equality $G_1 = G_3$, which follows from condition (CC1) for $R6(u, v)$ applied to G. In fact, we need ((CC1),⇒) and ((CC1),⇐) since the condition ((CC1),⇒) is not sufficient.

6. $(R6, R6)$: **Local confluence of Rule 6 with itself.**

For local confluence of $G_2 \overset{R6(u',v')}{\Longleftarrow} G \overset{R6(u,v)}{\Longrightarrow} G_1$, we have to consider again four different subcases: (6.1) $\{u, v\} \cap \{u', v'\} = \emptyset$ with (6.1.1) $u', v' \notin N[u]$, and (6.1.2) $u', v' \in N[u]$, and (6.2) $\{u, v\} \cap \{u', v'\} \neq \emptyset$ with (6.2.1) $u' = v$ and $v' \neq u$, and (6.2.2) $u = v'$, and $v = u'$.

6.1 $\{u, v\} \cap \{u', v'\} = \emptyset$.

6.1.1 $u', v' \notin N[u] = N[v]$.
In this case, we also have $u, v \notin N[u'] = N[v']$ and G, G_1, G_2, and G_3 are structurally equal, but they differ in the covering of

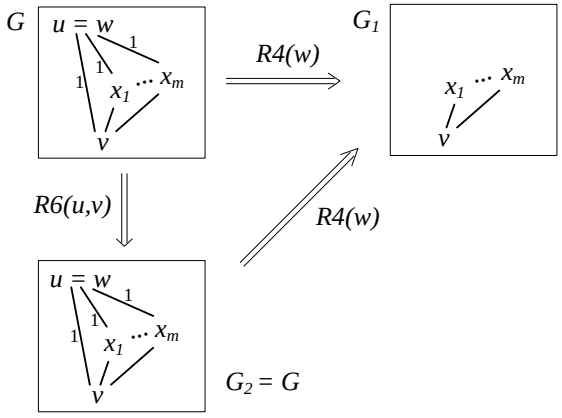


Figure 6: Local confluence of Rule 6 and Rule 4 for Case 5.2.2: $w = u$

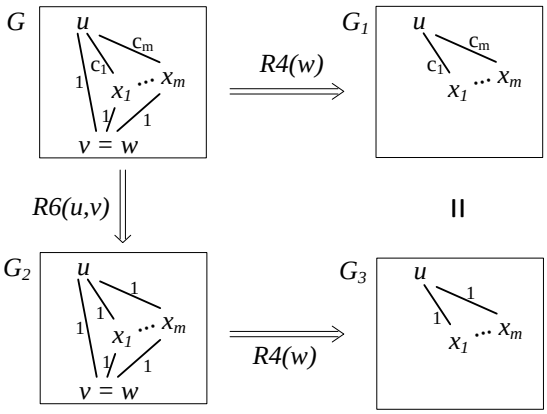


Figure 7: Local confluence of Rule 6 and Rule 4 for Case 5.2.3: $w = v$

edges:

$$\begin{array}{ccc} G & \xRightarrow{R6(u,v)} & G_1 \\ {\scriptstyle R6(u',v')}\Downarrow & & \Downarrow{\scriptstyle R6(u',v')} \\ G_2 & \xRightarrow[R6(u,v)]{} & G_3 \end{array}$$

Let $R6(u,v)$ be applicable to G with $\{u,v\},\{u,x_i\},\{v,x_i\}, i = 1,\ldots,m$, $m \geq 1$. Moreover, let $R6(u',v')$ be applicable to G with $\{u',v'\},\{u',x'_i\}$, $\{v',x'_i\}$, $i = 1,\ldots,m'$, $m' \geq 1$. According to our assumption $u',v' \notin N[u] = N[v]$, we have for $m,m' \geq 1$:

$$\{\{u,v\},\{u,x_i\},\{v,x_i\} \mid i = 1,\ldots,m\} \\ \cap \{\{u',v'\},\{u',x'_i\},\{v',x'_i\} | i = 1,\ldots,m'\} = \emptyset \quad (*)$$

Due to symmetry, it is sufficient to show that $R6(u,v)$ is applicable to G_2. Using (*), we can show that conditions (CC1) and (CC2) hold, and hence $R6(u,v)$ is applicable to G_2:

(CC1),$\Rightarrow$ For all $x \neq u,v$ in G_2 with $\{u,x\} \notin C \Rightarrow \{u,x\} \notin C$ in G. Hence, by $(1a)$ for G: $\{v,x\} \notin C \Longrightarrow \{v,x\} \notin C$ in G_2, using (*).

(CC1),$\Leftarrow$ For all $x \neq u,v$ in G_2 with $\{u,x\} \in C \Rightarrow \{u,x\} \in C$ in G, using (*). We have $\{u,x\} \in C$ in $G \Rightarrow \{v,x\} \in C$ in G_2.

(CC2) Given $\{u,v\} \notin C$ in $G_2 \Rightarrow \{u,v\} \notin C$ in G.
By (CC2) valid for the match of $R6(u,v)$ in G, we have $x \neq u,v$ with $\{v,x\} \notin C$ in $G \Longrightarrow \{v,x\} \notin C$ in G_2, using (*).

This implies that $R6(u,v)$ can be applied to G_2 and, due to symmetry, $R6(u',v')$ can be applied to G_1, leading to the same G_3 in both cases. Graph G_3 is structurally equal to G, G_1, and G_2, with all edges in both sets of (*) covered in G_3.

6.1.2 $u',v' \in N[u] = N[v]$.
In this case, we also have $u,v \in N[u'] = N[v']$ and G, G_1, G_2, and G_3 are structurally equal leading to the local confluence diagram in Fig. 8 using the notation of Case 3.1, where w.l.o.g. $u' = x_1, v' = x_2, u = x'_1$, and $v = x'_2$. Again, we only show the essential parts of G, G_1, G_2, G_3.
Due to symmetry, it is sufficient to show that $R6(u,v)$ applied to G_2 satisfies (CC1) and (CC2).

(CC1),$\Rightarrow$ Let $x \neq u,v$ in G_2 with $\{u,x\} \notin C \Rightarrow x \neq x_1$ and $x = x_2 = v'$ or $x = x_m$. Moreover, $\{u,x\} \notin C$ in $G \Rightarrow \{v,x\} \notin C$ in G by (CC1) for $G \Rightarrow \{v,x\} \notin C$ in G_2 because $x \neq x_1$.

(CC1),$\Leftarrow$ Let $x \neq u,v$ in G_2 with $\{u,x\} \in C$. If $\{u,x\} \in C$ also in G we have also $\{v,x\} \in C$ in G and hence also $\{v,x\} \in C$ in G_2. Otherwise $\{u,x\} \notin C$ in G and $\{u,x\} \in C$ in G_2 implies $\{u,x\} = \{u,u'\}$ and $x = u'$, because only edges incident to u' become covered in G_2. Hence $\{v,x\} = \{v,u'\} \in C$ in G_2 after application of $R6(u',v')$ to G.

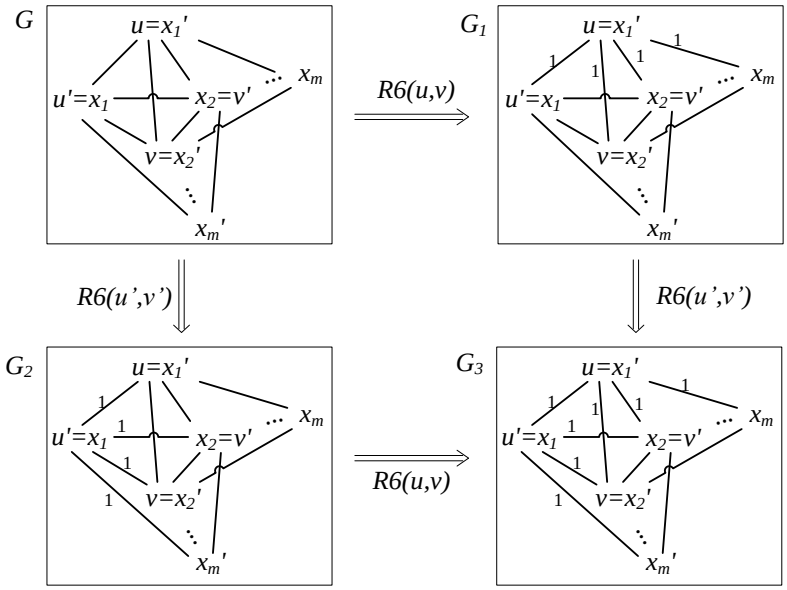


Figure 8: Local confluence of Rule 6 and Rule 6 for Case 6.1.2: $\{u,v\} \cap \{u',v'\} = \emptyset$ and $u', v' \in N[u] = N[v]$

(CC2) Let $\{u,v\} \notin C$ in G_2. This implies that $\{u,v\} \notin C$ in G. By (CC2) valid for the match of $R6(u,v)$ in G, we have $x \neq u, v$ with $\{v,x\} \notin C$ in G. If $\{v,x\} \notin C$ in G_2, we are done. Otherwise $\{v,x\} \in C$ in G_2 is only possible if $x = u'$. This implies that $\{v,u'\} = \{u',v\} \notin C$ in G by $\{v,x\} \notin C$ in G. $(1a)$ for G implies $\{v',v\} \notin C$ in G with $v, v' \neq u'$. This implies that $\{v,x\} = \{v,v'\} \notin C$ in G_2, because only edges incident to u' become covered in G_2.

6.2 $\{u,v\} \cap \{u',v'\} \neq \emptyset$.

6.2.1 $u' = v$ and $v' \neq u$.
In this case, we have $N[u] = N[v] = N[u'] = N[v']$ leading to the local confluence diagram in Fig. 9 with $c_i, d_i \in \{0,1\}$ where w.l.o.g. $v' = x_1 = x'_1, \ldots, x_m = x'_m$. We have that G_1 is isomorphic to G_2 ($G_1 \cong G_2$) if $c_1 = d_1, \ldots, c_m = d_m$. This follows from condition (CC1) for $R6(u,v)$ applied to G.

6.2.2 $u = v'$ and $v = u'$.
Note that $(u,v) \neq (u',v')$ excludes the case $u = u'$ and $v = v'$. Similarly to Case 6.2.1, we get the local confluence diagram in Fig. 10. Mapping u to v and v to u we have that G_1 is isomorphic to G_2 if $c_1 = d_1, \ldots, c_m = d_m$. This follows as above from condition (CC1) for $R6(u,v)$ applied to G.

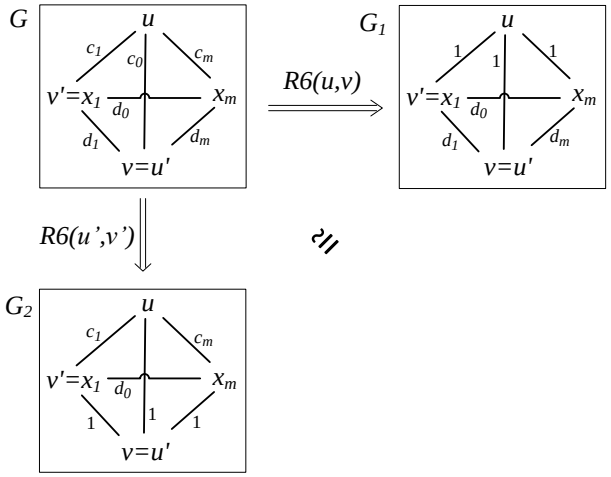


Figure 9: Local confluence of Rule 6 and Rule 6 for Case 6.2.1: $\{u, v\} \cap \{u', v'\} \neq \emptyset$ with $u' = v$ and $v' \neq u$

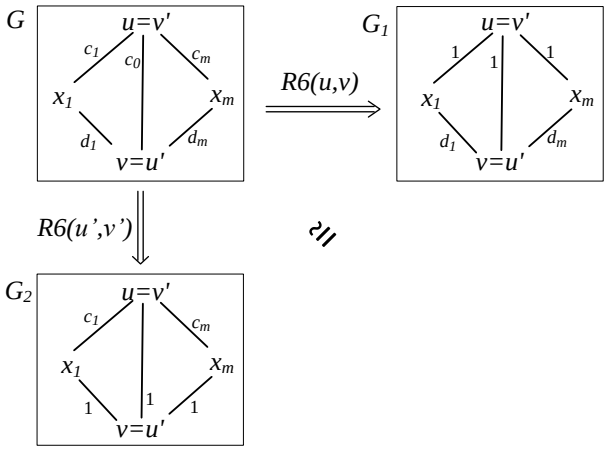


Figure 10: Local confluence of Rule 6 and Rule 6 for Case 6.2.2: $\{u, v\} \cap \{u', v'\} \neq \emptyset$ with $u = v'$ and $v = u'$

This completes the proof for Theorem 5. □

4.3 Confluence via Critical Pair Analysis

Since the confluence proof for Rules $R4$ to $R6$ in Theorem 5 is quite complex, we discuss an alternative approach via critical pair analysis. Similarly to Rule 3 in Fig. 2 of Section 3, we now consider Rules $R4$ to $R6$. Here, $R6$ has in addition to parameter r the parameters $c_0, c_1, \ldots, c_r \in \{0, 1\}$, where $c_i = 1$ means that the corresponding edge is covered. Note that covering condition (CC1) of $R6$ requires the same covering of edges $\{u, x_i\}$ and $\{v, x_i\}$ for $i = 1, \ldots, r$, and if $c_0 = 0$, then covering condition (CC2) requires $c_i = 0$ for some $i \in \{1, \ldots, r\}$. For $r = 1$, this leads to rule instances $R6.1.i$ ($i = 1, 2, 3$), for $r = 2$ to $R6.2.i$ ($i = 1, \ldots, 4$) and for $r = 3$ to $R6.3.i$ ($i = 1, \ldots, 7$). The critical pair table for $r = 1, 2, 3$ (up to isomorphism) is shown in Fig. 11 as a screenshot from the AGG tool [1], giving the number of critical pairs for each pair of rules for $r = 1, 2, 3$.

In the table in Fig. 11, we have e. g. seven instances of $R6$ for $r = 3$ with differently covered edges. For example, "Rule6.3.1.1" is the instance of $R6$ with $|N[u]| = |N[v]| = 3$, where the first "1" denotes the covering situation (explained below) for edges incident to u (except $\{u, v\}$), and the second "1" meaning that $\{u, v\}$ is covered (we would have "0" in case $\{u, v\}$ is uncovered). For the coverings of edges incident to u, we distinguish case 1 (all edges are uncovered), case 2 (one edge is covered), ..., and case $r + 1$ (all edges are covered). As described in Section 3.3, the AGG tool supports the inspection of reasons for conflicts by visualizing the critical overlapping regions of two conflicting rule applications. Hence, when an AGG user selects a rule pair by clicking on a field in the critical pair table (e. g. the highlighted field showing twelve minimal conflicts for 6:Rule6.2.1.0 and 12:Rule6.3.1.1 where rule 6:Rule6.2.1.0 is applied first), the twelve conflicting situations of these two rules are shown in detailed graphical views. Vertices and edges in the rules (shown in the bottom of Fig. 11) are numbered to define their conflicting overlapping situation. We can see one of the twelve conflicts in the overlapping graph in the lower right part of Fig. 11, where edges 5, 6, and 9 are uncovered and shall be marked as covered by application of Rule6.2.1.0, but edge 9 also needs to be uncovered for the application of Rule6.3.1.1, which is supposed to mark edges 8, 9, and 12 as covered. We have a *change-attribute* conflict here, because both rules want to access and change the same "attribute" $c = 0$ of edge 9.

Similarly to the case study in Section 3, we have shown confluence of the critical pairs in Fig. 11 using AGG, where the confluence analysis has been completed for overlapping graphs with at most five vertices. For a growing number (and size) of rule instances, also the number of critical pairs to be analyzed is growing. A more efficient technique based on critical pairs for rule schemes should be subject of future research.

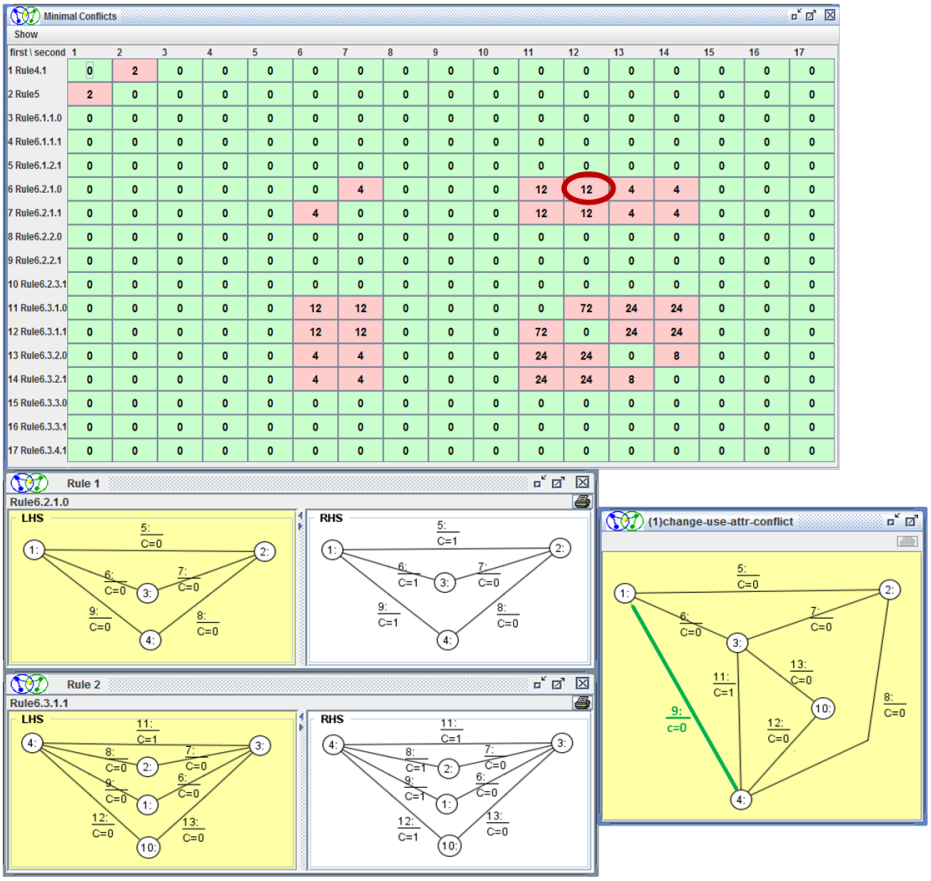


Figure 11: Critical pair table for Partial Clique Cover Rules 4 to 6.3.4.1, and one critical pair in detail

5 Discussion and Future Work

Seemingly for the first time, our work establishes a fruitful link between graph transformation theory and the theory of kernelization from parameterized algorithmics. While considering (comparatively simple) kernelizations for Clique Cover and Partial Clique Cover, already several theoretical and technical challenges popped up when proving confluent kernelizations. We believe that to analyze whether a set of data reduction rules is confluent is a well-motivated and natural theoretical question of practical relevance with the potential for numerous opportunities for (interdisciplinary) future research between so far unrelated research communities.

As to research questions that are more rooted in graph transformation theory, it is first important to extend the theory of critical pair analysis to undirected graphs. We are confident that this works not only for the examples in this paper.

Moreover, it is an important challenge to extend critical pair analysis from rules considering a constant-size subgraph to so-called rule schemes with unbounded number of vertices, that is, to transfer the "amalgamation" [3] of rewriting rules to this new context.

As to research on confluent kernelization rooted more in algorithmics, it appears to be of general interest to investigate how confluent problem kernels may help in deriving both upper and lower bounds for problem kernel sizes. In addition, it remains to study how confluence may contribute to speeding up kernelization algorithms and how the knowledge of having a uniquely determined problem kernel can help subsequent solution strategies that build on top of the kernel. Finally, studying confluence of data reduction and kernelization beyond graph problems, for example for string or set problems, remains a future task.

Acknowledgment. We thank Jiong Guo (Universität des Saarlandes) for pointing out Theorem 3.

References

[1] AGG. *Attributed Graph Grammar Tool*. TU Berlin, 2012. http://www.tfs.tu-berlin.de/agg. 4, 5, 6, 9, 22

[2] H. L. Bodlaender. Kernelization: New upper and lower bound techniques. In *Proc. 4th International Workshop on Parameterized and Exact Computation (IWPEC '09)*, volume 5917 of *LNCS*, pages 17–37. Springer, 2009. 2, 5

[3] P. Böhm, H.-R. Fonio, and A. Habel. Amalgamation of graph transformations: a synchronization mechanism. *Journal of Computer and System Sciences*, 34:377–408, 1987. 24

[4] M.-S. Chang and H. Müller. On the tree-degree of graphs. In *Proc. 27th International Workshop on Graph-Theoretic Concepts in Computer Science (WG '01)*, volume 2204 of *LNCS*, pages 44–54. Springer, 2001. 7

[5] M. Cygan, S. Kratsch, M. Pilipczuk, M. Pilipczuk, and M. Wahlström. Clique cover and graph separation: New incompressibility results. In *Proc. 39th International Colloquium on Automata, Languages, and Programming (ICALP '12)*, volume 7391 of *LNCS*, pages 254–265. Springer, 2012. 7

[6] P. Danziger, E. Mendelsohn, L. Moura, and B. Stevens. Covering arrays avoiding forbidden edges. *Theoretical Computer Science*, 410(52):5403–5414, 2009. 7, 10

[7] R. G. Downey and M. R. Fellows. *Parameterized Complexity*. Springer, 1999. 2, 5

[8] H. Ehrig, M. Pfender, and H. Schneider. Graph grammars: an algebraic approach. In *Proc. IEEE Symposium on Switching and Automata Theory*, pages 167–180. IEEE, 1973. 3

[9] H. Ehrig, H.-J. Kreowski, U. Montanari, and G. Rozenberg, editors. *Handbook of Graph Grammars and Computing by Graph Transformation. Vol 3: Concurrency, Parallelism and Distribution*. World Scientific, 1999. 4

[10] H. Ehrig, K. Ehrig, U. Prange, and G. Taentzer. *Fundamentals of Algebraic Graph Transformation*. EATCS Monographs in Theoretical Computer Science. Springer, 2006. 2, 4, 5, 6, 8, 13

[11] J. Flum and M. Grohe. *Parameterized Complexity Theory*. Springer, 2006. 2, 5

[12] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979. 7, 12

[13] J. Gramm, J. Guo, F. Hüffner, R. Niedermeier, H.-P. Piepho, and R. Schmid. Algorithms for compact letter displays: Comparison and evaluation. *Computational Statistics & Data Analysis*, 52(2):725–736, 2007. 7

[14] J. Gramm, J. Guo, F. Hüffner, and R. Niedermeier. Data reduction and exact algorithms for clique cover. *ACM Journal of Experimental Algorithmics*, 13: 2.2:1–2.2:15, 2008. 4, 7, 8, 10

[15] J. Guo and R. Niedermeier. Invitation to data reduction and problem kernelization. *ACM SIGACT News*, 38(1):31–45, 2007. 2, 5, 8

[16] A. Gyárfás. A simple lower bound on edge coverings by cliques. *Discrete Mathematics*, 85(1):103–104, 1990. 4, 7

[17] D. N. Hoover. Complexity of graph covering problems for graphs of low degree. *Journal of Combinatorial Mathematics and Combinatorial Computing*, 11:187–208, 1992. 7

[18] G. Huet. Confluent reductions: Abstract properties and applications to term rewriting systems. *Journal of the ACM*, 27(4):797–821, 1980. 4, 5, 6, 14

[19] E. Kellerman. Determination of keyword conflict. *IBM Technical Disclosure Bulletin*, 16(2):544–546, 1973. 7

[20] J. Kneis, D. Mölle, S. Richter, and P. Rossmanith. A bound on the pathwidth of sparse graphs with applications to exact algorithms. *SIAM Journal on Discrete Mathematics*, 23(1):407–427, 2009. 3

[21] L. T. Kou, L. J. Stockmeyer, and C.-K. Wong. Covering edges by cliques with regard to keyword conflicts and intersection graphs. *Communications of the ACM*, 21(2):135–139, 1978. 7

[22] J. van Leeuwen, editor. *Handbook of Theoretical Computer Science*. MIT Press, 1990. 2

[23] C. Lund and M. Yannakakis. On the hardness of approximating minimization problems. *Journal of the ACM*, 41(5):960–981, 1994. 7

[24] R. M. McConnell. Linear-time recognition of circular-arc graphs. *Algorithmica*, 37(2):93–147, 2003. 8

[25] M. H. A. Newman. On theories with a combinatorial definition of equivalence. *Annals of Mathematics*, 43(2):223–242, 1942. 4, 6, 14

[26] R. Niedermeier. *Invitation to Fixed-Parameter Algorithms*. Number 31 in Oxford Lecture Series in Mathematics and Its Applications. Oxford University Press, 2006. 2, 5

[27] I. Nor, D. Hermelin, S. Charlat, J. Engelstadter, M. Reuter, O. Duron, and M.-F. Sagot. Mod/resc parsimony inference: Theory and application. *Information and Computation*, 213:23–32, 2012. 7

[28] J. B. Orlin. Contentment in graph theory: Covering graphs with cliques. *Indagationes Mathematicae (Proceedings)*, 80(5):406–424, 1977. 7, 10

[29] D. Plump. Confluence of graph transformation revisited. In *Processes, Terms and Cycles: Steps on the Road to Infinity*, volume 3838 of *LNCS*, pages 280–308. Springer, 2005. 5

[30] G. Rozenberg. *Handbook of Graph Grammars and Computing by Graph Transformations, Volume 1: Foundations*. World Scientific, 1997. 2, 3

[31] K. Trifunovic, A. Cohen, D. Edelsohn, L. Feng, T. Grosser, H. Jagasia, R. Ladelsky, S. Pop, J. Sjödin, and R. Upadrasta. GRAPHITE two years after: First lessons learned from real-world polyhedral compilation. In *Proc. 2nd International Workshop on GCC Research Opportunities (GROW '10)*, pages 4–19, 2010. 7