Confluence in Data Reduction: Bridging Graph Transformation and Kernelization

Hartmut Ehrig, Claudia Ermel, Falk Hüffner*,
Rolf Niedermeier, and Olga Runge

Institut für Softwaretechnik und Theoretische Informatik, TU Berlin
{hartmut.ehrig, claudia.ermel, falk.hueffner,
rolf.niedermeier, olga.runge}@tu-berlin.de

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 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 Theoretical Computer Science [15]). 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, 11] and the theory of graph transformations [8, 20]: We employ the concept of confluence of rewriting systems to show "uniqueness results" for problem kernels. This leads to the natural concept of confluent data reduction rules, having a number of both theoretical and practical benefits as discussed in the following.

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

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, 11] 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, and a (*data*) *reduction rule* replaces in polynomial time a given problem instance I by an instance I' with $|I'| < |I|$. We say that the 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, 11]. This notion comes from the field of *parameterized complexity* [5, 9, 18], where 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 for a more formal definition).

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 reduction rules. Further, insight on the interaction of data reduction rules can lead to faster kernelizations. Confluence was also exploited by Kneis et al. [14], who showed that for their problem, one order of application of data reduction yields some desired property of the reduced instance, and another order yields a different desired property. A proof of confluence now shows that a reduced instance has both properties. 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 kernel.

If we allowed data reduction in kernelization with restriction of the rule execution order, we can force confluent kernelization in a trivial way by allowing only one execution order. In this paper, we avoid this trivial case by allowing any execution order.

Confluence of graph transformation systems. The theory of graph grammars and graph transformation systems has been started in the early 1970s [6] 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 [20], including logical and algebraic approaches. A graph transformation system consists of a set of graph rules, which 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. To apply a rule, a *match* is sought, that is, a subgraph in the input graph that is isomorphic to LHS. This subgraph without the intersection graph is then deleted, resulting in a context graph, which is glued together with RHS at the nodes and edges of the intersection graph. A graph transformation system is called *confluent* if for each pair of graph transformation sequences $G \overset{*}{\Longrightarrow} G_1$, $G \overset{*}{\Longrightarrow} G_2$, there is a graph G_3 together with sequences $G_1 \overset{*}{\Longrightarrow} G_3$ and $G_2 \overset{*}{\Longrightarrow} G_3$.

There are numerous applications in software engineering, concurrency, and distributed systems [7], where confluence of graph transformations plays an important role. Confluence together with termination, that is, 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 [8].

In order to show confluence it is sufficient to show local confluence and termination [13, 17], where 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. 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. The algebraic theory of graph transformations [8] provides a specific technique known from term rewriting systems [13], called *critical pair analysis*, which supports the verification of local confluence using the software system AGG [1].

Unfortunately, the theory of critical pair analysis developed for graph transformations [8] cannot be applied directly to data reduction rules. We discuss in Sections 3 and 4 what can be done so far. It is an interesting challenge for future work to extend the theory of graph transformations [8]—and the corresponding tool AGG—to handle also data reduction in a more general way.

Structure of the paper. After presenting basic concepts and definitions about kernelization and critical pair analysis in Section 2, we present our two case studies *Clique Cover* and *Partial Clique Cover* in Section 3 and 4, respectively, by showing that the corresponding data reduction rule sets yield problem kernels and are confluent. Section 5 concludes with an outlook to future work. Due to limited space, we defer some proofs and details to the full version of this paper.

2 Basic Concepts and Definitions

Kernelizations. A *parameterized problem* can be defined by a set of instances (x, k), where k is called the *parameter* [5, 9, 18]. 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 problem kernel of the parameterized problem L.

Critical pair analysis in graph transformation theory. As pointed out in the introduction, critical pair analysis is a prominent technique for showing confluence of rewriting systems [13], which has been generalized to graph transformation systems by Plump [19]. 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 leading to the same G_3. A pair is called *conflicting* if it is not parallel independent, and it has *minimal context* if each vertex and edge in G belongs to the match of r_1 or r_2 in G. For a graph transformation system with a finite number of rules based on finite graphs, there is 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 [8] 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 in Section 3 and Section 4.

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 [8] based on critical pairs is valid for directed graphs (with parallel edges and loops) and several other kinds of graphs, but not yet 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. Each rule schema corresponds to a—possibly infinite—set of rules in the sense of [8]. For these reasons, we prove confluence directly; 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 and should be replaced in future work by a uniform technique based on critical pairs with tool support by AGG.

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 [10]. Moreover, Clique Cover has a kernelization with a size bound of 2^k vertices [10, 12], and it was recently shown that under standard complexity-theoretic assumptions, this cannot be improved to a polynomial bound [4].

Kernelization for Clique Cover. For the currently only known kernelization for Clique Cover with parameter k, the following data reduction rules are used [10, 12].[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 excactly 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 ([10, 12]). *Rules 1 to 3 are correct and yield a problem kernel for* Clique Cover *with at most 2^k vertices.*

Note that for technical correctness of the kernel (as defined in Section 2), we need to add a fourth rule that checks whether after application of Rules 1 to 3 there are more than 2^k vertices left, and if so, replaces the instance with a small "no"-instance (for instance, $k+1$ disjoint edges). We omit such trivial rules 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. [10] 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. [10] does not treat isolated edges correctly; as already noted by Gyárfás [12], they require a special case.)

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, 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. [10] only state the running time of $O(|V|^4)$ for Rules 1 to 3 plus another rule).

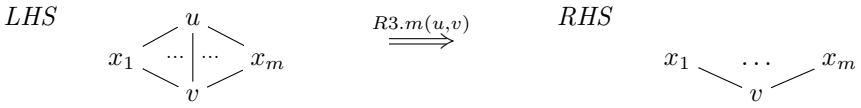
Corollary 1. *A 2^k-vertex kernel for* Clique Cover *can be found in linear time.*

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 [16, Corollary 7.4], so the kernelization can be done in linear time, too. □

Confluence via Critical Pair Analysis. As pointed out in Section 1, the standard way to show confluence of a rule set in graph transformation theory [8] 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 [8], 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.

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 $m \geq 1$:

[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. [11]).



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. 1 shows the table computed by the AGG tool [1] giving the number of critical pairs (CP) for each pair of rules and $r = 3$. Clicking on an entry in the CP table table (e. g. the highlighted field showing 12 minimal conflicts for Rule 3.2 and Rule 3.3 where rule Rule 3.2 is applied first), the 12 conflicting situations of these two rules are shown in detailed graphical views. Vertices and edges in the rules (in the bottom of Fig. 1) are numbered to define their conflicting overlapping situation. We can see one of the 12 conflicts in the overlapping graph P in the upper right part of Fig. 1, 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.

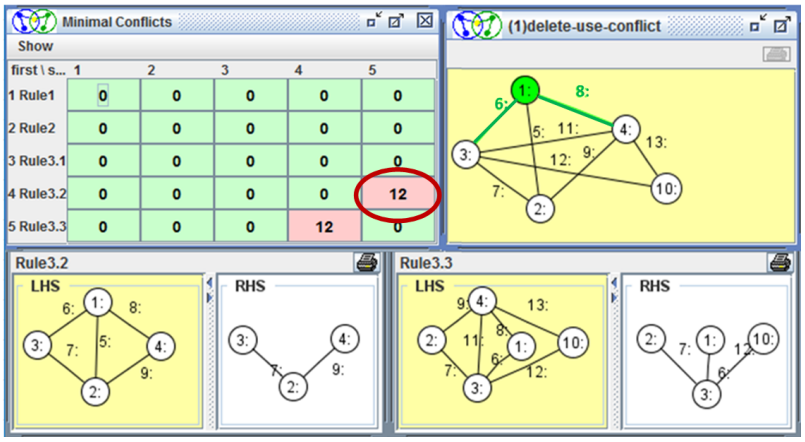
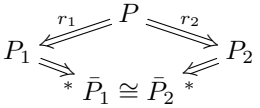


Figure 1. CP table for Clique Cover Rules 1 to 3.3, and one critical pair in detail

For each critical pair $P_1 \overset{r_1}{\Longleftarrow} P \overset{r_2}{\Longrightarrow} P_2$ of the rule set in Fig. 1, 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 to the right.



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. Due to space constraints, we can only sketch our results.

We generalize Rules 1 and 2 in a canonical way.

Rule 4 ([10, 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, 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:*[4]

$$\forall x \in V \setminus \{u, v\} : \{u, x\} \in C \iff \{v, x\} \in C \tag{1}$$

$$\{u, v\} \notin C \Rightarrow \exists x \in V \setminus \{u, v\} : \{v, x\} \notin C. \tag{2}$$

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-hard for $k = 3$, and thus cannot have a problem kernel unless P = NP. However, we can show a kernel for Partial Clique Cover with respect to the combined parameter (k, c), where $c = |C|$ is the number of covered edges.

Theorem 3. *Rules 4 to 6 yield a problem kernel for* Partial Clique Cover *with at most 2^{k+c} vertices.*

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 ise the bounds from Theorem 1.

Next, we claim that the new rules are confluent; the omitted proof uses local confluence and a somewhat involved case distinction.

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

The challenge in proving Theorem 4 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 addition to a direct proof, for graphs of bounded size Theorem 4 can be shown using critical pair analysis by AGG [1].

[4] Note that if we drop either (1) or (2), then the rule is not correct.

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 edge clique covering, 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 to an NP-hardness proof for Partial Clique Cover already for $k \geq 3$ covering cliques (see Section 4).

References

[1] AGG. *Attributed Graph Grammar Tool.* TU Berlin, 2011. http://tfs.cs.tu-berlin.de/agg.

[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.

[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.

[4] M. Cygan, S. Kratsch, M. Pilipczuk, M. Pilipczuk, and M. Wahlström. Clique cover and graph separation: New incompressibility results. Technical Report arXiv:1111.0570v1 [cs.DS], arXiv, 2011.

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

[6] 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.

[7] 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.

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

[9] J. Flum and M. Grohe. *Parameterized Complexity Theory.* Springer, 2006.

[10] 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.

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

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

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

[14] 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.

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

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

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

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

[19] 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.

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