Existential Second-Order Logic Over Graphs: Charting the Tractability Frontier

GEORG GOTTLOB

Technische Universität Wien, Wien, Austria

PHOKION G. KOLAITIS

UC Santa Cruz, Santa Cruz, California

AND

THOMAS SCHWENTICK

Philipps-Universität Marburg, Marburg, Germany

Abstract. Fagin's theorem, the first important result of descriptive complexity, asserts that a property of graphs is in NP if and only if it is definable by an existential second-order formula. In this article, we study the complexity of evaluating existential second-order formulas that belong to *prefix classses* of existential second-order logic, where a prefix class is the collection of all existential second-order formulas in prenex normal form such that the second-order and the first-order quantifiers obey a certain quantifier pattern. We completely characterize the computational complexity of prefix classes of existential second-order logic in three different contexts: (1) over directed graphs, (2) over undirected graphs with self-loops and (3) over undirected graphs without self-loops. Our main result is that in each of these three contexts a *dichotomy* holds, that is to say, each prefix class of existential second-order logic either contains sentences that can express NP-complete problems, or each of its sentences expresses a polynomial-time solvable problem. Although the

A short preliminary version of this article has been presented at the 41st Annual Symposium on Foundations of Computer Science (FOCS 2000) and was published in the respective proceedings.

G. Gottlob's research was partially supported by FWF (Austrian Science Funds) under project Z29-N04. This project also provided support for the visit of Thomas Schwentick at the Technical University of Vienna, where part of this research was carried out.

P. Kolaitis' research was partially supported by NSF Grants CCR-9610257 and IIS-9907419.

Th. Schwentick and P. Kolaitis wish to acknowledge the hospitality of the faculty and staff of the Technical University of Vienna during their visits there.

Authors' addresses: G. Gottlob, Institut für Informationssysteme, Technische Universität Wien, A-1040 Wien, Austria, e-mail: gottlob@dbai.tuwien.ac.at; P. G. Kolaitis, Computer Science Department, University of California, Santa Cruz, Santa Cruz, CA 95064, e-mail: kolaitis@cs.ucsc.edu; Th. Schwentick, Fachbereich Mathematik und Informatik, Philipps-Universität Marburg, D-35032 Marburg, Germany, e-mail: tick@informatik.uni-marburg.de.

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or direct commercial advantage and that copies show this notice on the first page or initial screen of a display along with the full citation. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, to redistribute to lists, or to use any component of this work in other works requires prior specific permission and/or a fee. Permissions may be requested from Publications Dept., ACM, Inc., 1515 Broadway, New York, NY 10036 USA, fax: +1 (212) 869-0481, or permissions@acm.org.
© 2004 ACM 0004-5411/04/0300-0312 $5.00

boundary of the dichotomy coincides for the first two cases, it changes, as one moves to undirected graphs without self-loops. The key difference is that a certain prefix class, based on the well-known *Ackermann class* of first-order logic, contains sentences that can express NP-complete problems over graphs of the first two types, but becomes tractable over undirected graphs without self-loops. Moreover, establishing the dichotomy over undirected graphs without self-loops turns out to be a technically challenging problem that requires the use of sophisticated machinery from graph theory and combinatorics, including results about graphs of bounded tree-width and Ramsey's theorem.

Categories and Subject Descriptors: F.2.3 [**Analysis of Algorithms and Problem Complexity**]: Tradeoffs between Complexity Measures; F.4.1 [**Mathematical Logic and Formal Languages**]: Mathematical Logic

General Terms: Theory, Algorithms, Languages

Additional Key Words and Phrases: Existential second-order logic, NP-complete problems, prefix classes, finite model theory, graph constraints, graph coloring

1. *Introduction and Summary of Results*

Descriptive complexity is the study of the connections between computational complexity and expressibility in logic. Over the past twenty-five years, research in this area has established that essentially all major complexity classes have natural characterizations in terms of expressibility in extensions of first-order logic on classes of finite structures (see Immerman [1998]). The prototypical result in descriptive complexity is the well-known Fagin's theorem [Fagin 1974], which asserts that a decision problem on finite graphs is in NP if and only if it is expressible in existential second-order logic ESO, that is, it is definable on all finite graphs by a sentence of the form $\exists R_1 \cdots \exists R_r \varphi$, where $R_1, \ldots, R_r$ are relational variables of various arities and φ is a first-order formula. This machine-independent characterization of NP provided the impetus for the subsequent development of descriptive complexity, found numerous applications to other areas, and motivated Papadimitriou and Yannakakis [1991] to develop a complexity theory of approximability of NP-optimization problems.

Are there tighter connections between NP-computability and expressibility in ESO that remain to be discovered? Ideally, one would like to be able to examine a given second-order sentence and determine whether it expresses a property that is NP-complete or solvable in polynomial time or of intermediate complexity (recall that Ladner [1975] showed that if P $\neq$ NP, then there are NP-problems that are neither NP-complete nor solvable in polynomial time). This goal, however, turns out to be entirely unattainable, since, using Trahtenbrot's theorem [Trahtenbrot 1963], it is easy to see that if P $\neq$ NP, then it is an undecidable problem to tell whether a given second-order sentence defines an NP-complete problem. In view of this state of affairs, one can only hope to analyze specific syntactic fragments of ESO and determine whether or not they contain sentences that express NP-complete problems. For instance, existential second-order sentences of the form $\exists P_1 \cdots \exists P_r \forall x_1 \forall x_2 \vartheta$, where each P_i is a monadic relational variable, x_1 and x_2 are first-order variables and ϑ is a quantifier-free formula, can express r-COLORABILITY. In contrast, Grädel [1991, 1992] showed that every problem expressible by a Horn existential second-order sentence is solvable in polynomial time, where an existential second-order sentence is *Horn* if it is of the form $\exists R_1 \cdots \exists R_r \forall x_1 \cdots \forall x_k \vartheta$, where $x_1, \ldots, x_k$ are first-order variables and ϑ is a quantifier-free formula in conjunctive normal form such that each conjunct contains at most one positive occurrence of each relational variable R_i, $1 \leq i \leq r$.

Among all syntactic fragments of first-order and second-order logic, *prefix classes* are undoubtedly the most well-studied ones. A *prefix class* is obtained by considering formulas in prenex normal form and imposing restrictions on the pattern of the quantifiers in the formulas (but, unlike Horn formulas, no restrictions are imposed on the quantifier-free part of the formulas). Prefix classes of first-order logic have been extensively investigated in the context of the *classical decision problem*, which is the satisfiability problem for first-order logic: given a first-order sentence ψ, is ψ satisfiable? Since this problem is unsolvable, researchers toiled for several decades and eventually succeeded in delineating the boundary between solvability and unsolvability by identifying all prefix classes of first-order logic that have a solvable satisfiability problem (see Börger et al. [1997]).

In this article, we systematically investigate and completely characterize the computational complexity of prefix classes of ESO in three different contexts: (1) over directed graphs, (2) over undirected graphs with self-loops, and (3) over undirected graphs without self-loops. Our main result is that in each of these three contexts a *dichotomy* holds, that is to say, each prefix class of ESO either contains sentences that can express NP-complete problems, or each of its sentences expresses a polynomial-time solvable problem. Although the boundary of the dichotomy coincides for the first two cases (to which we refer as *general graphs* from now on), it changes, as one moves to undirected graphs without self-loops. The key difference is that a certain prefix class, based on the well-known *Ackermann class* of first-order logic, contains sentences that can express NP-complete problems over general graphs, but becomes tractable over undirected graphs without self-loops. Moreover, establishing the dichotomy over undirected graphs without self-loops turns out to be a technically challenging problem that requires the use of sophisticated machinery from graph theory and combinatorics, including results about graphs of bounded tree-width and Ramsey's theorem.

To describe the results of this article in precise terms, we use a special notation for denoting prefix classes of ESO-sentences in prenex normal form. For example, E_1^*eaa denotes the prefix class of all prenex formulas of ESO of the form: $\exists P_1 \cdots \exists P_r \exists x \forall y \forall z \varphi$, where each P_i is a monadic relational variable, x, y, and z are first-order variables, and φ is a quantifier-free formula. It should be noted here, that we represent graphs as finite structures in the standard model-theoretic way where the universe of the structure is the set of vertices and the edges are represented by one binary relation. More generally, expressions in our special notation are built according to the following rules:

—E (respectively, E_i) denotes the existential quantification over a single predicate of arbitrary arity (arity $\leq i$).

—a (respectively, e) denotes the universal (existential) quantification of a single first-order variable.

—If η is a quantification pattern, then η^* denotes all patterns obtained by repeating η zero or more times.

An expression $\mathcal{E}$ in the special notation consists of a string of existential second-order quantification patterns (E-patterns) followed by a string of first-order quantification patterns (a or e patterns); such an expression represents the class of all

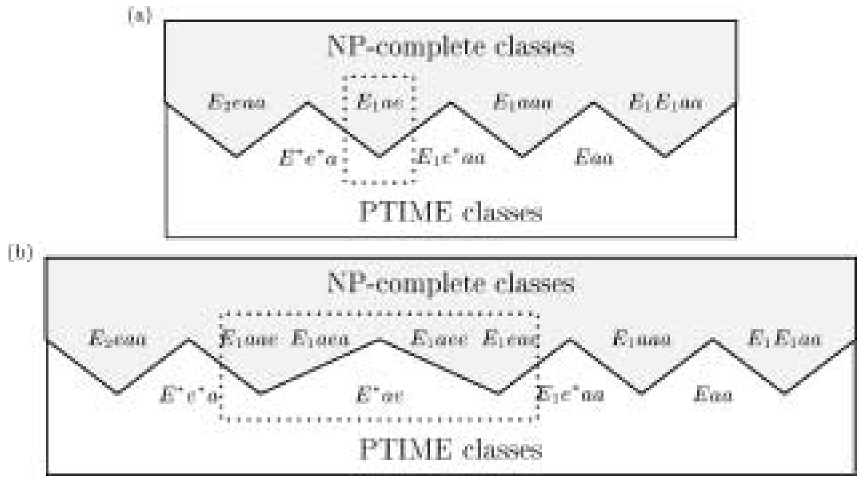

FIG. 1. (a) ESO on arbitrary structures, directed graphs and undirected graphs with self-loops; (b) ESO on undirected graphs without self-loops. The dotted boxes highlight the difference between the two cases.

ESO-formulas in prenex normal form whose quantifier prefix corresponds to a not-necessarily contiguous substring of $\mathcal{E}$.

We say that a prefix class $\mathcal{Q}$ is NP-*hard* on a class $\mathcal{K}$ of relational structures if at least one NP-hard property on $\mathcal{K}$ is expressible by a formula in $\mathcal{Q}$. A prefix class $\mathcal{Q}$ is *polynomial-time* (PTIME) on $\mathcal{K}$ if for each formula $\Phi \in \mathcal{Q}$, the model-checking problem for Φ is solvable in polynomial time, which means that the set of all structures $A \in \mathcal{K}$ such that $A \models \Phi$ is in P. A prefix class is called *first-order* (FO) if every formula in it is equivalent to a first-order formula. A prefix class is *syntactically first order* if its definition involves only first-order quantifiers.

Our first result completely characterizes the computational complexity of prefix classes of ESO on general graphs. In fact, the same characterization holds true on the collection of all finite structures over any relational vocabulary that contains a relation symbol of arity ≥ 2. This characterization is obtained by showing that (assuming P $\neq$ NP) there are four *minimal* NP-hard prefix classes and three *maximal* PTIME prefix classes, and that these seven classes combine to give complete information about all other prefix classes. This means that every other prefix either contains one of the minimal NP-hard prefix classes as a substring (and, hence, it is NP-hard) or is a substring of a maximal PTIME prefix class (and, hence, it is PTIME). Figure 1(a) depicts the characterization of the NP-hard and PTIME prefix classes of ESO fragments on general graphs. As seen in that figure, the four minimal NP-hard classes are E_1ae, E_1aaa, E_2eaa, and E_1E_1aa, whereas the three maximal PTIME classes are E^*e^*a, E_1e^*aa, and Eaa. As an example, it is easy to see that a certain encoding of SAT on general graphs can be expressed using a sentence in E_1ae. Note that the first-order prefix class *ae* is known as the *Ackermann* class and has played a key role in the study of the classical decision problem for fragments of first-order logic (see Börger et al. [1997]). As regards the maximal PTIME classes, the prefix class E^*e^*a is actually FO, while the model checking problem for fixed sentences in the prefix classes E_1e^*aa and Eaa can be reduced to 2SAT and, thus, it is in PTIME (in fact, it is in NL).

Our second result completely characterizes the computational complexity of prefix classes of ESO on undirected graphs without self-loops. As mentioned earlier, we establish that a dichotomy still holds, but the boundary of the dichotomy changes. The key difference is that the prefix class E^*ae turns out to be a PTIME class on undirected graphs without self-loops, whereas its subclass E_1ae is NP-hard on general graphs. It should be pointed out that certain interesting properties of graphs can be expressed using E^*ae-formulas. Specifically, for each positive integer m, there is a E^*ae-formula expressing that a connected graph contains a cycle whose length is divisible by m. These problems were shown to be solvable in polynomial time by Thomassen [1988]. E^*ae constitutes a maximal PTIME class, because we show that all four extensions of E_1ae by a single first-order quantifier (universal or existential) are NP-hard on undirected graphs without self-loops. The other minimal NP-hard prefixes on general graphs remain NP-hard on undirected graphs without self-loops as well. Consequently, on undirected graphs without self-loops there are seven minimal NP-hard prefix classes and four maximal PTIME prefix classes that determine the computational complexity of all other prefix classes of ESO, as seen in Figure 1(b).

From the technical point of view, the most difficult result of the article is showing that the prefix class E^*ae is PTIME on undirected graphs without self-loops. First, using syntactic methods, we show that this fragment has the same expressive power as its monadic subfragment, that is to say, each E^*ae-formula is equivalent to some E_1^*ae-formula. After this, we analyze the prefix class E_1^*ae on undirected graphs without self-loops and show that the model-checking problem for each E_1^*ae-formula is equivalent to a natural coloring problem, which we call the *saturation problem*. More specifically, the saturation problem asks whether there is a mapping with special properties from a given undirected graph without self-loops to a fixed, directed *pattern graph* P that is extracted from the E_1^*ae-formula under consideration. Depending on the labelings of cycles in P, we distinguish two cases of this coloring problem: the saturation problem for *pure pattern graphs* and the saturation problem for *mixed pattern graphs*. We then design polynomial-time algorithms that solve the saturation problem in each of these two cases. In simplified terms and focussed on the case of connected graphs, the polynomial-time algorithm for the saturation problem for pure pattern graphs has three main ingredients. First, adapting results by Thomassen [1988] and introducing a new graph coloring method, we show that if a E_1^*ae-formula gives rise to a pure pattern graph, then we can find a fixed integer k such that the formula is satisfied by every undirected graph without self-loops having tree-width bigger than k. Second, we use Courcelle's theorem [Courcelle 1990] to the effect that the model-checking problem for formulas of monadic second-order logic on graphs of bounded tree-width is solvable in polynomial-time. Third, we use Bodlaender's result [Bodlaender 1996] to the effect that, for each fixed k, there is a polynomial-time algorithm to check if a given graph has tree width at most k. The polynomial-time algorithm for the saturation problem for mixed pattern graphs has similar architecture overall, but requires the development of substantial additional technical machinery, including a generalization of the concept of graphs of bounded tree-width. The results of the article can be summarized in the following theorem.

THEOREM 1.1. *The classifications in Figure* 1 *determine the complexity of all ESO prefix classes on graphs.*

The results presented here should be contrasted with those in an earlier paper that gives a complete characterization of the ESO prefix classes over *strings* [Eiter et al. 2000]. While graphs are structures with an arbitrary binary relation, strings are structures consisting of unary relations (one for each letter of the alphabet) and a binary *successor* relation. The main result of Eiter et al. [2000] is a dichotomy theorem for ESO prefix classes over strings; this dichotomy, however, differs from the present one in both its scope and the methods used to establish it. First of all, the dichotomy in Eiter et al. [2000] is between NP-hard classes and classes of *regular* languages, that is, languages recognized by finite automata. Second, due to the combinatorially simpler structure of strings, some ESO prefix classes that are NP-hard over graphs turn out to be regular over strings. Specifically, in Eiter et al. [2000] it was shown that there are precisely two maximal polynomial ESO prefix classes over strings: the class $E^*e^*ae^*$ and the class E^*e^*aa. It is perhaps interesting to note that e^*ae^* and e^*a^* are the only prefix classes of first-order logic with equality that have a solvable satisfiability problem, while for first-order logic without equality, the prefix e^*aae^* has a solvable satisfiability problem as well (see Börger et al. [1997]). In contrast, here we show that the classes $E^*e^*ae^*$ and E^*e^*aa are NP-complete over graphs, but certain proper prefix subclasses of them are polynomial-time. Moreover, the classification in Figure 1 holds for both first-order logic with equality and first-order logic without equality. We also note that, in Eiter et al. [2000], the classes $E^*e^*ae^*$ and E^*e^*aa were shown to be regular over strings by first proving that they are actually contained in monadic second order logic (MSO) and then using Büchi's theorem to the effect that MSO over strings expresses precisely the regular languages [Büchi 1960]. Since MSO over *graphs* does express NP-complete properties (e.g., 3-COLORABILITY), we cannot use this approach in the present article for establishing polynomial-time upper bounds; instead, we have to use completely different methods from graph theory.

The work presented here suggests several directions of future research. First, it would be interesting to investigate the complexity of prefix classes of full second-order logic over directed graphs and over undirected graphs (with and without self-loops). It should be noted that such a study for prefix classes of full second-order logic over strings has already been carried out in Eiter et al. [2002], and the prefix classes describing regular languages have been determined. Second, it would be interesting to investigate the complexity of ESO prefix classes on restricted collections of graphs of algorithmic significance. Here, we focussed on the complexity of ESO prefix classes on the collection of all directed graphs and the collections of undirected graphs with and without self-loops, as these are the most extensively studied classes of binary relational structures. It is well known, however, that there are important decision problems that are NP-complete when the inputs are arbitrary graphs (directed or undirected), but are solvable in polynomial time on restricted collections of graphs, such as planar graphs or regular graphs. Consequently, the boundary between tractability and intractability for ESO prefix classes over such restricted collections of graphs may be different from the corresponding boundary for ESO prefix classes over arbitrary graphs discovered in this article, and remains to be explored and delineated. Finally, although the quantification pattern is arguably one of the most natural ways to obtain syntactic fragments of logical formalisms, it is not the only one. Different fragments can be obtained by considering syntactic properties of the quantifier-free part of formulas (such as

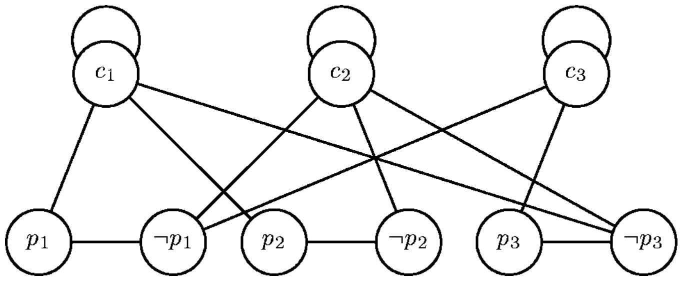

FIG. 2. Encoding of clauses $c_1 = \{p_1, p_2, \neg p_3\}$, $c_2 = \{\neg p_1, \neg p_2, \neg p_3\}$, and $c_3 = \{\neg p_1, p_3\}$.

being Horn) or by combining quantification patterns with syntactic properties of the quantifier-free part. Determining the complexity of such fragments of existential second-order logic on arbitrary graphs or on restricted collections of graphs remains an open problem.

2. *NP-Hardness Results*

THEOREM 2.1. *E_1ae is* NP*-hard on graphs. This holds even for self-loop free directed graphs and for undirected graphs with some self-loops.*

PROOF. We give a reduction from SATISFIABILITY OF PROPOSITIONAL FORMULAS (SAT). Let $S = \{c_1, \ldots, c_m\}$ be a set of clauses over propositional variables $p_1, \ldots, p_n$. Construct an undirected graph $G_S = (V_S, E_S)$ such that V_S contains a vertex c_i for $1 \leq i \leq m$, that is, each clause is identified with a vertex; moreover, V_S contains a couple of vertices p_i and $\neg p_i$ $1 \leq i \leq n$, that represent the literals p_i and $\neg p_i$, respectively. The edge relation E_S contains a self-loop $\{c_i, c_i\}$ for $1 \leq i \leq m$, i.e., each vertex representing a clause has a self-loop. Moreover, for $1 \leq i \leq m$ and $1 \leq j \leq n$, the edge relation E_S contains an edge $\{c_i, p_j\}$ (respectively, $\{c_i, \neg p_j\}$), if clause c_i contains literal p_j (respectively, $\neg p_j$). Finally, for $1 \leq i \leq n$, there is an edge $\{p_i, \neg p_i\}$.

Figure 2 depicts the encoding of a clause set $S = \{c_1, c_2, c_3\}$ with $c_1 = \{p_1, p_2, \neg p_3\}$, $c_2 = \{\neg p_1, \neg p_2, \neg p_3\}$, and $c_3 = \{\neg p_1, p_3\}$.

Let Φ be the following E_1ae formula over the structure $G_S = (V_S, E_S)$:

$$\begin{aligned}\Phi : \exists T\, \forall x \exists y\, [(E_S(x, x) \wedge E_S(x, y) \wedge \neg E_S(y, y) \wedge T(y)) \vee (\neg E_S(x, x) \\ \wedge E_S(x, y) \wedge \neg E_S(y, y) \wedge T(x) \not\leftrightarrow T(y))].\end{aligned}$$

Intuitively, the monadic predicate T expresses a truth value assignment to the literals of S. The first disjunct in the matrix of Φ says that every clause must contain at least one true literal. The second disjunct says that opposite literals have opposite truth values by T (and hence T is a correct truth value assignment). The clause set S is thus satisfiable iff Φ evaluates to true over G_S. It follows that the prefix class E_1ae is NP-hard over undirected graphs with (some) self-loops.

Note that the self-loops in G_S are used to distinguish vertices that represent clauses from vertices representing literals, i.e., they are used for *typing* vertices. To see that E_1ae is also NP-hard on directed self-loop free graphs, it suffices to exploit arc directions for this typing task. In this case, the clause set S is encoded by a directed graph $G'_S = (V_S, E'_S)$ where the arcs of E'_S are as follows. For $1 \leq i \leq m$

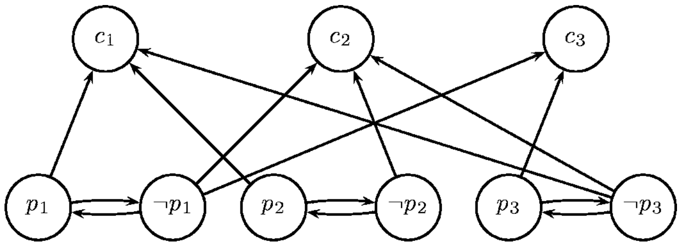

FIG. 3. Encoding of same set S as self-loop free directed graph.

and $1 \leq j \leq n$, if p_j occurs in clause c_i, then E'_S contains (p_j, c_i) as an arc, while if $\neg p_j$ occurs in clause c_i, then E'_S contains $(\neg p_j, c_i)$ as an arc. Moreover, E'_S contains the arcs $(p_j, \neg p_j)$, $(\neg p_j, p_j)$, for $1 \leq j \leq n$. See Figure 3.

Instead of the formula Φ, we now use the following formula Φ' on G'_S:

$$\Phi' : \exists T\, \forall x \exists y\, [(E'_S(y, x) \wedge \neg E'_S(x, y) \wedge T(y)) \vee (E'_S(x, y) \wedge E'_S(y, x) \wedge T(x) \not\leftrightarrow T(y))]. \qquad \square$$

THEOREM 2.2. *E_1aaa is* NP-*hard on self-loop free undirected graphs.*

PROOF. The following problem POSITIVE ONE-IN-THREE 3SAT (which we abbreviate to POIT3SAT) is well-known to be NP-complete (see Garey and Johnson [1979]). Let U be a set of propositional variables and let S be a set of positive 3-clauses over U, that is, each clause consists of a disjunction of three distinct propositional variables from U. Is there a truth value assignment for U such that in each clause of S exactly one propositional variable becomes true?

We reduce each instance S of POIT3SAT to a graph $G_S = (V_G, E_G)$ as follows. Each *occurrence* of a propositional variable p in some clause c of S gives rise to a vertex p_c in the graph. Moreover, each variable p in U gives rise to a vertex p. There are no other vertices in V_G. Vertices corresponding to occurrences of variables belonging to the same clause are linked by an edge (we thus have a triangle for each clause). Moreover, for each vertex p_c in V_S, there is an edge between p_c and p.

Let Φ be a $Eaaa$ formula stating that there is a P (intended to simulate the truth value assignment to the propositional variables) such that:

(1) for every three nodes x, y, z forming a triangle, exactly one is in P;
(2) for every three nodes x, y, z such that there are edges $\{x, y\}$, $\{x, z\}$, but no edge $\{y, z\}$ (i.e., x, y, z form an angle with x as pivot), at least one of the following holds:
    —both x and y are in P;
    —neither x, nor y are in P;
    —both x and z are in P;
    —neither x, nor z are in P.
    In other words, either x and y get the same P-type or x and z get the same P-type.

Now, we need to verify that this formula indeed expresses that S is in POIT3SAT.

First, it is clear that if there is a 1-IN-3 satisfying assignment, then the formula is satisfied by assigning a node to P if and only if the associated variable takes value true.

For the other direction, assuming that a graph satisfies the formula, it suffices to show that all vertices corresponding to the same variable have the same P-type (i.e., either all of them are in P or none is in P).

Suppose we have two clauses $\{x, y, z\}$ and $\{x, w, v\}$. Let $\{x', y, z\}$ and $\{x'', w, v\}$ be the associated triangles, and let x be the node that has edges to the nodes x' and x''. If x' is in P then y is not in P (by condition (1) above). Hence, by applying condition (2) to the angle y, x', x, we infer that x must be in P. We now claim that x'' must also be in P. Otherwise, by condition (1) on the triangle $\{x'', w, v\}$, we have that either w or v is in P. Assume that w is in P. But now we have a violation of condition (2) for the angle w, x'', x as x'' has different P-type than both w and x. So, we now have that x, x', x'' are all in P. We can continue this way and repeat the argument for the other occurrences of x in clauses.

It follows that, for each propositional variable in S, the set P contains either all or none of the nodes of G_S that represent the occurrences of that propositional variable. Hence, P correctly describes a truth value assignment τ_P to the propositional variables of S. Clearly, τ_P satisfies S. Moreover, each clause of S contains exactly one literal made true by τ_P. Consequently, S is in POIT3SAT. □

THEOREM 2.3. *E_1E_1aa is* NP-*hard on self-loop free undirected graphs.*

PROOF. GRAPH 3COLORABILITY (3COL) of an undirected self-loop free graph $G = (V, E)$ is expressed by the following E_1E_1aa formula Φ

$$\exists Red\ \exists Green\ \forall x \forall y\ [(\neg Red(x) \vee \neg Green(x)) \wedge (E(x, y) \rightarrow diffcol(x, y))],$$

where *Red* and *Green* are monadic predicates expressing the coloring of vertices, where the third color, say blue is represented by the complement of *Red* ∪ *Green*, and where *diffcol*(x, y) is a quantifier-free formula stating that x and y are of different color. □

THEOREM 2.4. *E_2eaa is* NP-*hard on self-loop free undirected graphs.*

PROOF. GRAPH 3COLORABILITY of an undirected self-loop free graph $G = (V, E)$ can be expressed by an E_2eaa formula of the form $\exists R\ \exists z\ \forall x\ \forall y\ \varphi$ in a similar way as in the proof of Theorem 2.3, except that the colors are now expressed differently. Specifically, it suffices to replace *Red*(x) by $R(z, x)$ and *Green*(x) by $R(x, z)$. Note that z must then be colored blue, but this can be assumed without loss of generality. □

The following NP-hardness results are of relevance to the self-loop free undirected case only (Figure 1(b)). When self-loops or directed arcs are permitted, all these results are implied by Theorem 2.1.

THEOREM 2.5. *E_1eae is* NP-*complete over self-loop free undirected graphs.*

PROOF. We show that E_1eae expresses NP-complete problems on undirected graphs with some self-loops by modifying the encoding of SAT used in the proof of Theorem 2.1 (see Figure 2). Specifically, we delete all self-loops and add a special node a with edges to each node representing a literal (variable or negated variable). This way, each literal is part of some triangle, but no clause is part of any triangle.

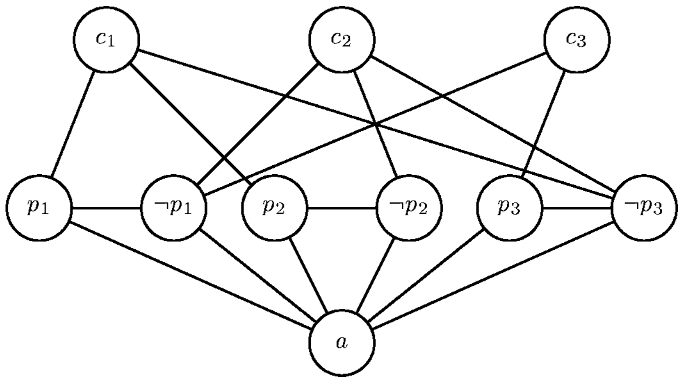

FIG. 4. New encoding of clauses $c_1 = \{p_1, p_2, \neg p_3\}$, $c_2 = \{\neg p_1, \neg p_2, \neg p_3\}$, and $c_3 = \{\neg p_1, p_3\}$.

Figure 4 depicts this for the clause set $S = \{c_1, c_2, c_3\}$ with $c_1 = \{p_1, p_2, \neg p_3\}$, $c_2 = \{\neg p_1, \neg p_2, \neg p_3\}$, and $c_3 = \{\neg p_1, p_3\}$.

The sentence that defines SAT over these graphs is:

$$\begin{aligned}\Phi : \exists P\, \exists w \forall x\, \exists y\, [(\neg E(w, x) &\to (E(x, y) \wedge E(w, y) \wedge P(y))) \wedge (E(w, x)\\ &\to (E(w, y) \wedge E(x, y) \wedge (P(x) \not\leftrightarrow P(y))))].\end{aligned}$$

If the formula is satisfiable, then the graph satisfies the above sentence. To see this, let τ be a satisfying truth value assignment for S and let P be interpreted by the set containing all vertices that correspond to literals made true by τ, and interpret w by a. The graph then clearly satisfies Φ.

Conversely, assume that the graph satisfies Φ. Then the special node a is the only node that can witness the existential quantifier $\exists w$. The reason is that the second conjunct of Φ rules out the possibility that w represents a clause or a literal. For example, suppose that w represents a clause c_i. Then choose a literal x in c_i. By the second conjunct of Φ, there is a y so that $\{w, x, y\}$ form a triangle, which is impossible. Similarly, if w represents a literal, then choose a clause x in which it occurs, so by the second conjunct of Φ, there is a y so that $\{w, x, y\}$ is a triangle, which is again impossible.

Given that w must be interpreted by a, each pair x, y of vertices representing opposite literals $p_j, \neg p_j$ must be P-inequivalent, that is, $P(x) \not\leftrightarrow P(y)$. Thus, any witness set P for Φ correctly represents a truth value assignment to the variables occurring in S. By the first conjunct of Φ, each vertex x representing a clause must be linked to some literal y made true by this assignment. Thus, P represents a satisfying truth value assignment for S and hence S is satisfiable. $\square$

THEOREM 2.6. *E_1aee is* NP-*complete over self-loop free undirected graphs.*

PROOF. We use the NP complete problem NOT-ALL-EQUAL-3SAT, the version of 3SAT in which a truth value assignment is sought such that each clause has at least one true and one false literal. Without loss of generality, we assume that for each instance I of NOT-ALL-EQUAL-3SAT, every literal occurs in at least one clause (if some literal α does not appear in any clause, then we can redress this by adding two clauses $\{\alpha, r, s\}$ and $\{\alpha, \neg r, \neg s\}$ to I; clearly, the modified instance is in NOT-ALL-EQUAL-3SAT iff I is). Further, we require that no clause contains a literal as well as its negation.

An instance I of this problem is encoded as follows by a graph $G(I) = (V_I, E_I)$. For each propositional variable p, create three vertices p, p', and p^*, and make a triangle out of them, i.e., add edges (p, p'), (p, p^*), and (p^*, p') to E_I. Intuitively, vertex p stands for the variable p, vertex p' for its negation $\neg p$, and p^* is a dummy vertex whose role will become clear soon. For each clause c_i, create a vertex c_i and link it by an edge to the vertex representing each literal occurring in it. This completes the encoding.

Let Φ be the following E_1aee-formula:

$$\exists P\, \forall x\, \exists y\, \exists z\, [(triangle(x, y, z) \wedge (P(y) \not\leftrightarrow P(z)) \wedge P(x)) \vee (E(x, y) \wedge E(x, z) \wedge \neg E(y, z) \wedge P(y) \wedge \neg P(z))],$$

where *triangle(x, y, z)* states that x, y, z form a triangle in $G(I)$.

We claim that I is a yes instance of NOT-ALL-EQUAL-3SAT iff $G(I) \models \Phi$.

First, assume I is a yes instance of NOT-ALL-EQUAL-3SAT. Let τ be a truth value assignment witnessing this. Define P as follows: $P(p)$ iff $\tau(p) = true$; $P(p')$ iff $\tau(p) = false$; and for all propositional variables p, $P(p^*)$, while for all clauses c, $\neg P(c)$. It is clear that such a P witnesses that $G(I)$ satisfies Φ. Conversely, if $G(I) \models \Phi$, then the first disjunct of the formula (with $x = p^*$) enforces $P(p*)$, for each p; moreover, it enforces that for all vertices p and p' it must hold that $P(p) \not\leftrightarrow P(p')$. By the second disjunct, every clause c must have at least one true and one false literal. Thus, I is a yes instance of NOT-ALL-EQUAL-3SAT. □

THEOREM 2.7. *E_1aea is* NP-*complete over self-loop free undirected graphs.*

PROOF. Consider POSITIVE 1-IN-3-SAT (POIT3SAT) again and use the same encoding of an instance S into a graph $G_S = (V_S, E_S)$ as in the proof of Theorem 2.2. This means that there is a node for each occurrence of each variable; the nodes that correspond to a clause are linked to form a triangle; for each variable x, there is a special node x^* with edges to each node corresponding to an occurrence of x.

Let Φ be a E_1aea-formula asserting the following:
$\exists P\, \forall x\, \exists y\, \forall z$ such that

(1) There is an edge from x to y and
(2) x and y have different P-type (i.e., $P(x) \not\leftrightarrow P(y)$); and
(3) if x, y, and z form a triangle, then exactly one of x, y, and z is in P;
(4) if z is connected to x, but is not connected to y, then y and z have the same P-type (i.e., $P(y) \leftrightarrow P(z)$).

We claim that $G_S \models \Phi$ iff S is in POIT3SAT.

If direction. If S has a 1-IN-3 satisfying assignment, then put in P the occurrences of variables that are *true* under this assignment; also, for each variable x, put the special node x^* in P if and only if x is *false*. To see that this P satisfies the first-order part of the formula, observe that

—for each occurrence x of a variable, take the occurrence of a variable of different P-type occurring in the same clause as the witness for $\exists y$.
—for each special node x^*, take one of the occurrences of the variable x as the witness for $\exists y$.

Only if direction. Assume that we have a graph that satisfies the above formula. We want to show that the original Boolean formula that generated the graph has a 1-IN-3 satisfying assignment.

First, consider a special node x^*. Since it has edges only to occurrences of the variable x, the node that witnesses y must be one of these occurrences. From condition (4), it follows that all occurrences of the variable x have the same P-type.

So, it suffices to show that for every triangle it is the case that exactly one of its nodes is in P.

Let $\{A, B, C\}$ be a triangle. We consider two cases. In the first case we assume that at least one node, say A, is in P. Let w be the variable corresponding to A. We consider the node that witnesses the existential quantifier $\exists y$ in the formula above if x is instantiated by A. If this witness y happens to be another node in the same triangle, then by condition (3) exactly one of the nodes of that triangle is in P. If not, then w^* must be the witness for y (since, by condition (1), there is an edge between x and y). By condition (2), as A is in P, w^* is not in P. Hence, by condition (4), as B and C are connected to A but not to w^*, we have that B and C are not in P (they must have the same P-type as w^*). Consequently, exactly one of A, B, C is in P, as required.

In the remaining case, we assume that neither of A, B, C is in P. As the first case shows, this implies that we get w^* as witness for y, if we instantiate x by A. By condition (2), it follows that w^* is in P, hence again by condition (4), the remaining two nodes B and C of the triangle are also in P, a contradiction. This completes the proof. □

THEOREM 2.8. *E_1aae is* NP*-complete over self-loop free undirected graphs.*

PROOF. Once again, we use POIT3SAT, but the encoding is different. For each instance S of POIT3SAT, we construct the graph $G_S = (V_S, E_S)$ as follows. Create a node for each occurrence of each variable; the nodes that correspond to a clause are linked to form a triangle; for each variable, we pick one of its occurrences (say, the one that occurs in the clause c_i with the smallest index i) and connect this occurrence to all other occurrences of the same variable. Note that, without loss of generality, we may assume that each variable has at least two occurrences (else, we eliminate it).

Let Φ be a E_1aae-formula asserting the following:

$\exists P\, \forall x\, \forall y\, \exists z$ such that if x and y are connected via an edge, then one of the following two properties holds:

(1) x, y, z form a triangle and exactly one of x, y, z is in P; or

(2) x and y have the same P-type (i.e., $P(x) \leftrightarrow P(y)$); and z is connected to x, but not to y; and z has different P-type than x (and than y) (i.e., $P(x) \not\leftrightarrow P(y)$).

We claim that $G_S \models \Phi$ iff S is in POIT3SAT.

If. If the Boolean formula has a 1-IN-3 satisfying assignment, then put into P the occurrences of the variables that are *true* under that assignment. One has to verify then that for every pair (x, y) connected via an edge we can find a suitable witness z. If (x, y) are part of a triangle, then z is the third node of that triangle. If (x, y) is an edge that results from two occurrences of the same variable, then as z take a node in the same triangle as x that has different P-type than x (which is possible by the 1-IN-3 satisfiability condition).

Only if. For the more interesting direction, assume that the graph satisfies the above formula. First, consider edges that result from different occurrences of the same variable x. Since they do not extend to a triangle, they must satisfy condition 2. In particular, the endpoints of each such edge must have the same P-type. But since all such edges have a node in common, we conclude that all occurrences of the same variable have the same P-type (all are in P or none is in P). So, it remains to show that for each triangle exactly one of its three nodes is in P. Towards a contradiction, assume that we have a triangle with nodes A, B, C such that it is not the case that exactly one of A, B, C is in P. We distinguish two cases (in what follows, we will write $A', A'', \ldots,$ for other occurrences of the same variable as the one for the node A):

Case 1. at least two of the nodes A, B, and C are in P. Let us assume that A and B are in P.

As seen in the preceding paragraph, the nodes $A', A'', \ldots$ must also be in P. Consider now the edge (A, B) and the witness for $\exists z$. This witness can not be the node C, so it must be one of $A', A'', \ldots$ and also it must have different P-type than A, which is impossible. So, this case cannot occur.

Case 2. none of the nodes A, B, C is in P.

As before, the nodes $A', A'', \ldots$ are not in P. Again, consider the edge (A, B) and the witness for $\exists z$. It cannot be C, so it must be one of $A', A'', \ldots$ and must have different P-type than A, which is impossible. So, this case cannot occur as well. This completes the proof. □

3. *Tractability Results over General Graphs*

THEOREM 3.1. *E^*ea is in* PTIME *and actually in FO over general graphs.*

PROOF. Lemma 12.2 of Eiter et al. [2000] shows that E^*e^*a is FO over *strings* (i.e., the inputs are monadic structures with a successor relation). An inspection of that proof reveals that it actually holds for arbitrary relational structures as inputs. □

THEOREM 3.2. *E_1e^*aa is in* NL *over general graphs and, hence, it is in* PTIME.

PROOF. Let ψ be a fixed E_1e^*aa-sentence. Without loss of generality, we may assume that ψ is of the form $\exists P \exists x_1 \cdots \exists x_m \forall y \forall z \bigwedge_{i=1}^{k} \vartheta_i(x_1, \ldots, x_m, y, z)$, where P is a unary predicate and each $\vartheta_i(x_1, \ldots, x_m, y, z)$, $1 \leq i \leq k$, is a disjunction of atomic or negated atomic formulas. In what follows, we describe a logspace algorithm with oracle in NL (short, an L^{NL} algorithm) that solves the model-checking problem for ψ over general graphs.

Denote the input graph by $G = (V, E)$. Let *accept* be a global variable which is initially set to *false*. The algorithm cycles over all m-tuples $(a_1, \ldots, a_m)$ of vertices from V and over m-tuples of truth values to the atomic statements $P(a_1), \ldots, P(a_m)$ until either *accept* $=$ *true* or all pairs of m-tuples are considered. The algorithm returns the value of the variable *accept* before stopping.

For each m-tuple $(a_1, \ldots, a_m)$ of vertices from V and for each m-tuple of truth-values to the atomic statements $P(a_1), \ldots, P(a_m)$, the algorithm generates in logarithmic-space a 2CNF-formula $\varphi_{(a_1,\ldots,a_m)}$ whose clauses are formed as follows: for each pair (b, c) of nodes from V and each formula ϑ_i, $1 \leq i \leq k$, let $\vartheta_i^{(b,c)}$ be the propositional clause obtained from $\vartheta_i(x_1, \ldots, x_m, y, z)$ by making the substitutions $x_1/a_1, \ldots, x_m/a_m, y/b, z/c$, evaluating all atomic and negated formulas involving the edge relation E and the equality symbol $=$, evaluating all atomic and negated atomic formulas of the form $P(a_i)$, $1 \leq i \leq m$, and replacing all occurrences of the atomic formulas $P(b)$ and $P(c)$ by propositional variables P_b and P_c respectively. Note that this process gives rise to a propositional clause with at most two literals, since either it is logically equivalent to one of the logical constants *True* and *False*, or it is one of the clauses $P(b)$, $\neg P(b)$, $P(c)$, $\neg P(c)$, $(P(b) \vee P(c))$, $(\neg P(b) \vee P(c))$, $(\neg P(c) \vee P(b))$, $(\neg P(b) \vee \neg P(c))$. Let $\varphi_{(a_1,\ldots,a_m)}$ be the 2NCF-formula

$$\bigwedge_{i=1}^{k} \bigwedge_{(b,c)\in V^2} \vartheta_i^{(b,c)}.$$

The algorithm then queries a 2SAT oracle asking whether $\varphi_{(a_1,\ldots,a_m)}$ is satisfiable and if so makes the assignment *accept* := *true*.

It is quite clear that $G \models \psi$ if and only if there is an m-tuple $(a_1, \ldots, a_m)$ of nodes and an m-tuple of truth values to the atomic formulas $P(a_1), \ldots, P(a_m)$ such that the 2CNF-formula $\varphi_{(a_1,\ldots,a_m)}$ is satisfiable. Thus, the algorithm is correct. Note that all actions of the algorithm, except its oracle queries are computable in logarithmic space. Note that 2SAT is in NL (see Papadimitriou [1994, page 185]). The algorithm is thus effectively an L^{NL} procedure. By well-known results [Immerman 1988; Szelepcsènyi 1988], L^{NL} collapses to NL. It follows that the model-checking problem for ψ over general graphs is in NL. $\square$

Remark 3.1. It should be pointed out that E_1e^*aa-formulas can express natural NL-complete problems over graphs. For example, DISCONNECTIVITY on directed graphs is expressible by the E_1eeaa-formula

$$\exists P \exists x_1 \exists x_2 \forall y \forall z (P(x_1) \wedge \neg P(x_2) \wedge ((P(y) \wedge \neg P(z)) \rightarrow (\neg E(y, z) \wedge \neg E(z, y)))).$$

Moreover, it is not hard to see that even E_1eaa-formulas can express NL-complete problems. Indeed, UNREACHABILITY on directed graphs has a logarithmic-space reduction to instances of 2SAT in which every clause is a unit clause p, or a negated unit clause $\neg p$, or an implication $(p \rightarrow q)$ (see Papadimitriou [1994, Theorem 16.3, page 398]). Consequently, the restriction of 2SAT to such 2CNF-formulas is NL-complete. Note that each such 2CNF-formula χ can be encoded by a directed graph G_χ with a single self-loop as follows: the nodes of G_χ are the variables of χ and a distinguished node a that is not a variable of χ; for every unit clause p of χ, there is an edge $E(a, p)$; for every negated unit clause $\neg p$ of χ, there is an edge $E(p, a)$; for every clause $(p \rightarrow q)$ of χ, there is an edge (p, q); finally, there is a self-loop $E(a, a)$ that enables us to distinguish a from all other nodes of $G\chi$. It is now clear that χ is satisfiable if and only if $G\chi$ satisfies the E_1eaa-sentence

$$\begin{aligned} \exists P \exists x \forall y \forall z (E(x, x) \ \wedge \ & (E(x, y) \rightarrow P(y)) \wedge (E(y, x) \rightarrow \neg P(y)) \\ \wedge \ & ((E(y, z) \wedge (x \neq y) \wedge (x \neq z)) \rightarrow (\neg P(y) \vee P(z)))). \end{aligned}$$

Observe that if we consider structures over a vocabulary consisting of a binary predicate E and a unary predicate T, then the satisfiability of χ is expressible by the E_1aa-sentence

$$\exists P\forall y\forall z((T(y)\ \wedge\ E(y,z))\rightarrow P(z))\wedge((T(y)\wedge E(z,y))\rightarrow\neg P(z))$$
$$\wedge\ ((\neg T(y)\wedge\neg T(z)\wedge E(y,z))\rightarrow(\neg P(y)\vee P(z))).$$

Here, T is used to mark the node a, that is, only a is in T. We also note that 2-COLORABILITY on undirected self-loop free graphs is expressible by the E_1aa-sentence

$$\exists P\forall y\forall z(E(y,z)\rightarrow(P(y)\leftrightarrow\neg P(z)).$$

This problem, however, is not known to be NL-complete. In fact, there is strong evidence that it is not, since Jones et al. [1976] showed that 2-COLORABILITY is logarithmic-space equivalent to UNDIRECTED REACHABILITY, a problem which is generally believed to be of complexity strictly between L and NL.

LEMMA 3.3. *Every Eaa-sentence over graphs is equivalent to some E_1aa-sentence.*

PROOF. Let m be a positive integer greater than 1 and let $\exists R\forall x\forall y\varphi(x,y)$ be a E_maa-sentence, where R is an m-ary predicate symbol and $\varphi(x,y)$ is a quantifier-free formula. Without loss of generality, we may assume that for every graph $G=(V,E)$ and every m-ary relation R^G on G it is the case that

$$(V,E,R^G)\models\forall x\forall y(\varphi(x,y)\rightarrow\varphi(y,x)),$$

where $\varphi(y,x)$ is the formula obtained from $\varphi(x,y)$ by switching the occurrences of x and y. This is because the formula $\forall x\forall y\varphi(x,y)$ is clearly equivalent to the formula $\forall x\forall y(\varphi(x,y)\wedge\varphi(y,x))$, whose quantifier-free part possesses this symmetric property. Also, without loss of generality, we may assume that $\varphi(x,y)$ is the disjunction of *complete consistent types*, that is to say, $\varphi(x,y)$ is of the form $\bigvee_{i=1}^k\vartheta_i(x,y)$, where each $\vartheta_i(x,y)$ is a conjunction of atomic or negated atomic formulas such that for every atomic formula either the atomic formula itself or its negation (but not both) occur as a conjunct of $\vartheta_i(x,y)$. In particular, each $\vartheta_i(x,y)$ has exactly one of the two formulas $x=y$ and $x\neq y$ as one of its conjuncts. Let P be a unary predicate and let

$$\exists P\forall x\forall y\left(\bigvee_{i=1}^k\vartheta_i'(x,y)\right)$$

be the E_1aa-sentence constructed as follows:

—If $\vartheta_i(x,y)$ has $x=y$ as one of its conjuncts, then $\vartheta_i'(x,y)$ is obtained from $\vartheta_i(x,y)$ by replacing every atomic formula (possibly in a negated atomic formula) involving the predicate symbol R by the atomic formula $P(x)$.

—If $\vartheta_i(x,y)$ has $x\neq y$ as one of its conjuncts, then $\vartheta_i'(x,y)$ is obtained from $\vartheta_i(x,y)$ by making the following changes: every occurrence of $R(x,\ldots,x)$ is replaced by $P(x)$; every occurrence of $R(y,\ldots,y)$ is replaced by $P(y)$; every occurrence of $R(\cdots)$ that involves both x and y is deleted.

We now claim that the E_maa-sentence

$$\exists R\forall x\forall y\left(\bigvee_{i=1}^k\vartheta_i(x,y)\right)$$

is equivalent to the E_1aa-sentence

$$\exists P \forall x \forall y \left(\bigvee_{i=1}^{k} \vartheta'_i(x, y) \right).$$

To establish this claim, assume first that $G = (V, E)$ is a graph and R^G is an m-ary relation on V witnessing that $G \models \exists R \forall x \forall y (\bigvee_{i=1}^{k} \vartheta_i(x, y))$. Let P^G be the *diagonal* of R^G, that is, $P^G = \{a \in V : R^G(a, \ldots, a)\}$. Then P^G witnesses that $G \models \exists P \forall x \forall y (\bigvee_{i=1}^{k} \vartheta'_i(x, y))$. In the other direction, assume that P^G is a unary relation on V witnessing that $G \models \exists P \forall x \forall y (\bigvee_{i=1}^{k} \vartheta'_i(x, y))$. We define the following m-ary relation R^G on V:

—For every node a of G such that $a \in P^G$, we put the m-tuple $(a, \ldots, a)$ in R^G. Since $(V, E, P^G) \models \bigvee_{i=1}^{k} \vartheta'_i(x/a, y/a)$, this guarantees that $(V, E, R^G) \models \bigvee_{i=1}^{k} \vartheta_i(x/a, y/a)$.

—Let $a_1, \ldots, a_n$ be an exhaustive list of all nodes of G without repetitions. For every pair (a_r, a_s) of nodes with $r < s$, let $i_0 \leq k$ be such that $(V, E, P^G) \models \vartheta'_{i_0}(x/a_r, y/a_s)$. For every positive occurrence $R(\cdots)$ of R in $\vartheta_{i_0}(x, y)$ that involves both x and y, we put in R^G the m-tuple that has a_r or a_s in each coordinate, depending on whether the variable in the corresponding coordinate of $R(\cdots)$ is x or y. For instance, if R is ternary and the atomic formula $R(y, x, y)$ is one of the conjuncts of ϑ_{i_0}, then we put the triple (a_s, a_r, a_s) in R^G. This guarantees that $(V, E, R^G) \models \vartheta_{i_0}(x/a_r, y/a_s)$ and, hence, $(V, E, R^G) \models \varphi(x/a_r, y/a_s)$. By the aforementioned symmetric property of $\varphi(x, y)$, it follows that also $(V, E, R^G) \models \varphi(x/a_s, y/a_r)$

Consequently, the relation R^G witnesses that $G \models \exists R \forall x \forall y (\bigvee_{i=1}^{k} \vartheta_i(x, y))$. □

By combining Theorem 3.2 and Lemma 3.3, we obtain the following result.

THEOREM 3.4. *Eaa is in* NL *over general graphs and, hence, it is in* PTIME.

4. E*∀∃ *over Self-Loop Free Undirected Graphs*

In order to deal with E^*ae over self-loop free undirected graphs, we first show (in Section 4.1) that this fragment is semantically contained in the monadic fragment E_1^*ae, that is, that for every E^*ae formula Φ there exists an equivalent formula Φ' in E_1^*ae; as a matter of fact, this equivalence holds over arbitrary graphs, that is, the graphs may be infinite and may also contain directed edges and self-loops. Then in Section 4.2 we further analyze E_1^*ae and show that on finite self-loop free undirected graphs the model checking problem for each formula of this fragment is equivalent to a specific graph coloring problem referred to as the *graph saturation problem*. The graph saturation problem is then proven to be in PTIME in Sections 5 and 6.

4.1. E_1^*ae EXPRESSES ALL OF E^*ae.

THEOREM 4.1. *For each E^*ae formula Φ, there exists an E_1^*ae formula Ψ that is equivalent to Φ over all graphs.*

PROOF. Let Φ be a formula of the form

$$\exists P_1, \ldots, P_k \forall x \exists y \varphi,$$

where, without loss of generality, all relation symbols P_i have the same arity l. Without loss of generality, we may assume that $\forall x \forall y(\varphi(x, y) \rightarrow x \neq y)$ holds, since on graphs with at least two nodes

$$\forall x \exists y \varphi(x, y) \iff \forall x \exists y[(x \neq y) \wedge (\varphi(x, y) \vee \varphi(x, x))].$$

We call a tuple p of length l with entries from $\{x, y\}$ an *xy-tuple*. For each xy-tuple p, we define the *opposite xy-tuple* $\tilde{p}$ by interchanging x and y in p. As an example, (x, y, x, x, y) is an xy-tuple of length 5 and (y, x, y, y, x) is its opposite tuple. We call an xy-tuple *proper* if it contains both x and y.

We are going to construct a formula Ψ which is equivalent to Φ and has the following form:

$$\exists \vec{R}_1, \ldots, \vec{R}_k, Q_1, \ldots, Q_6 \; \forall x \; \exists y \; (\psi \wedge \varphi').$$

Here, each $\vec{R}_i$ is a vector of unary relation symbols R_i^p, one for each possible xy-tuple p of length l. The formulas ψ and φ' are quantifier-free.

If Φ holds in a graph G, then there exist relations $\bar{P}_1, \ldots, \bar{P}_k$ and a function $f : V \rightarrow V$ such that, for each vertex v, it holds $(G, \bar{P}_1, \ldots, \bar{P}_k) \models \varphi[x/v, y/f(v)]$.

The principal idea of the proof is to make Ψ true by choosing relations R_i^p, for each i and p, defined as

$$\bar{R}_i^p = \{v \mid p[x/v, y/f(v)] \in \bar{P}_i\}.$$

To this end, let φ' be the formula which results from φ by replacing

—each atomic formula $P_i(p)$, for $i \leq k$ and proper xy-tuples p by $R_i^p(x)$,
—each atomic formula $P_i(x, \ldots, x)$ by $R_i^{(x,\ldots,x)}(x)$, and
—each atomic formula $P_i(y, \ldots, y)$ by $R_i^{(x,\ldots,x)}(y)$.

It is easy to see that if Φ holds in G and f and the $\bar{R}_i^p$ are defined as above then, for each v, we get $G' \models \varphi'[x/v, y/f(v)]$, where G' denotes the extension of G by all the sets $\bar{R}_i^p$.

Our goal is now to assure that Ψ holds in G *only if* Φ holds in G. It might be possible to make $\exists \vec{R}_1, \ldots, \vec{R}_k \; \forall x \; \exists y \; \varphi'$ true by chosing sets R_i^p that do not correspond to any choice of l-ary relations P_i. This can happen, if, for some vertices v, w, it holds $f(v) = w$ and $f(w) = v$, but there is an i and a proper xy-tuple p such that $v \in P_i^p$ but $w \notin P_i^{\tilde{p}}$.

To deal with this problem we make use of the following observation, which will be verified below. The vertices of G can be colored by 6 colors $1, \ldots, 6$ in such a way that, for each vertex v, the colors of v and $f(v)$ are related as indicated in Table I, for example, if v is colored by 3, then $f(v)$ has to be colored by 4 or 1. Note that v and $f(f(v))$ have different colors, unless the color of v is 6.

The six colors are represented by the set variables $Q_1, \ldots, Q_6$. Let ϑ be a formula stating that $Q_i(x)$ holds for exactly one $i \in \{1, \ldots, 6\}$ and let χ be a formula expressing the conditions of Table I. Finally, let ψ be the formula

$$\vartheta \wedge \chi \wedge \left(Q_6(x) \rightarrow \bigwedge_{i, p \text{ proper}} \left(R_i^p(x) \longleftrightarrow R_i^{\tilde{p}}(y) \right) \right).$$

TABLE I. HOW THE COLORS OF v AND $f(v)$ ARE RELATED

v	$f(v)$
1	2
2	3
3	4 or 1
4	5 or 1
5	6 or 1
6	6

and Ψ be

$$\exists \vec{R}_1, \ldots, \vec{R}_k, Q_1, \ldots, Q_6 \forall x \, \exists y \; \psi \wedge \varphi'.$$

We show that Φ and Ψ are equivalent on all graphs G.

Assume first that Φ holds for G. Let $P_1, \ldots, P_k$ and f be such that, for each vertex v, $(G, P_1, \ldots, P_k) \models \varphi[x/v, y/f(v)]$. We define the sets R_i^p as before. This implies that, for each v, $\varphi'[x/v, y/f(v)]$ holds in G', the extension of G by the sets R_i^p. We show next that sets $Q_1, \ldots, Q_6$ can be found such that, for each v, $\psi[x/v, y/f(v)]$ holds in G'', the extension of G' by the sets Q_j. We define the sets Q_j, by inspecting the (weakly) connected components of the graph of f. If the set of vertices is finite, then each such component consists of a directed cycle C of length at least two (recall that always $f(v) \neq v$), and zero or more directed trees that are rooted at vertices of C. In these trees, the edges are directed from the leaves to the root. If the set of vertices is infinite, then the components of f may also include infinite chains. Components with a cycle C with more than two vertices can be colored with $Q_1, \ldots, Q_5$ such that each vertex fulfils χ. If C contains $3m$ vertices, for some m, they can be colored by Q_1, Q_2, Q_3 in a round robin manner. If it contains $3m+1$ vertices, then color Q_4 is used for the last vertex. Analogously, if it contains $3m+2$ vertices, the colors Q_4 and Q_5 are used for the last two vertices. Finally, if the component is an infinite chain, then it can be colored with just three colors.

The components with a cycle of size two can be colored such that the vertices of the cycle are colored with Q_6 and the remaining vertices with $Q_1, \ldots, Q_5$ in a way that assures that χ holds for all vertices. If a vertex v gets the color Q_6, then $f(v) \neq v$ but $f(f(v)) = v$ and therefore $p[x/v, y/f(v)] \in P_i \Longleftrightarrow \tilde{p}[x/f(v), y/v] \in P_i$, for all i and p. Hence, $G'' \models ((\psi \wedge \varphi') \vee (x = y \wedge \varphi''))[x/v, y/f(v)]$ for all v and therefore $G \models \Psi$.

For the converse direction, assume that $G \models \Psi$. Let f be a corresponding function, and let $\bar{Q}_1, \ldots, \bar{Q}_6$ and, for each i and p, $\bar{R}_i^p$ be corresponding sets. We define relations $\bar{P}_1, \ldots, \bar{P}_k$ as follows:

$$\bar{P}_i = \left\{ p[x/v, y/f(v)] \mid v \in \bar{R}_i^p \text{ for some } p \right\} \cup \left\{ (v, \ldots, v) \mid v \in \bar{R}_i^{(x,\ldots,x)} \right\}.$$

This definition ensures that, for each i and p,

$$\bar{R}_i^p = \{ v \mid p[x/v, y/f(v)] \in \bar{P}_i \},$$

as, for every vertex v,

—either $v \in \bar{Q}_6$ and therefore $v \in \bar{R}_i^p$ if and only if $f(v) \in \bar{R}_i^{\tilde{p}}$,
—or $f(f(v))$ has a different color than v and therefore $v \neq f(f(v))$.

It is straightforward to see that $G \models \Phi$. $\square$

4.2. E_1^*ae MODEL-CHECKING AND GRAPH COLORING. In this section, we consider E_1^*ae-formulas over self-loop free undirected graphs $G = (V, E)$. We show that in this case the model-checking problem for such formulas is equivalent to a certain graph coloring problem, which we call the *graph saturation problem*.

Let Φ be an E_1^*ae-formula of the form $\exists P_1, \ldots, P_k \forall x \exists y \varphi$, where φ is a quantifier-free formula. As in the proof of Theorem 4.1, we may assume, without loss of generality, that $\forall x \forall y(\varphi(x, y) \rightarrow x \neq y)$ holds. Every satisfiable quantifier-free formula is logically equivalent to a disjunction of *complete consistent types*, that is to say, conjunctions of atomic or negated atomic formulas involving the variables x, y and the symbols E, $=$, $P_1, \ldots, P_k$ and such that for each atomic formula in these symbols and variables either the atomic formula itself or its negation occurs as a conjunct. In particular, this holds true for the quantifier-free formula $\varphi(x, y)$. Moreover, since $\forall x \forall y(\varphi(x, y) \rightarrow x \neq y)$ holds and since we are focusing on self-loop free undirected graphs, we may assume that no disjunct contains one of the atomic formulas $x = y$, $E(x, x)$, $E(y, y)$ as a conjunct. We may also simplify each disjunct by eliminating occurrences of the atomic formula $E(y, x)$ or of the atomic formula $\neg E(y, x)$ (depending on which of the two occurs), while keeping occurrences of the atomic formula $E(x, y)$ and the negated atomic formula $\neg E(x, y)$. These considerations motivate the following definition.

Definition 4.2. An E_1^*ae-formula Φ is in *normal form* if it is of the form

$$\exists P_1, \ldots, P_k \forall x \exists y \varphi(x, y),$$

where φ is a disjunction of conjunctions of atomic and negated atomic formulas such that each disjunct δ of φ has the following conjuncts:

—For each relation symbol P_i, $1 \leq i \leq k$, and each variable $\xi \in \{x, y\}$, either the atomic formula $P_i(\xi)$ or the negated atomic formula $\neg P_i(\xi)$ is a conjunct of δ.

—Either the atomic formula $E(x, y)$ or the negated atomic formula $\neg E(x, y)$ is a conjunct of δ.

—The negated atomic formula $x \neq y$ is a conjunct of δ.

The remarks preceding Definition 4.2 show that over self-loop free undirected graphs every E_1^*ae-formula is logically equivalent to one in normal form. For this reason, from now on, we will work with E_1^*ae-formulas in normal form.

If ξ is a variable, then there are 2^k different complete specifications $(\neg)P_1(\xi) \wedge (\neg)P_2(\xi) \wedge \cdots \wedge (\neg)P_k(\xi)$ that may occur in a disjunct δ of an E_1^*ae-formula in normal form. Each such specification can be identified with a "coloring" by a color $C_1, \ldots, C_r$ from a set of colors Colors $= \{C_1, \ldots, C_r\}$, where $r = 2^k$. We write $C_i(\xi)$ to indicate the "coloring" of the variable ξ by the complete specification corresponding to color C_i. Thus, each disjunct δ of an E_1^*ae-formula in normal form consists of a coloring $C_i(x)$, a coloring $C_j(y)$, the atomic formula $E(x, y)$ or the negated atomic formula $\neg E(x, y)$, and the inequality $x \neq y$.

Definition 4.3. A *pattern graph* is a directed graph with no isolated nodes and such that each arc is labeled $\oplus$ or $\ominus$ (note that a pattern graph may have two arcs with different labels from a vertex C to a vertex D).

With each E_1^*ae-formula Φ in normal form, we associate a pattern graph $P(\Phi)$ as follows. The vertices of $P(\Phi)$ consist of the colors (complete types for the set variables) occurring in the disjuncts of Φ. There is an edge (C_i, C_j) labeled $\oplus$ in $P(\Phi)$ iff there exists a disjunct δ in Φ such that δ contains as conjuncts $C_i(x)$ and $C_j(y)$ and $E(x, y)$. There is an edge (C_i, C_j) labeled $\ominus$ in $P(\Phi)$ iff there exists a disjunct δ in Φ such that δ contains as conjuncts $C_i(x)$ and $C_j(y)$ and $\neg E(x, y)$.

Definition 4.4. If $P = (\text{Colors, Arcs})$ is a pattern graph and $G = (V, E)$ is an undirected graph, then a *coloring* of G with respect to P is a function $col : V \longrightarrow$ Colors. A *witness function* for a given coloring *col* is a function $wit : V \longrightarrow V$ such that the following hold for each $x \in V$:

—$x \neq w(x)$;
—if $(x, wit(x)) \in E$, then $(col(x),\ col(wit(x)))$ is an arc labeled $\oplus$ in P; and
—if $(x, wit(x)) \notin E$, then $(col(x), col(wit(x)))$ is an arc labeled $\ominus$ in P.

A coloring is *legal* if there exists a witness function for it.

Definition 4.5. Given a pattern graph $P = (\text{Colors, Arcs})$, the *pattern graph saturation problem* SATU_P for P is defined as follows:

Instance: A self-loop free undirected graph G.
Question: Is there a legal coloring of G with respect to P?

We denote by SATU(P) the set of all self-loop free undirected graphs that are yes-instances of SATU_P.

Figure 5 shows two pattern graphs P_1 and P_2 and a graph G which is a yes instance of $SATU_{P_1}$ via the exhibited coloring and via a witness function w defined as follows: w maps each of the five green-colored nodes to the unique yellow node, the unique red node to the blue one, the yellow node to the blue one as well, and the blue node to one of its green neighbours. On the other hand, G is a negative instance of SATU_{P_2}; in other terms, $G \in \text{SATU}(P_1)$ and $G \notin \text{SATU}(P_2)$.

THEOREM 4.6. *For each E_1^*ae-formula Φ in normal form and each self-loop free undirected graph G, we have that $G \models \Phi$ iff $G \in SATU(P(\Phi))$.*

PROOF. Although the proof is quite straightforward, we include it here for mere completeness reasons.

Let $G = (V, E)$. If $G \models \Phi$, then there exists an interpretation $I : \overline{P_1}, \ldots, \overline{P_k}$ of the set variables, such that for each $v \in V$, there exists a disjunct $\delta(v)$ of Φ and a vertex $f(v) \in V$ such that $(G, \overline{P_1}, \ldots, \overline{P_k}) \models \delta(v)[x/v, y/f(v)]$. If $\xi \in \{x, y\}$ and if δ is a disjunct of Φ, then let $qual(\xi, \delta)$ denote the color qualification of ξ in δ. We define the coloring *col* as follows: $\forall v \in V\ col(v) = qual(x, \delta(v))$. If $(v,\ f(v)) \in E$, then, given that the disjunct $\delta(v)[x/v, y/f(v)]$ is satisfied, it must contain the literal $E(x, y)$, hence, by definition of $P(\Phi)$, $(col(v), col(f(v)))$ is an arc labeled $\oplus$ in $P(\Phi)$. Similarly, if $(v,\ f(v)) \notin E$, then, given that the disjunct $\delta(v)[x/v, y/f(v)]$ is satisfied, it must contain the literal $\neg E(x, y)$, hence $(col(v), col(f(v)))$ is an arc labeled $\ominus$ in $P(\Phi)$. It follows that f is a witness function for *col* and thus $G \in \text{SATU}(P)$.

Conversely, assume that $G \in \text{SATU}(P)$. Let *col* be a legal coloring of G with respect to $P(\Phi)$ and let *wit* be a witness function for *col*. Let $I : \overline{P_1}, \ldots, \overline{P_k}$ be the interpretation of the set variables $P_1, \ldots, P_k$ corresponding to *col*. Let $v \in V$ be a vertex of G. We distinguish between two cases.

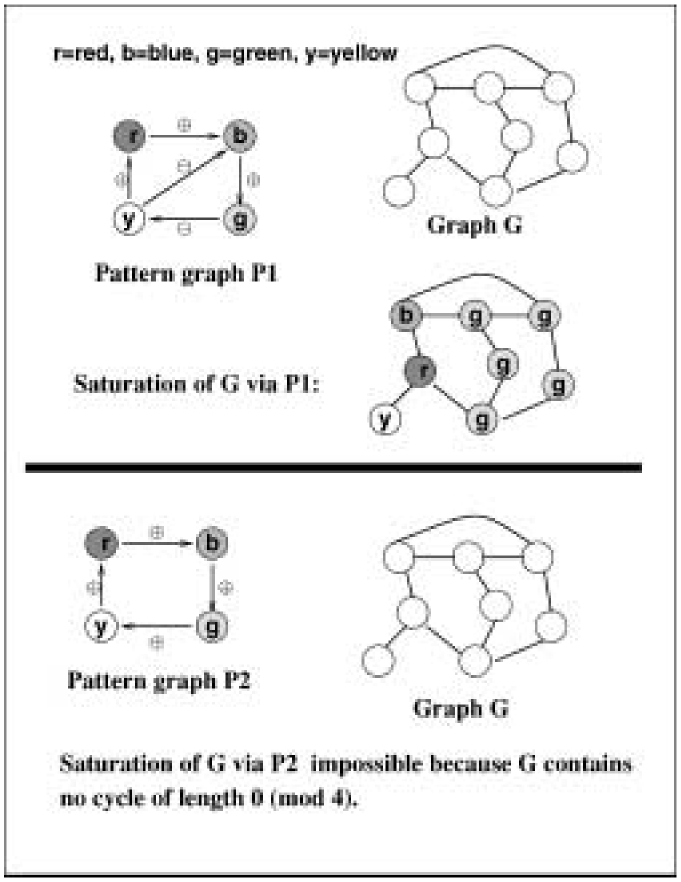
r=red, b=blue, g=green, y=yellow
Pattern graph P1
Graph G
Saturation of G via P1:
Pattern graph P2
Graph G
Saturation of G via P2 impossible because G contains no cycle of length 0 (mod 4).

FIG. 5. A yes and a no instance of SATU.

(1) $(v, wit(v)) \in E$. Then $(col(v),\ col(wit(v)))$ is an arc labeled $\oplus$ in $P(\Phi)$. Therefore, by definition of $P(\Phi)$, there exists a disjunct $\delta(v)$ of Φ such that $\delta(v)$ contains as conjuncts the qualifications $col(v)(x)$ and $col(wit(v))(y)$ and the edge literal $E(x, y)$. It then holds that $(G, P_1, \ldots, P_k) \models \delta(v)[x/v, y/wit(v)]$. Note that $col(v)$ represents a unary relation symbol C_i.

(2) $(v, wit(v)) \notin E$. Then $(col(v), col(wit(v)))$ is an arc labeled $\ominus$ in $P(\Phi)$. Therefore, by definition of $P(\Phi)$, there exists a disjunct $\delta(v)$ of Φ such that $\delta(v)$ contains as conjuncts the qualifications $col(v)(x)$ and $col(wit(v))(y)$ and the edge literal $\neg E(x, y)$. It then holds that $(G, P_1, \ldots, P_k) \models \delta(v)[x/v, y/wit(v)]$.

In summary, for every $v \in V$ there exists a $w = wit(v) \in V$ and a disjunct $\delta(v)$ of Φ such that $(G, P_1, \ldots, P_k) \models \delta(v)[x/v, y/w]$. This just means that $G \models \Phi$. $\square$

Conversely, each saturation problem SATU(P) can be expressed by an E_1^*ae-formula in normal form.

THEOREM 4.7. *For each pattern graph P, there exists an E_1^*ae formula Φ_P in normal form such that for each undirected self-loop free graph G we have that $G \in SATU(P)$ iff $G \models \Phi_P$.*

PROOF. Let Colors $= \{Q_1, \ldots, Q_n\}$ and let $P = (\text{Colors}, \text{Arcs})$ be a pattern graph. For $1 \le i \le n$ and $\xi \in \{x, y\}$, identify Q_i with a monadic predicate symbol and define

$$C_i(\xi) \;=\; Q_i(\xi) \wedge \bigwedge_{j \neq i, 1 \le j \le n} \neg Q_j(\xi).$$

For each edge $e = (Q_i, Q_j)$ of P, define $\delta(e)$ as:

$$\delta(e) = \begin{cases} C_i(x) \wedge C_j(y) \wedge E(x, y) \wedge x \neq y & \text{if } e \text{ is labeled } \oplus \\ C_i(x) \wedge C_j(y) \wedge \neg E(x, y) \wedge x \neq y & \text{if } e \text{ is labeled } \ominus \end{cases}$$

Define the formula Φ_P by:

$$\Phi_P \;=\; \exists Q_1, \ldots, Q_n \, \forall x \, \exists y \bigvee_{e \in E_P} \delta(e).$$

Observe that (up to renaming of vertices) it holds that $P(\Phi_P) = P$. Thus, by Theorem 4.6, $G \in \text{SATU}(P)$ iff $G \models \Phi_P$. $\square$

Consequently, the model checking problem for E^*ae-formulas over self-loop free undirected graphs coincides with the pattern graph saturation problem. In Sections 5 and 6, we will show the following result.

THEOREM 4.8. *The pattern graph saturation problem is solvable in polynomial time.*

Combining Theorems 4.1, 4.6 and 4.8, we get the following result which completes the proof of Theorem 1.1.

THEOREM 4.9. *Eae is in* PTIME *over self-loop free undirected graphs.*

5. *Saturation using Pure Cycles*

This section and the following one are devoted to the proof of Theorem 4.8. This section treats the case of so-called pure pattern graphs, a notion that will be defined shortly. The case of general pattern graphs is investigated in Section 6.

Proviso 5.1. In this section, the term, *graph*, always refers to self-loop free undirected graphs whereas *pattern graph* refers to a directed graph with edge-labels from $\{\ominus, \oplus\}$.

The following structural properties of pattern graphs are relevant for our study. A *positive (negative) cycle* of the pattern graph is a directed cycle all of whose edges are labeled $\oplus$ ($\ominus$). A *mixed cycle* of the pattern graph is a directed cycle containing at least one edge labeled $\ominus$ and at least one edge labeled $\oplus$.

A pattern graph is:

—*positive* if all its edges are labeled $\oplus$.
—*negative* if all its edges are labeled $\ominus$.
—*monotone* if it is positive or negative;
—*acyclic* if P does not contain a directed cycle (loops do count as cycles).
—*mixed* if P contains a mixed cycle;
—*pure* if it is not mixed.

Note that the class of monotone pattern graphs and the class of acyclic pattern graphs are both properly contained in the class of pure pattern graphs.

The following two simple lemmas follow easily from Definitions 4.4 and 4.5.

LEMMA 5.2. *If P is a positive pattern graph, and G is a loop free undirected graph, then $G \in SATU(P)$ iff for each connected component K of G, $K \in SATU(P)$.*

If $G = (V, E)$ is a graph, then G^c denotes its *complement*, that is, $G^c = (V, V^2 - (E \cup \{(v, v) : v \in V\}))$ is the self-loop free undirected graph obtained from G by turning edges to nonedges, and vice-versa, between pairs of distinct nodes. If P is a pattern graph, we write $P^\bullet$ for the pattern graph obtained from P by replacing every $\ominus$-labeled edge by a $\oplus$-labeled edge and vice-versa.

LEMMA 5.3. *$G \in SATU(P)$ iff $G^c \in SATU(P^\bullet)$.*

As already mentioned, we consider in this section cases of the saturation problem in which only pure cycles of the pattern graphs matter. In Section 5.1, we prove that the saturation problem for pure pattern graphs is in PTIME. In Section 5.2, we generalize this result to a setting where the pattern graph may contain some mixed cycles, but these mixed cycles cannot be used for saturation.

5.1. THE SATURATION PROBLEM FOR PURE PATTERN GRAPHS. We prove that the saturation problem SATU(P) for a pure pattern graph P is tractable.

First of all, we will see that SATU(P) can be efficiently decided for input graphs of bounded tree-width. We assume the reader to be familiar with the notion of *treewidth* of undirected graphs [Robertson and Seymour 1986; Downey and Fellows 1999].

We denote by LINTIME the class of all problems solvable in linear time on a random access machine. As usual, MSO denotes monadic second-order logic. The next proposition is a restricted version of a well-known theorem of Courcelle.

THEOREM 5.4 (COURCELLE 1990). *Model checking for MSO-formulas over graphs of constant-bounded treewidth is in* LINTIME.

Since SATU(P) is definable by a MSO-formula, by Theorem 4.7, we derive the following corollary.

COROLLARY 5.5. *For each fixed constant k and pattern graph P, the problem of deciding whether a graph of treewidth $\leq k$ belongs to SATU(P) is in* LINTIME.

Recognizing whether a graph has bounded treewidth is an easy problem.

THEOREM 5.6 (BODLAENDER 1996). *For each fixed constant k, the problem of checking whether a graph has treewidth $\leq k$ is in* LINTIME.

Hence, for fixed k, an algorithm can check whether the input graph has treewidth $\leq$k and if this the case decide whether it is in SATU(P) in linear time. It only remains to show that the graph saturation problem can be solved efficiently also for input graphs of large tree-width. Before we start with this case, we state some easy observations.

A vertex v of a pattern graph P is *useful* if it lies on a directed cycle or there is a directed path from that node to a node lying on a cycle. A vertex that is not useful is termed *useless*. The following lemma shows that it is sufficient to consider pattern

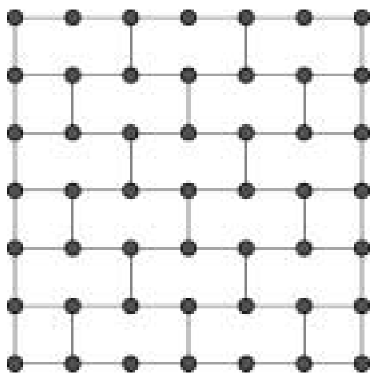

FIG. 6. The grid H_7.

graphs all of whose vertices are useful. If P is a pattern graph, then P^* denotes the pattern graph obtained from P by dropping all useless vertices.

LEMMA 5.7. *For each pattern graph P, it holds that SATU(P) = SATU(P^*).*

PROOF. The inclusion SATU$(P^*) \subseteq$ SATU(P) follows trivially from the definition of the saturation problem. Let us prove SATU$(P) \subseteq$ SATU(P^*). Let $G = (V, E) \in$ SATU(P), where $P =$ (Colors, Arcs). Let $P^* =$ (Colors*, Arcs*). Then there exists a legal coloring *col* of G with respect to P and an associated witness function w. Assume for some vertex $v \in V$, $col(v)$ is a useless color. Then $wit(v)$, $wit(wit(v))$, etc. must all be colored by useless colors, and at some point we arrive at a vertex $w = wit^i(v)$ such that $col(w)$ is a vertex having out-degree zero in the pattern graph P. Clearly, $wit(w)$ cannot be colored correctly. Contradiction. Thus, $col(V) \subseteq$ Colors* and thus the function $col^* : V \longrightarrow$ Colors* where $\forall x \in V, col^*(x) = col(x)$ is a legal coloring of G with respect to P^*, whence $G \in$ SATU(P^*). $\square$

COROLLARY 5.8. *If P is acyclic, then the only element of SATU(P) is the empty graph.*

To deal with cyclic pattern graphs and input graphs of large tree-width, we need a number of auxiliary results. We start with some known results on grids and treewidth.

As in Thomassen [1988], we define the grid H_k (where k is a natural number) as follows: Take k disjoint paths $\pi_1, \ldots, \pi_k$ where $\pi_i : x_{1,i}, x_{2,i}, \ldots, x_{k,i}$ and add all edges $\{x_{i,j}, x_{i,j+1}\}$ where $i + j$ is even. Also add the edges of the paths $\pi'_1 = x_{1,1}, x_{1,2}, \ldots, x_{1,k}$ and $\pi'_2 = x_{k,1}, x_{k,2}, \ldots, x_{k,k}$. The grid H_7 is depicted in Figure 6.

A *subdivision* of an undirected graph G is a graph obtained from G by replacing some edges of G with simple paths (i.e., by "subdividing" edges of G). The following fundamental result is well known.

THEOREM 5.9 (ROBERTSON AND SEYMOUR 1984). *For every natural number k, there exists a natural number $g(k)$ such that every graph of treewidth at least $g(k)$ contains a subdivision of H_k.*

The next result is due to Carsten Thomassen [1988].

THEOREM 5.10 (THOMASSEN 1988). *For all natural numbers k and m, there exists a natural number $h(k, m)$ such that every subdivision of $H_{h(k,m)}$ contains as subgraph a subdivision H_k^* of H_k, where H_k^* is obtained from H_k by replacing each edge of H_k by a path of length* $0(\textit{mod}\, m)$.

A (directed or undirected) graph consisting of a single (directed or undirected) cycle and a (directed or undirected) path such that one of the endpoints of the path is also a vertex of the cycle and the path and the cycle have no other vertices in common is called a *racket* (the form is akin to a tennis racket). A racket whose cycle has length c and whose path has length p is called a (c, p)-*racket*. Cycles are special cases of rackets with path length 0.

If a racket R is not a cycle, then it has a unique vertex of (total) degree 1 (i.e., the outdegree and the indegree add up to 1). This vertex is called the *endpoint* of R and is denoted by $\textit{endpt}(R)$. Moreover, the unique vertex belonging both to the path and to the cycle is called the *junction* of R and is denoted by $\textit{junct}(R)$.

The following proposition was shown (in more general form) by Thomassen [1988]:

THEOREM 5.11 (THOMASSEN 1988). *For every integer m there exists an integer $r(m)$ such that every self-loop free undirected graph having treewidth $\geq r(m)$ contains a cycle whose length is a positive multiple of m.*

The following lemma slightly generalizes Proposition 5.11.

LEMMA 5.12. *For every pair (c, p) of integers there exists an integer $f(c, p)$ such that every self-loop free undirected graph having treewidth $\geq f(c, p)$ has a (c', p)-racket such that c' is a multiple of c.*

PROOF. If c is odd, let $c_1 = 2c$, otherwise, let $c_1 := c$. Let $k = \max(c_1, p) + 1$. Observe that H_k contains a (c_1, p)-racket R as subgraph and that c_1 is a multiple of c. By Proposition 5.10 (by taking $m = c$), there exists a natural number $h(k, c)$ such that every subdivision of $H_{h(k,c)}$ contains as subgraph a subdivision H_k^* of H_k where H_k^* is obtained from H_k by replacing each edge e of H_k by a path $\rho(e)$ of length $0(\textit{mod}\, c)$. Let $f(c, p) = g(h(k, c))$, where g is as specified by Proposition 5.9. By Proposition 5.9, every self-loop free undirected graph G of treewidth $\geq f(c, p)$ contains a subdivision of $H_{h(k,c)}$. Thus G contains a subgraph H^* as specified above. This H^* contains the image of R by the edge-replacement ρ where every edge e of R is replaced by a path $\rho(e)$ of length $0(\textit{mod}\, c)$. Observe that $\rho(R)$ is a racket. Let c' be the length of the cycle of $\rho(R)$ and let p' be the length of the path of $\rho(R)$. Clearly, c' is a multiple of c and $p' \geq p$. By shortening the path of $\rho(R)$ to length p, we obtain the desired racket. □

A subgraph Q of a pattern graph P is *positive* (*negative*) if all edges in Q are labeled $\oplus$ ($\ominus$).

LEMMA 5.13. *Let $P = (\textit{Colors}, \textit{Arcs})$ be a pattern graph having as subgraph a positive (c, p)-racket R. Let $G = (V, E)$ be a connected self-loop free undirected graph having as subgraph a (c', p)-racket S, where c' is a multiple of c. Then $G \in \textit{SATU}(P)$. Moreover, there exists a legal coloring col of G with respect to P such that each vertex of the cycle of R is an element of $\textit{col}(V)$ and, if $p \neq 0$, then also $\textit{endpt}(R) \in \textit{col}(V)$.*

PROOF. Let $r = c'/c$.

Let R_c be the cycle of R. R_c can be written as: $C^1, C^2, \dots, C^c, C^1$. Denote by R_p the path of R. If $p \neq 0$, assume without loss of generality that $junct(R) = C^1$. In case R_p is nonempty, it can be written

$$R_p : D^p,\ D^{p-1},\ D^{p-2}, \dots, D^2,\ D^1,\ C^1.$$

Here, in case $p > 0$, we have $endpt(R) = D^p$.

Let S_c be the cycle of racket S. Impose an arbitrary orientation S' on the cycle S_c. S' can then be written as:

$$S' : v_1^1,\ v_2^1, \dots, v_c^1, v_1^2,\ v_2^2, \dots, v_c^2, \dots, v_1^r,\ v_2^r, \dots, v_c^r,\ v_1^1.$$

Denote by S_p the path of S. If $p \neq 0$, assume without loss of generality that $junct(S) = v_1^1$. In case S_p is nonempty, it can be written

$$S_p : w_p,\ w_{p-1},\ w_{p-2}, \dots, w_2, w_1, v_1^1.$$

Here, in case $p > 0$, we have $endpt(S) = w_p$.

The successor of a vertex α in a cycle (R_c or S_c) is denoted by $succ(\alpha)$; the predecessor of α by $pred(\alpha)$.

We distinguish two cases according to the structure of R.

Case 1. R is a cycle (i.e., $p = 0$).

We define a coloring function $col : V \longrightarrow$ Colors and a witness function $V \longrightarrow V$ by the following "saturation" procedure.

```
(1) For $1 \leq i \leq c$ and $1 \leq j \leq r$, let $col(v_i^j) := C^i$.
(2) For all vertices $v$ of $S_c$, let $wit(v) = succ(v)$.
(3) Let $Settled$ be initially the set of all vertices of the cycle $S = S_c$.
(4) WHILE $Settled \neq V$ DO
—Choose a vertex $v \notin Settled$ such that $v$ is a neighbor of a vertex $w$ in $Settled$.
—Let $col(v) := pred(col(w))$ and let $wit(v) := w$.
—$Settled := Settled \cup \{v\}$.
```

Given that G is connected, at the end of this process, *col* and *wit* are total functions and *col* is a legal coloring of G with respect to P. In fact, for each $x \in V$, $(x, wit(x)) \in E$ and $(col(x), col(wit(x)))$ is an arc of P labeled $\oplus$. It follows that $G \in \text{SATU}(P)$. Note also that each vertex of the cycle of R appears in $col(V)$.

Case 2. R is not a cycle (i.e., $p > 0$).

In this case, we could color G in the same way as in Case 1, by starting to color the cycle S_c according to R_c, and then coloring the rest of the graph by applying the saturation procedure as described in Case 1. Although this shows that $G \in \text{SATU}(P)$, the coloring obtained this way does *not* fulfill the requirement that $endpt(R) \in col(V)$.

A naive approach for achieving $endpt(R) \in col(V)$ would be to color S_c in accordance with to R_c as in Case 1, and to color S_p in accordance with R_p. This is possible, but it may then not be possible to correctly color the rest of the graph. In fact, problems may already arise with the neighbors of $endpt(S)$ in G. Recall

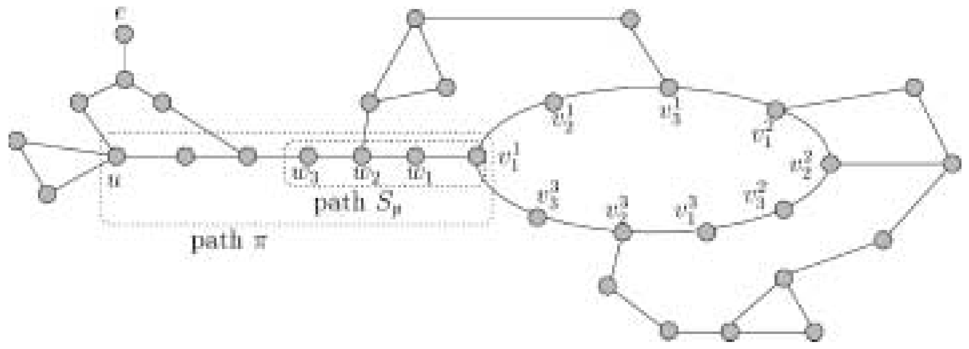

FIG. 7. An example of a graph G with a (9,3)-racket.

that *endpt*(S) is colored $D^p =$ *endpt*(R). However, *endpt*(S) may have a neighbor b of degree 1 (i.e., a neighbor which is otherwise isolated in G). In case D^p has no incoming edge in P, b is not colorable. We thus need a more astute coloring method. Figure 7 illustrates the proof.

Let $G^* := (G - S_c) \cup \{v_1^1\}$, that is, G^* is the subgraph induced by v_1^1 together with all vertices outside S_c.

Let σ be a longest simple path in G^* starting at v_1^1. Note that *length*$(\sigma) \geq p$ must hold, given that already the path S_p is of length p. Denote by e the end of σ. Let u be the (unique) vertex of σ such that the segment $[u, e]$ of σ has length p. Denote by π the segment $[v_1^1, u]$ of σ. π is a simple path, possibly of length zero (in case $u = v_1^1$).

CLAIM 1. *The length of any longest simple path in the subgraph* $[G - (S_c \cup \pi)] \cup \{u\}$ *starting at u is exactly p.*

To prove the claim, observe that the segment $[u, e]$ which lies entirely in $[G - (S_c \cup \pi)] \cup \{u\}$ has precisely length p. Now assume that there exists a path γ in $[G - (S_c \cup \pi)] \cup \{u\}$ having length $p + k$ where $k > 0$. Then the concatenation of π with γ is a simple path of length *length*$(\sigma) + k$ lying in G^* and starting at v_1^1. This contradicts the assumption that σ is a longest simple path in G^* starting at v_1^1. The claim is proved.

We are now ready to color G.

We define a coloring function $col : V \longrightarrow$ Colors and a witness function $wit : V \longrightarrow V$ through the following five steps:

(1) Let K denote the component of $[G - (S_c \cup \pi)] \cup \{u\}$ which contains u. We color K as follows: First, u is colored C^1, that is, $col(u) := C^1$. Then, for each vertex $v \neq u$ in K, let $d(v)$ denote the distance between u and v in K and color v with color $D^{d(v)}$, that is, let $col(v) := D^{d(v)}$. Note that by Claim 1, for each $v \in K$ it holds that $d(v) \leq p$. Thus, the described coloring is well defined and covers all vertices of K. Observe that there exists at least one vertex v such that $col(v) = D^p =$ *endpt*(R).

(2) We assign a witness $wit(v)$ to each vertex $v \in K$ as follows: First, if $u \neq v_1^1$, then let $wit(u) = w$, where w is u's unique neighbor in π. If $u = v_1^1$, then $wit(u) :=$ *succ*(v_1^1). For each other vertex v of K, let $wit(v) = v'$, where v' is any neighbor of v such that $d(v') = d(v) - 1$. Note that wit is totally defined on

K and that for each vertex v of $K - \{v\}$, the pair $(v, wit(v))$ is legally colored, that is, is an edge of R.

(3) We assign colors and witnesses to the vertices of $\pi - \{u\}$. Let $\ell := length(\pi)$. We can write π as: $u = t_1, t_2, \dots, t_{\ell-1}, t_\ell = v_1^1 = junct(S_c)$. We define for all $1 < i \leq \ell$: $col(t_i) := C^{i\ (mod\ c}$. Thus, for example, t_2 is colored C^2, and t_{c+1} is colored C^1, and so on. Moreover, for all $1 < i < \ell$, let $wit(t_i) = t_{i+1}$. Finally, for $t_\ell = v_1^1$ we define: $wit(v_1^1) = succ(v_1^1)$. Note that for each vertex v of $\pi - \{v_1^1\}$, the pair $(v, wit(v))$ is legally colored.

(4) We assign colors and witnesses to the vertices of $S_c - \{v_1^1\}$. Recall that v_1^1 was already colored in Step (3). Let $col(succ(v_1^1)) := succ(col(v_1^1))$, $col(succ(succ(v_1^1))) := succ(col(succ(v_1^1))) = succ(succ(col(v_1^1)))$, and so on. More formally, for each $1 \leq i < c$, let $col(succ^i(v_1^1)) := succ^i(col(v_1^1))$. Moreover, for each $v \in S_c - \{v_1^1\}$, let $wit(v) := succ(v)$. Thus, each vertex v in S_c has a well-defined color and a well-defined witness, and the pair $(v, wit(v))$ is legally colored.

(5) We assign colors and witnesses to all remaining vertices. It is crucial at this point to observe that each still uncolored vertex v of G is connected to a (already colored) vertex of $(\pi \cup S_c) - \{u\}$ via a path that does not cross K. In fact, since the connected component K of $[G - (S_c \cup \pi)] \cup \{u\}$ was completely colored in step 1, v must belong to some other connected component K' of $[G - (S_c \cup \pi)] \cup \{u\}$. Since G is connected, K' must be adjacent to $(S_c \cup \pi) - \{u\}$. Thus, there is a path from v to some vertex $v' \in (S_c \cup \pi) - \{u\}$ such that all vertices of this path except v' belong to K' and are so far uncolored.

We can thus apply a similar saturation procedure as in Case 1 and will eventually reach all remaining vertices. In particular:

(a) Let *Settled* be initially the set $(\pi \cup S_c) - \{u\}$. Notice that all vertices in *Settled* are colored by some color from $\{C^1, \dots, C^c\}$, that is, from R_c.

(b) WHILE $Settled \neq V$ DO
—Choose a vertex $v \notin Settled$ such that v is a neighbor of a vertex w in *Settled*.
—Let $col(v) := pred(col(w))$ and $wit(v) = w$.
—$Settled := Settled \cup \{v\}$.

This concludes Step (5).

At the end of Step (5), *col* and *wit* are total functions and *col* is a legal coloring of G with respect to P. It follows that $G \in \text{SATU}(P)$. Moreover, each vertex of the cycle of R appears in $col(V)$ and also $endpt(R) \in col(V)$. □

We are now ready for the main result of this section.

THEOREM 5.14. *If P is a pure pattern graph, then SATU$(P) \in$* PTIME.

PROOF. By Corollary 5.8, for acyclic P, SATU(P) consists of the empty graph, thus SATU$(P) \in$ PTIME. Assume P is cyclic. If P contains some useless vertices, we can eliminate them by Lemma 5.7, that is, we can replace P by P^*. We thus assume all vertices of P are useful.

We distinguish three cases.

Case 1. P has both positive and negative cycles.

We consider two subcases.

(a) G is *connected*. Let c be the length of an arbitrary positive cycle R in P. Let $r(c)$ be as in Proposition 5.11. By Corollary 5.5, there is a linear-time procedure *satucheck*$_P(H)$, which for each self-loop free undirected input graph H of treewidth $\leq r(c)$ determines whether $H \in$ SATU(P).

To check that $G \in$ SATU(P), perform the following steps:

—Check whether G has treewidth $\leq r(c)$.
—*If* G has treewidth $\leq r(c)$, THEN RETURN(*satucheck*$_P(G)$).
—IF G has treewidth $r(c)$, THEN RETURN(*true*).

The last step is justified as follows. IF G has treewidth $>r(c)$, then G contains by Lemma 5.11 a cycle of length 0 (*mod* c), that is, a $(c', 0)$-racket, where c' is a multiple of c. Given that R is a positive $(c, 0)$-racket, by Lemma 5.13, $G \in$ SATU(P).

(b) G is *disconnected*. Then G^c is connected and $P^\bullet$ has a positive and a negative cycle. Proceed by checking $G^c \in$ SATU$(P^\bullet)$ according to (a) and using Lemma 5.3.

Case 2. All cycles of P are positive.

We consider two subcases:

(a) *P is positive*. By Lemma 5.2, $G \in$ SATU(P) iff $K \in$ SATU(P) for each connected component K of G. This check is done for each K according to Case 1(a). It should be noted that the proof of Case 1(a) only made use of the existence of one positive cycle.

(b) *P has some edges labeled* $\ominus$. Let $e = (D^i, D^j)$ be such an edge. Given that P contains only useful vertices, D^j is useful and there is a simple path σ (of possible length zero) from D^j to a positive cycle T in P. If σ does not contain any negative edge after e, then let $D^p := D^j$; otherwise, let D^p be the first vertex of σ reachable from D^j such that there is no negative arc on σ after D^p. Let D^- be the predecessor of D^p on σ and retain that the arc (D^-, D^p) is negative.

Denote by ω the subpath of σ starting at D^p and ending at the first vertex C^1 of the cycle T. (Note that it may happen that ω is of length zero, that is, $D^p = C^1$.) Let R be the union of ω and T, let p be the length of ω, and let c be the length of T. Note that R is a positive (c, p)-racket of P.

Let $f(c, p)$ be as in Lemma 5.12. By Corollary 5.5 there is a linear-time procedure *satucheck*$'_P(H)$ which for each self-loop free undirected input graph H of treewidth $\leq f(c, p)$ determines whether $H \in$ SATU(P).

To check whether $G \in$ SATU(P), proceed as follows.

—Compute the connected components of G.
—Check, whether each connected component has treewith $\leq f(c, p)$. (If so, then the graph G itself is of treewidth $\leq f(c, p)$).
—IF all components have treewidth $\leq f(c, p)$ THEN RETURN(*satuckeck*$'_P(G)$).
—IF there exists a component K having treewidth $> f(c, p)$ THEN RETURN(*true*).

The last step is justified as follows: If K has treewidth $> f(c, p)$, then K contains by Lemma 5.12 (a) (c', p)-racket where c' is a multiple of c. Given that R is a positive (c, p)-racket, by Lemma 5.13, $K \in$ SATU(P). Moreover, since either $D^p =$ *endpt*(R) (in case *length*$(\omega) \neq 0$) or D^p is a vertex of T (in case *length*$(\omega) = 0$), it holds by Lemma 5.13 that there is a legal coloring *col* of K with respect to P and a vertex w_0 of K such that $col(w_0) = D^p$. Let *wit* be a witness function for *col*. We extend *col* to a coloring col^* for the entire graph G by defining: $\forall v \in G - K$, $col^*(v) = D^-$. We extend *wit* to a function wit^* on G by defining: $\forall v \in G - K$, $wit^*(v) = w_0$. It is obvious that col^* is a legal coloring for G with respect to P via the witness function wit^*. Thus, $G \in \text{SATU}(P)$.

Case 3. All cycles of P are negative.

By Lemma 5.3, $G \in \text{SATU}(P)$ iff $G^c \in \text{SATU}(P^\bullet)$. Note that $P^\bullet$ contains only positive cycles. Checking whether $G^c \in \text{SATU}(P^\bullet)$ can thus be done in accordance with Case 2.

The three cases exhaustively cover all possibilities; thus the algorithm effectively decides whether $G \in \text{SATU}(P)$. Moreover, whether $G \in \text{SATU}(P)$ can be determined in quadratic time. In fact, the computationally relevant actions of the algorithm described in this proof are:

—Computing the complement G^c of G (this may require quadratic time).
—Determining the connected components of G or G^c (this can be done in linear time by Tarjan's algorithm [Tarjan 1972]).
—Checking for each component, whether its treewidth is smaller than a constant (this is solvable in linear time by Proposition 5.6).
—Performing a constant number of further linear time actions on single components, such as the procedure calls $satucheck_P(G)$ or $satucheck'_P(G)$.

In summary, all this requires no more than quadratic time (in the size of the input G). □

We conclude this section with a remark.

Remark 5.1. We currently do not know whether the quadratic upper bound stated in the proof of Theorem 5.14 can be improved. Note also that for each P, SATU(P) is probably not a PTIME-complete set. The set SATU(P) can be seen to be in the complexity class LOGCFL, the class of all languages that are logspace-reducible to a context-free language. This class is contained in NC_2, and consists of highly parallelizable problems. LOGCFL is currently the best known upper bound for SATU(P). This is due to the check for bounded treewidth, which is in LOGCFL (cf. Wanke [1994]) but not known to be in NL.

5.2. A Slight Generalization: When Mixed Cycles Do not Matter.

In this section, we slightly generalize Theorem 5.14. We deal with the case in which a pattern graph P may have mixed cycles, but a graph G cannot use mixed cycles of P for saturation.

Definition 5.15. Let $P = (\text{Colors}, \text{Arcs})$ be a pattern graph and $G = (V, E)$ a self-loop free undirected graph. Let $col : V \longrightarrow \text{Colors}$ be a legal coloring of G with respect to P with associated witness function *wit*. Then the witness graph $G[wit]$ is the directed edge-labeled graph (V, E_{wit}, ℓ), where $E_{wit} = \{(x, wit(x)) \mid x \in V\}$ and where ℓ is an edge labeling function such that $\ell(x, wit(x)) = \oplus$ if $\{x, wit(x)\} \in E$ and $\ell(x, wit(x)) = \ominus$ otherwise.

A mixed cycle of $G[wit]$ is a cycle having mixed edge labels. Pure, positive, and negative cycles of $G[wit]$ are defined in the obvious way.

Definition 5.16. Let $P = (\text{Colors}, \text{Arcs})$ be a pattern graph and $G = (V, E)$ an undirected graph. We say that G is *impurely saturated by P* if $G \in \text{SATU}(P)$ and there exists a legal coloring $col : V \longrightarrow$ Colors of G with respect to P with associated witness function wit such that $G[wit]$ contains at least one mixed cycle.

The set of all self-loop free undirected graphs G which are impurely saturated by a pattern graph P is denoted by $\mathit{ISATU}(P)$.

Obviously, if P is a pure pattern graph, then $\mathit{ISATU}(P) = \emptyset$. We now establish the following result.

THEOREM 5.17. *For each pattern graph P, the following problem is decidable in polynomial time. Given a self-loop free undirected graph G such that $G \notin \mathit{ISATU}(P)$, decide whether $G \in \mathit{SATU}(P)$.*

PROOF. Let P be a fixed pattern graph. If P is acyclic, then, by analogy to the proof of Corollary 5.8, $\text{SATU}(P)$ is the empty graph. If P is cyclic but contains only mixed cycles, then, in analogy to Corollary 5.8, it is easy to see that $\text{SATU}(P) - \text{ISATU}(P)$ is again the empty graph. Thus, if P is acyclic or contains only mixed cycles, our problem is trivially decidable in polynomial time. For the rest of the proof, we thus assume that P contains at least one pure cycle.

A vertex (= color) C of P such that there is no directed path from C to any pure cycle in P is called a *superfluous* color. Assume that for some graph $G \in \text{SATU}(P)$ there is a legal coloring col that uses a superfluous color C, that is, there is a vertex v of G such that $col(v) = C$ and C is superfluous in P. Let wit be a witness function for this coloring. Then the subgraph of $G[wit]$ induced by the vertices $\{v, wit(v), wit(wit(v)), \ldots, wit^i(v), \ldots\}$ must contain a cycle Z, given that it is finite. If Z was a pure cycle, then $col(Z)$ would induce a pure cycle in P which is reachable by a directed path from C; contradiction. Thus Z is a mixed cycle and hence $G \in \mathit{ISATU}(P)$. It follows that whenever $G \in \text{SATU}(P) - \mathit{ISATU}(P)$, then no legal coloring of G with respect to P uses a superfluous color. Thus, for the purpose of our theorem we can assume without loss of generality that P does not contain any superfluous color (otherwise, such a color could be deleted without harm from P).

Again, we distinguish three cases.

Case 1. P has positive and negative cycles.

This case is solved in exactly the same way as Case 1 in the proof of Theorem 5.14.

Case 2. All pure cycles of P are positive.

We consider two subcases:

(a) *P is positive*. By Lemma 5.2, $G \in \text{SATU}(P)$ iff $K \in \text{SATU}(P)$ for each connected component K of G. Again, as in the proof of Theorem 5.14, this check is done for each K in accordance with Case 1(a) (of Theorem 5.14).

(b) *P has some edges labeled $\ominus$*. Let $e = (D^i, D^j)$ be such an edge. Given that P contains no superfluous vertices, there exists a simple path σ (of possible length zero) from D^j to a positive cycle T in P. Continue as in Case 2(b) in the proof of Theorem 5.14.

Case 3. All pure cycles of P are negative.

By Lemma 5.3, $G \in \text{SATU}(P)$ iff $G^c \in \text{SATU}(P^\bullet)$. Note that all pure cycles of $P^\bullet$ are positive. Checking whether $G^c \in \text{SATU}(P^\bullet)$ can thus be done in accordance with Case 2.

The three cases exhaustively cover all possibilities; thus the algorithm effectively decides whether $G \in \text{SATU}(P)$. Moreover, determining whether $G \in \text{SATU}(P)$ is feasible in quadratic time (in the size of the input G). $\square$

6. *The Saturation Problem for General Pattern Graphs*

In this section, we show that $\text{SATU}(P)$ is in PTIME, for *each* pattern graph P. The proof is similar in spirit to the proof for pure pattern graphs but the details are a bit more complicated. Similar to Section 5, cycles in the graph G will play an important role. But in this section we have to focus on the existence of *mixed cycles*, that is, closed "paths" consisting of edges and nonedges. Again, we give an algorithm which distinguishes between two main cases. In the first case, the input graph G has a special structure which is a slight generalization of constant-bounded tree-width. We show that on such graphs the $\text{SATU}(P)$ test can be easily reduced to the case of graphs of constant-bounded tree-width. On the other hand, we show that if a graph does *not* have this special structure, then (1) a legal coloring of a mixed cycle can always be extended to a legal coloring of the whole graph and (2) a graph with a legally colorable mixed cycle always contains a legally colorable mixed cycle of constant-bounded size that depends only on P. Hence, in the second case the $\text{SATU}(P)$ test essentially boils down to checking for the existence of a legally colorable mixed cycle of constant-bounded size and, if no such cycle exists, calling the procedure of Section 5.

Notation. Before we define the notion of special graphs that is needed for our purposes, we introduce some more notation. Let P be a pattern graph. Edges of P that are labelled with $\ominus$ are called $\ominus$-*edges* and edges labelled with $\oplus$ are called $\oplus$-*edges*. It will be convenient to view an input graph $G = (V, E)$ as a complete, undirected, self-loop free graph where the edges carry labels $\ominus$ or $\oplus$. The $\oplus$-labelled edges are those from E and the $\ominus$-labelled edges are those that are not in E. In this view, we call G $\ominus$-$\oplus$-*labelled* and refer to the set of $\ominus$-labelled edges and the set of $\oplus$-labelled edges as $E^\ominus$ and $E^\oplus$, respectively. To emphasize the symmetry of the two kinds of labels and to support intuition, we will often call $\ominus$-labelled edges *negative* edges and $\oplus$-labelled edges *positive* edges. In figures, $\oplus$-edges will be depicted by solid lines, $\ominus$-edges by dashed lines. To summarize, we use the following equivalent notations.

$$\begin{array}{rcccccc} \text{edge in } E & \equiv & \text{label } \oplus & \equiv & \text{——} & \equiv & \text{positive} \\ \text{edge not in } E & \equiv & \text{label } \ominus & \equiv & - - - & \equiv & \text{negative} \end{array}$$

We write $\text{lab}(v, w)$ to refer to the label of the edge between v and w. Hence $\text{lab}(v, w) \in \{\ominus, \oplus\}$, for each pair of vertices $v \neq w$. To indicate that $\text{lab}(v, w) = \oplus$ we sometimes also write $\text{positive}(v, w)$. If $\text{positive}(v, w)$ for all vertices w of some set A, we also write $\text{positive}(v, A)$. If all vertices from a set A are connected by a positive edge with all vertices from B, then we write $\text{positive}(A, B)$. If all edges between vertices of A are positive, we call A a positive clique. For a set A of vertices, we call the graph (in the standard sense) which consists of all vertices of A and the edges of $E^\oplus$ between vertices of A the *positive graph of* A. Analogous notations are used with negative in place of positive. We write uni(A, B) to indicate that all edges between a set A and a set B have the same label, no matter which.

A sequence $v_1, \ldots, v_l$ of (pairwise different) vertices of G determines a *path* p of G which consists of all edges $\{v_i, v_{i+1}\}$, $i < l$. We write pat(p) for the $\ominus$-$\oplus$-string lab$(v_0, v_1) \cdots$ lab(v_{l-1}, v_l). By adding the edge $\{v_l, v_1\}$, the path p also determines a *cycle* C of G. To simplify notation we refer to paths and cycles by only listing their constituting vertices. We write first(p) and last(p) to refer to v_1 and v_l, respectively. We call a cycle $C = v_1, \ldots, v_l$ *self-saturating* with respect to a pattern graph P, if there is a coloring col of the vertices of C, that has the function f, defined by $f(v_i) = v_{i+1}$ as a witness function. Here, as in the following, arithmetic on indices is modulo l, hence $v_{l+1} = v_1$. A cycle is called *mixed* if it contains positive and negative edges.

Now we turn to the definition of the generalization of constant-bounded tree-width. For integers k and t, we call a $\ominus$-$\oplus$-graph G (k, t)-*special* if its set of vertices can be partitioned into sets A and $A_1, \ldots, A_k$ such that the following conditions hold.

—The positive graph of A or the negative graph of A has tree-width at most t.
—For each $i \leq k$, A_i is either a negative clique or a positive clique (or might be empty).
—For each i and each vertex v, uni(v, A_i).

The Algorithm. Next, we give a high-level description of the algorithm for the general SATU(P) problem. Its input consists of a pattern graph P, a $\ominus$-$\oplus$-graph G and constants c, k and t. The algorithm only decides SATU(P) correctly, if c, k and t are chosen appropriately with respect to P. For the complexity analysis, P, c, k and t will be considered as fixed.

Algorithm 6.1

```
(1) Test whether $G$ is $(k, t)$-special using the algorithm of Lemma 6.3 below.
(2) If $G$ is $(k, t)$-special, then test whether $G \in$ SATU$(P)$ using the algorithm of Lemma 6.4 below.
(3) If $G$ is not $(k, t)$-special
    (a) Test whether $G$ has a mixed cycle $C$ of size $\leq c$ that has a legal coloring with respect to $P$.
    (b) If this is the case, accept.
    (c) Otherwise, call the algorithm that was given in the proof of Theorem 5.17 above.
```

Now we can state the main result of this section which immediately implies Theorem 4.8.

THEOREM 6.2. *For each pattern graph P, there are constants k, c and t such that Algorithm* 6.1 *decides SATU(P) correctly. Furthermore, for each choice of P, k, c and t, Algorithm* 6.1 *runs in polynomial time.*

Theorem 6.2 will be proven in the remainder of the section. The proof consists of a series of lemmas and is spread over three subsections.

—In Section 6.1, we show that, for fixed k and t, whether a graph G is (k, t)-special can be checked in polynomial time, and, for fixed k, t and P, whether a (k, t)-special graph is in SATU(P) is also decidable in polynomial time. Therefore, for each fixed choice of P, k and t, the first two steps of the algorithm can be performed in polynomial time.

—In Section 6.2, it is shown that for each pattern graph P there is a constant c such that whenever a graph G has a self-saturating mixed cycle with respect to P it already has one of size at most c. Therefore, if Step 3(a) of the algorithm does not find a self-saturating cycle (for suitable choice of c) there is no self-saturating cycle at all in G.

—Finally, in Section 6.3, we show that if a graph that is not (k, t)-special has a self-saturating mixed cycle with respect to P then it is already in SATU(P). This justifies acceptance in Step 3(b) of the algorithm. On the other hand, if 3(a) fails then G can not be saturated impurely, hence the correctness of Step 3(c) follows from Theorem 5.17.

6.1. The Saturation Problem for Special Graphs. We show first, that, for each fixed choice of P, c, k and t, the first two steps of Algorithm 6.1 can be performed in polynomial time. We call two vertices v and v' of a $\ominus$-$\oplus$-graph $G = (V, E^{\ominus}, E^{\oplus})$ *equivalent* if, for all vertices w in $V - \{v, v'\}$, $\text{lab}(v, w) = \text{lab}(v', w)$.

Lemma 6.3. *For each k and t it can be checked in polynomial time whether a $\ominus$-$\oplus$-graph G is (k, t)-special. Furthermore, a corresponding partition can be constructed in polynomial time.*

Proof. The algorithm works as follows.

```
for each $k' \le k$, each tuple $(v_1, \dots, v_{k'})$ of pairwise distinct vertices of $G$
          and each $j \in \{0, \dots, k'\}$
    $B := V - \{v_1, \dots, v_{k'}\}$
    for $i = 1$ to $j$ do
          $A_i := \{v_i\} \cup$ all vertices from $B$ that are equivalent to $v_i$ and
               connected to $v_i$ via a positive edge
          $B := B - A_i$
    end
    for $i = j + 1$ to $k'$ do
          $A_i := \{v_i\} \cup$ all vertices from $B$ that are equivalent to $v_i$ and
               connected to $v_i$ via a negative edge
          $B := B - A_i$
    end
    if the negative graph on $B$ or the positive graph on $B$ has tree-width $\le t$,
          then accept
end
Reject
```

For fixed k, this algorithm runs in polynomial time. It is straightforward to verify that it recognizes (k, t)-special graphs correctly. In particular, if $u \neq u'$ are in A_i, distinct from v_i then the edge between u and u' has the same polarity as the edges between u and v_i and between u' and v_i, respectively. Therefore, each A_i is a (positive or negative) clique. $\square$

Lemma 6.4. *There is an algorithm which tests, for each fixed choice of k, t and a pattern graph P, in polynomial time whether a (k, t)-special input graph G is in SATU(P).*

Proof. We describe how a (k, t)-special graph G can be transformed in polynomial time into a graph G' of tree-width t' (only depending on k, t and P but not on G) that is equivalent to G with respect to SATU(P). As $G' \in$ SATU(P) can be tested in polynomial time in accordance with Corollary 5.5, this implies the statement of the Lemma. Let $A, A_1, \ldots, A_{k'}$ be the partition of G as computed by the algorithm of Lemma 6.3. Let l be the number of vertices in P. For each $i \in \{1, \ldots, k'\}$, let $A'_i := A_i$, if $|A_i| \leq 2l$, or let A'_i consist of arbitrarily chosen $2l$ vertices from A_i, otherwise. The computation of the A'_i clearly works in polynomial time. Let G' be the subgraph of G which is induced by A and $A'_1, \ldots, A'_{k'}$. As each set A'_i has at most $2l$ elements, the tree-width of the positive graph of G' or the negative graph of G' is at most $t' = t + 2k'l$.

It remains to show that $G' \in \text{SATU}(P)$ if and only if $G \in \text{SATU}(P)$.

Assume $G' \in \text{SATU}(P)$. Then, by definition, there is a legal coloring of G' with respect to P. Let $i \geq 1$, let v_i be some node in A'_i and let p be the color of v_i in the coloring of G'. Then all vertices v from $A_i - A'_i$ can be colored by p as they are related to all vertices of G' exactly as v_i. In particular, we can set $wit(v) = wit(v_i)$.

For the opposite direction, assume that $G \in \text{SATU}(P)$. Then there is a legal coloring of G with respect to P. Let, for each $i \geq 1$, F_i be the set of colors which occur in A_i. The vertices of A'_i are colored as in G, if $A'_i = A_i$ or such that each color from F_i is used at least twice. Assume $v \in G'$ and v is saturated in G by a vertex $u \in A_i$, for some i. If $v \notin A_i$, then v can be saturated in G' by a vertex of A'_i with the same color as u. Now assume $v \in A_i$. If $A'_i = A_i$ then v can be saturated in G' by u. Otherwise, there exists at least one vertex different from v in A'_i that has the same color as u and saturates v. This completes the proof of the lemma. □

Remark 6.1. By a similar proof it can be shown that, for each k and t, model-checking of MSO-formulas for (k, t)-special graphs is in PTIME.

6.2. EXISTENCE OF SMALL SELF-SATURATING CYCLES. The next lemma shows that whenever a graph G has a self-saturating mixed cycle with respect to a pattern graph P then it also has a *constant-bounded* self-saturating mixed cycle.

LEMMA 6.5. *Let P be a pattern graph. There is a constant d such that whenever a $\ominus$-$\oplus$-graph G has a self-saturating mixed cycle with respect to P then it has such a cycle with at most d vertices.*

PROOF. Let l be the number of vertices of P. Let $m := l^4$ and $d := (lm)^2(2l)^{lm+1} + 2$. Let $C = v_0, \ldots, v_{n-1}$ be a self-saturating mixed cycle of G with respect to P of length $n > d$ and let col be a corresponding coloring function. Without loss of generality, we can assume that the edges between v_{n-2} and v_{n-1} on one hand and between v_{n-1} and v_0 on the other hand have different labels.

We will show that there is a smaller self-saturating mixed cycle C' which actually only consists of vertices of C. We construct C' either by skipping one subpath of $v_1, \ldots, v_{n-1}$ in C or by combining some vertices of C in an entirely new way, resulting in a small mixed cycle. In the former case, the edges between v_{n-2} and v_{n-1} and between v_{n-1} and v_0, respectively, guarantee that C' is mixed.

We distinguish two cases.

Case 1. C has a subpath $v_k, \ldots, v_{k+m}$ of length m in which all edges have the same label.

Without loss of generality, we may assume this label is $\oplus$. For notational convenience, we also assume $k = 0$. The general case is completely analogous.

We consider the l^3 paths $P_i = v_{il}, \ldots, v_{(i+1)l}$, for all $i \in \{0, \ldots, l^3 - 1\}$. As there are only l colors for each P_i, there are $j < j' \leq l$ such that v_{il+j} and $v_{il+j'}$ have the same color. For each i, we choose such j and j' and let $J_i := \{v_{il+j+1}, \ldots, v_{il+j'}\}$

and $\delta_i := j' - j$. There are at least l^2 subpaths P_i with the same value of δ_i. Let δ denote this value.

If there exists a path Q from v_0 to v_m such that

(1) the length of Q is $m - j\delta$, for some $j \leq l^2$,

(2) all intermediate vertices of Q are from $\{v_1, \dots, v_{m-1}\}$, and

(3) all edges of Q carry the label $\oplus$

then we can construct a self-saturating mixed cycle of length $n - j\delta$ by simply replacing the path $v_0, \dots, v_m$ in C by Q and coloring the vertices of Q according to $v_0, \dots, v_m$ but skipping all vertices of sets J_i, $i < j$.

A first immediate consequence is that we are done if there exists an edge with label $\oplus$ between vertices v_i and $v_{i+\delta+1}$, for some $i < m - \delta - 1$. Hence, we assume in the following that all such edges have label $\ominus$.

As C is a mixed cycle there is a closed mixed (directed) path in P. A straightforward argument shows that there must be such a path $w_0, \dots, w_{t-1}$ of some length $t \leq l + 1$. Let c_1 be the number of $\ominus$-edges on this path and c_2 the number of $\oplus$-edges. Of course, $c_1 + c_2 = t$. Without loss of generality, we may assume that the w_i are numbered such that the edge in P from w_{t-1} and w_0 has label $\ominus$. Let $c' := (c_1 - 1)(\delta - 1) + c_2$. Note that $c' \leq l^2$. Consider the edges between $v_{ic'}$ and $v_{ic'+c'}$, for $i = 0, \dots, \delta - 1$. If all these edges have label $\oplus$, there is a path Q of length $m - (c' - 1)\delta$ from v_0 to v_m with the above properties (1)–(3).

Hence, we may assume that there is a $i \leq \delta - 1$ such that the label between $v_{ic'}$ and $v_{ic'+c'}$ is $\ominus$. We can construct a mixed self-saturating cycle $v'_0, \dots, v'_{m-1}$ as follows.

Let $v'_0 := v_{ic'}$. Now let v'_j be already chosen, for a $j < t - 1$, as some $v_{i'}$. We set $v_{j+1} := v_{i'+1}$ if the label between w_j and w_{j+1} has label $\oplus$, $v'_{j+1} := v_{i'+\delta+1}$, otherwise. By the definition of c_1, c_2 and c', we get $v'_{t-1} = v_{ic'+c'}$. As the label between $v_{ic'}$ and $v_{ic'+c'}$ is $\ominus$, the cycle $C' = v'_0, \dots, v'_{t-1}$ is self-saturating according to $w_0, \dots, w_{t-1}$.

This completes the proof of Case 1.

Case 2. Every subpath of C of length m contains $\ominus$-edges and $\oplus$-edges.

Claim. There exist $k \leq l$ and lm edge-disjoint subpaths $p_j = v_{i_j}, \dots, v_{i_j+km}$, $j \in \{1, \dots, lm\}$, such that

—all subpaths are colored identically, that is, for all $j, j' \leq lm$ and $s \leq km$, $\text{col}(v_{i_j+s}) = \text{col}(v_{i_{j'}+s})$,

—all subpaths have the same pattern, that is, for all $j, j' \leq lm$, $\text{pat}(v_{i_j}, \dots, v_{i_j+km}) = \text{pat}(v_{i_{j'}}, \dots, v_{i_{j'}+km})$, and

—for each $j \leq lm$, $\text{col}(v_{i_j}) = \text{col}(v_{i_j+km})$.

To prove this claim, consider the $p := lm(2l)^{lm+1}$ paths $q_j = v_{jlm}, \dots, v_{jlm+lm}$, $j = 0, \dots, p - 1$ of length lm of C. As each vertex v_i carries one of l colors and each edge is positive or negative, there are only $(2l)^{lm+1}$ different possibilities of vertex colorings or patterns of a path of length lm. Hence, there exist lm paths $q_{j_1}, \dots, q_{j_{lm}}$ with the same coloring and pattern. Again, as there are only l colors, at least two of the $l + 1$ vertices $v_{j_ilm}, v_{j_ilm+m}, \dots, v_{j_ilm+lm}$ must have the same color, for each $i \leq lm$. As the paths are colored identically, we can select in each path

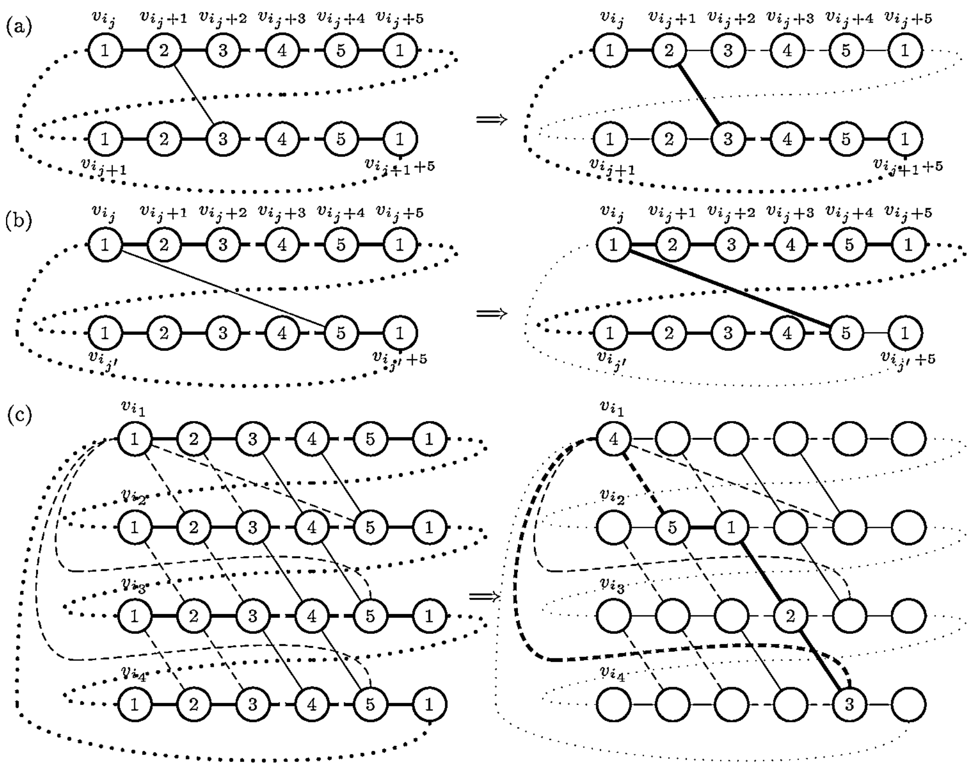

FIG. 8. Construction of C' (right hand side) from C (left hand side). $km = 5$. Colors are 1, 2, 3, 4, 5. Thick lines indicate the self-saturating cycle. Solid lines carry $\oplus$, dashed lines $\ominus$, dotted lines represent subpaths. Some vertex names are omitted to improve readability.

two such vertices at the same relative positions. By taking the subpaths between (and including) these two vertices, we get paths p_i with the properties stated in the claim. Recall that each p_i contains $\ominus$-edges and $\oplus$-edges.

Without loss of generality, $i_1 < i_2 < \cdots < i_{lm}$.

If for some $j < lm$ and some $s < km$ it holds $\text{lab}(v_{i_j+s}, v_{i_{j+1}+s+1}) = \text{lab}(v_{i_j+s}, v_{i_j+s+1})$, then we get a smaller self-saturating cycle C' by eliminating the vertices $v_{i_j+s+1}, \ldots, v_{i_{j+1}+s}$ from C (cf. Figure 8(a)).

On the other hand, if for some $j \leq j' \leq l$ it holds $\text{lab}(v_{i_j}, v_{i_{j'}+km-1}) = \text{lab}(v_{i_{j'}+km-1}, v_{i_{j'}+km})$, then $v_{i_j}, \ldots, v_{i_{j'}+km-1}$ defines a smaller self-saturating cycle (cf. Figure 8(b)).

Hence, we can assume that

(a) for each $j < lm$ and for all $i < km$,

$$\text{lab}(v_{i_j+i}, v_{i_{j+1}+i+1}) \neq \text{lab}(v_{i_j+i}, v_{i_j+i+1}),$$

and

(b) for each $j \leq j' \leq lm$,

$$\text{lab}(v_{i_j}, v_{i_{j'}+km-1}) \neq \text{lab}(v_{i_{j'}+km-1}, v_{i_{j'}+km}).$$

We show next how this can be used to construct a self-saturating mixed cycle $C' = v'_0, \ldots, v'_{km-1}$ consisting only of vertices from $\{v_{i_1}, \ldots, v_{i_{tm}+km}\}$.

To this end, let $s \in \{1, \dots, km\}$ be chosen such that $\mathrm{lab}(v_{i_1+s-1}, v_{i_1+s}) \neq \mathrm{lab}(v_{i_1+km-1}, v_{i_1+km})$. Such an s exists because, by the assumption of Case 2, no m consecutive edges in C carry the same label. As $\mathrm{lab}(v_{i_1+km-1}, v_{i_1+km}) = \mathrm{lab}(v_{i_{j'}+km-1}, v_{i_{j'}+km})$, for each $j' \leq lm$, we can conclude from (b) that, for each $j' < lm$, $\mathrm{lab}(v_{i_1+s-1}, v_{i_1+s}) = \mathrm{lab}(v_{i_1}, v_{i_{j'}+km-1})$.

To simplify notation, we set $w_i := v_{i_1+i}$, for each $i \in \{0, \dots, km\}$. Our goal is to construct $C' = v'_0, \dots, v'_{km-1}$ such that the following holds.

(i) $\mathrm{pat}(v'_0, \dots, v'_{km-1-s}) = \mathrm{pat}(w_s, \dots, w_{km-1})$;
(ii) $\mathrm{lab}(v'_{km-1-s}, v'_{km-s}) = \mathrm{lab}(w_{km-1}, w_{km})$;
(iii) $\mathrm{pat}(v'_{km-s}, \dots, v'_{km-1}) = \mathrm{pat}(w_0, \dots, w_{s-1})$;
(iv) $\mathrm{lab}(v'_{km-1}, v'_0) = \mathrm{lab}(w_{s-1}, w_s)$

If we color the nodes v'_i in accordance with this correspondence, we get a self-saturating coloring of the cycle C', as, for each $j < km$, w_j is saturated by w_{j+1}. Note that the coloring of v'_{km-s} is unambigous as we have $\mathrm{col}(w_0) = \mathrm{col}(w_{km})$.

The vertices v'_i, $i = 0, \dots, km-1-s$, are inductively chosen as follows: Each v'_i will be a vertex $v_{i_j} + i$, for some j. Let $v'_0 = v_{i_1}$. Now assume that $v'_i = v_{i_j+i}$ is already selected and $i < km-1$. Recall that $\mathrm{lab}(v_{i_j+i}, v_{i_j+i+1}) \neq \mathrm{lab}(v_{i_j+i}, v_{i_{j+1}+i+1})$. Hence, we can ensure $\mathrm{lab}(v'_i, v'_{i+1}) = \mathrm{lab}(w_{s+i}, w_{s+i+1})$ by either choosing v'_{i+1} as v_{i_j+i+1} or as $v_{i_{j+1}+i+1}$.

In an analogous fashion, we choose the vertices $v'_{km-s}, \dots, v'_{km-1}$, such that (i)–(iii) hold. As stated above, the choice of s guarantees $\mathrm{lab}(v_{i_1+s-1}, v_{i_1+s}) = \mathrm{lab}(v_{i_1}, v_{i_j+km-1})$, for each j, hence we can also conclude (iv).

The construction is exemplified in Figure 8(c). This concludes the proof of the second case. □

6.3. Extending a Legal Coloring of a Cycle.

This section is devoted to the proof that, for each pattern graph P there exist k and t, such that, for each $\ominus$-$\oplus$-graph G which is *not* (k, t)-special but has a self-saturating cycle it holds that $G \in \mathrm{SATU}(P)$. This is actually the most complicated part of the proof of Theorem 6.2.

If $C = w_1, \dots, w_l$ is a self-saturating mixed cycle of a graph G with a fixed coloring, we say that *C saturates a vertex $v \notin C$ directly,* if $\mathrm{lab}(v, w_i) = \mathrm{lab}(w_{i-1}, w_i)$, for some i, that is, v is saturated by some w_i. Here, as always in the following, arithmetic within the subscript of the vertices w_i is modulo l; hence, all indices are from $\{0, \dots, l-1\}$.

In principle, a graph might be saturated by using several self-saturating (mixed and pure) cycles and extending their colorings via paths of various lengths. However, we show in the following that if there is a saturating coloring involving a mixed cycle at all then there exists a coloring with one self-saturating cycle C which saturates almost all other vertices directly. Only for a constant number of vertices, depending on the pattern graph, there might be the need for a saturating path. But the lengths of these paths are bounded by a constant, too.

We call a vertex $v \notin C$ *bad with respect to C* if it is not directly saturated by C. The set of all such vertices is denoted $\mathrm{Bad}(C)$.

We proceed in two main steps.

First, we show that there is a constant c depending on P, k and t such that for each self-saturating mixed cycle C that has more than c bad vertices there exists

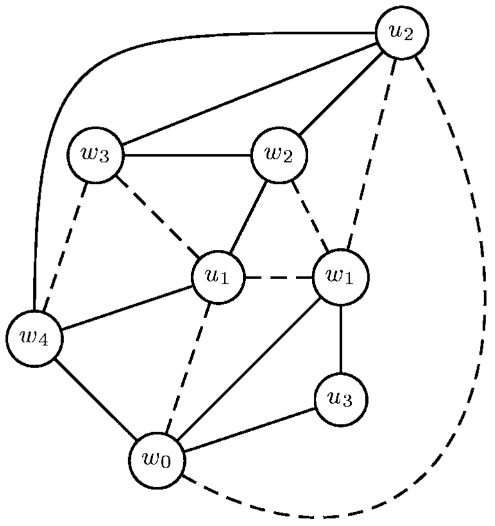

FIG. 9. Vertex u_1 is bad with respect to the cycle $C = w_0, w_1, w_2, w_3, w_4$. Vertex u_2 is neutral as it can only be saturated by w_3. Vertex u_3 is good as it is saturated by the $\oplus$-assistant w_1 and the $\ominus$-assistent w_0. Only the relevant edges are shown.

a self-saturating mixed cycle C' such that all vertices from Bad(C) are directly saturated by C' or the coloring of C can be extended to a saturating coloring of G. By slightly modifying this construction, we can get a self-saturating cycle D such that actually $|\text{Bad}(D)| < |\text{Bad}(C)|$ (Lemma 6.8). By repeatedly applying Lemma 6.8, we arrive at a self-saturating cycle C for which $|\text{Bad}(C)| < c$.

Second, Lemma 6.10 shows that this constant number of remaining vertices can be saturated along suitable paths in G.

Let, in the following, P be a fixed pattern graph, let k and t be chosen large enough, let G be a $\ominus$-$\oplus$-graph that is not (k, t)-special and let $C = w_1, \dots, w_l$ be a mixed cycle that is self-saturated with respect to P for some coloring col.

We continue by defining a few additional notions and establishing some notation that will be used in the sequel. If, for some $i \leq l$, lab$(w_{i-1}, w_i) = \oplus$ (respectively, $\ominus$), we say that w_i *saturates positively* (respectively, *negatively*). We say that w_i is a $\oplus$-*assistant* ($\ominus$-*assistant*) if lab(w_{i-1}, w_{i-2}) is $\oplus$ (respectively, $\ominus$).

For a vertex v that is not bad with respect to C, there might exist several vertices w_i that can saturate v. If v can be saturated directly by a 0-assistant and a 1-assistant, then we can choose whether v itself can saturate other vertices via positive or negative edges. As this gives us some additional flexibility, we call such vertices *good with respect to* C.

Vertices that are neither good nor bad are called *neutral* with respect to C. We write Good(C) and Neu(C) for the sets of vertices that are good and neutral, respectively. Figure 9 exemplifies the definition of Good(C), Bad(C), and Neu(C).

As stated above, good vertices give us an additional flexibility in extending colorings. On the other hand, if we know that a vertex v is not good and can saturate only via, say, positive edges, we know it is not directly saturated by any $\ominus$-assistant. Hence, for each $\ominus$-assistant w_i, we can infer the polarity of the edge between v and w_i. This knowledge will be useful in the construction of a better cycle.

The following lemma guarantees that the number of good vertices is small compared to the number of bad vertices unless the coloring of C can be extended to a saturating coloring.

LEMMA 6.6. *If* $|Good(C)| > \log(|Bad(C)|)$, *then the coloring of* C *can be extended to a legal coloring of* G.

PROOF. We color the vertices of $\text{Neu}(C)$ such that they are saturated directly by C. Then, we ensure that all vertices from $\text{Bad}(C)$ are saturated by vertices from $\text{Good}(C)$. We proceed inductively as follows: Initialize $A := \text{Good}(C)$ and $B := \text{Bad}(C)$. We repeat the following until A is empty. We pick a vertex v from A. If at least half of the vertices of B are connected to v by a positive edge, then we color v such that it saturates positively. Otherwise, we color v such that it saturates negatively. In either case, v saturates at least half of the vertices in B. We delete v from A and delete the vertices saturated by v from B. Clearly, the size of B after each step is at most half the size as before. Hence, after $\log(|\text{Bad}(C)|) < |\text{Good}(C)|$ steps all vertices are saturated. $\square$

In a similar fashion, we can deal with the particular case where $\ominus$-edges and $\oplus$-edges occur alternatingly in C. Before we formally prove this statement, we introduce some more notation that will be used in the subsequent proofs. Recall that if $p = v_0, \ldots, v_m$ is a path, $\text{pat}(p)$ denotes the $\ominus$-$\oplus$-string $\text{lab}(v_0, v_1) \cdots \text{lab}(v_{m-1}, v_m)$. For a regular expression e over $\{\ominus, \oplus\}$, we say that p *matches* e if the string $\text{pat}(p)$ matches e in the usual sense. Analogously, if we view $C = v_0, \ldots, v_m$ as a cycle, we define $\text{pat}(C)$ as $\text{pat}(v_0, \ldots, v_m, v_0)$.

LEMMA 6.7. *If* $pat(C)$ *matches* $(\ominus\oplus)^*$ *or* $(\oplus\ominus)^*$, *then* $G \in SATU(P)$.

PROOF. If $|\text{Good}(C)| \geq |\text{Bad}(C)|$, then the statement follows from Lemma 6.6. Otherwise, we take the reversal of C as C'. It is easy to see that $\text{Neu}(C) \subseteq \text{Neu}(C') \cup \text{Good}(C')$ and $\text{Bad}(C) \subseteq \text{Good}(C')$, hence $|\text{Good}(C')| \geq |\text{Bad}(C')|$ and the statement follows, again by Lemma 6.6. $\square$

6.3.1. *Reducing the Number of Bad Vertices*

LEMMA 6.8. *For each* $l > 0$, *there are* c, k, t *such that the following holds. If* G *is not* (k, t)*-special,* C *is a self-saturating mixed cycle of* G *of size at most* l *with a coloring col and* $|Bad(C)| > c$, *then either col can be extended to a legal coloring of* G *or there is a mixed self-saturating cycle* D *in* G *with* $pat(D) = pat(C)$ *and* $|Bad(D)| < |Bad(C)|$.

PROOF. Let c be large enough such that each $\ominus$-$\oplus$-graph with c vertices contains a negative or a positive clique of size l; such a c is guaranteed to exist by Ramsey's classical theorem. Let t be large enough such that each graph of tree-width larger than t contains a cycle of length at least $2l$ and let $k = 3l$.

Let G be a graph which is not (k, t)-special and let $C = w_1, \ldots, w_l$ be a cycle self-saturated by a coloring col and witness function $f(w_i) = w_{i+1}$.

Although the vertices in $\text{Bad}(C)$ are the troublesome vertices, they will play an important role in the construction of D. The reason for this is that a lot of information can be deduced from knowing that a vertex is bad; specifically, if a vertex v is in $\text{Bad}(C)$, then we can derive the labels of all edges between v and C.

If pat(C) matches $(\ominus\oplus)^*$ or $(\oplus\ominus)^*$, then we are done by Lemma 6.7. Therefore, we assume in the following that this is not the case. Because of Lemma 6.6, we can also assume $|\text{Good}(C)| \leq \log(|\text{Bad}(C)|)$.

The proof consists basically of two parts. We first describe the construction of a cycle C' which almost has the desired properties. In the second part, it might be necessary to modify C' slightly to obtain D. The construction maintains pat(C) = pat(C') = pat(D).

To be more precise, the cycle C' will saturate all vertices in Bad(C) directly, that is, we get $\text{Bad}(C) \subseteq \text{Good}(C') \cup \text{Neu}(C')$. It might happen that some vertices in Good(C) are no longer saturated by C' but this does not hurt too much as $|\text{Good}(C)| \leq \log(|\text{Bad}(C)|)$. Nevertheless, there are some cases, depending on pat(C), in which it can not be guaranteed that the vertices of Neu(C) are saturated by C'. These are the cases where we need to modify C' to get the desired cycle D. Otherwise we can simply set $D := C'$. The modification will be conservative in the sense that the vertices from Bad(C) are guaranteed to be in $\text{Good}(D) \cup \text{Neu}(D)$.

6.3.1.1. CONSTRUCTION OF C'. By the choice of c, Bad(C) contains a positive or negative clique K of size l. As pat(C) is neither of the form $(\ominus\oplus)^*$ nor of the form $(\oplus\ominus)^*$, a simple induction shows that we can partition C into subpaths $p_1, \ldots, p_n$ such that each subpath conforms to exactly one of the regular expressions

(1) $\ominus\ominus^+\oplus^+$,
(2) $(\ominus\oplus)^+\oplus^+$, or
(3) $(\ominus\oplus)^+\ominus\ominus^+\oplus^+$.

Here, as usual r^+ stands for rr^*. Besides the choice of the first subpath, this partitioning is unique. Note that each subpath starts with $\ominus$ and ends with $\oplus$. In particular, all vertices first(p_i) saturate positively.

For each subpath p_i, we construct a path p_i' such that the following statements hold.

(a) $\emptyset \neq p_i' \cap C \subseteq p_i - \{\text{last}(p_i)\}$;
(b) $\text{last}(p_i') = \text{first}(p_{i+1}')$
(c) $\text{pat}(p_i') = \text{pat}(p_i)$;
(d) There is a vertex v in $p_i \cap p_i'$ which has reversed polarity in p_i' compared to p_i (i.e., the label between v and its successor is different in p_i' and in p_i);

Conditions (a) and (b) guarantee that the p_i' can again be concatenated to a cycle C'. Condition (c) ensures that this cycle has the same pattern as C. Let p_i consist of $v_0, \ldots, v_m$ and p_i' of $v_0', \ldots, v_m'$. We color each vertex v_i' by col(v_i). The last condition ensures that the vertices of Bad(C) can be directly saturated by this coloring of C'. Note, that the latter condition is already guaranteed by each single subpath p_i'.

We construct the paths p_i' inductively. It should be noted first, that in the construcion of p_i' the first vertex of p_i' can always be chosen arbitrarily from K. In particular, it can always be chosen in a way that ensures condition (b). Hence, we only need to care about conditions (a), (c) and (d).

Let $i > 0$. Let Q_i denote the set $\bigcup_{j=1}^{i-1} p_j'$, let $K_i := K - Q_i$ and let $p_i = v_0, \ldots, v_m$. We will describe the construction of $p_i' = v_0', \ldots, v_m'$. In most cases, p_i' will consist of vertices of p_i and of vertices from K_i. As it does not make a difference

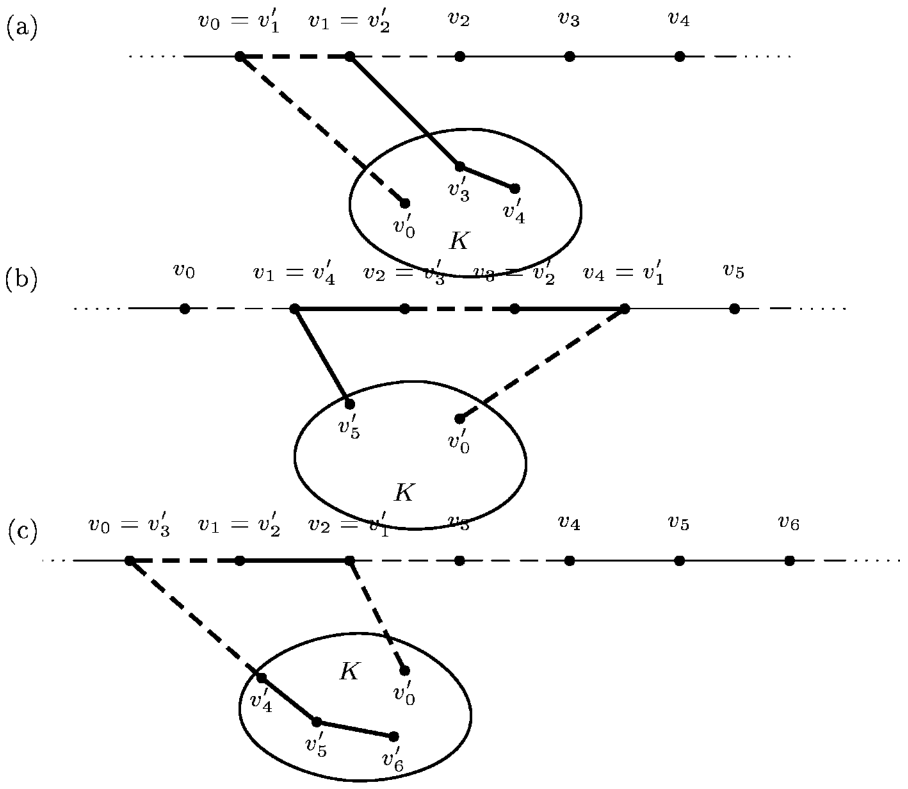

FIG. 10. Construction of p'_i from p_i. (a) displays case (1) with $j = 2$, (b) shows case (2) with $j = 2$ and (c) shows case (3) with $j = 1$ and $j' = 2$. The path p'_i is indicated by thick lines.

which vertices from K_i are actually chosen we simply write $*$ for an arbitrary vertex from K_i. Hence, for example, $*, v_0, \ldots, v_{j-1}, *^{m-j}$ refers to a path which consists of a vertex from K_i, followed by the subpath $v_0, \ldots, v_{j-1}$ of p_i and completed by $m - j$ vertices from K_i. In all cases, the path ends with an appropriate number of vertices from K_i. We simply write $**$ to denote this terminal part of the path.

We distinguish between three main cases, depending on whether the pattern of p_i conforms to (1), (2) or (3). Case (3) splits into two subcases depending on whether the second block of (negative) edges has length 2 or more. In the latter case of more than two negative edges, there will be some further local case distinctions. In all cases the verification of conditions (a), (c) and (d) is straightforward given the respective constructions.

If p_i conforms to (1) and $\text{pat}(p_i) = \ominus^j \oplus^{m-j}$, then we choose $p'_i = *, v_0, \ldots, v_{j-1}, **$ (cf. Figure 10(a)).

If p_i conforms to (2) and $\text{pat}(p_i) = (\ominus\oplus)^j \oplus^{m-2j}$, then we choose $p'_i = *, v_{2j}, \ldots, v_1, **$ (cf. Figure 10(b)).

If p_i conforms to (3), let j, j', j'' be such that $\text{pat}(p_i) = (\ominus\oplus)^j \ominus^{j'} \oplus^{j''}$ (where $j' > 1$ and $j, j'' \geq 1$). If $j' = 2$, we choose $p'_i = *, v_{2j}, \ldots, v_0, **$ (cf. Figure 10(c)).

The remaining case is when $j' > 2$. This is the only case where we have to consider other parts of the graph, besides C and K_i. As we assume that G is not

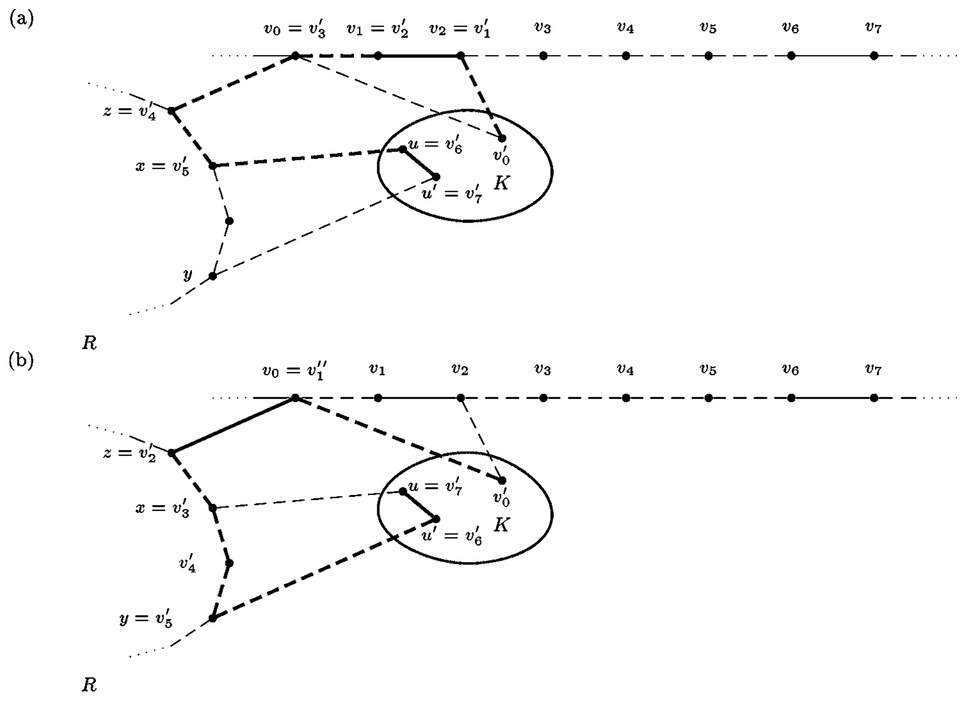

FIG. 11. Construction of p_i' from p_i in case of pattern (3) (with $j = 1$, $j' = 4$), subcase (i). The crucial point is here, that p_i' can be found, no matter the value of lab(v_0, z).

(k, t)-special, by the choice of t and k, the negative graph $G - (C \cup K_i \cup Q_i)$ has tree-width more than t and therefore it contains a negative cycle R with at least $2l$ vertices. The path p_i' will start in K_i, traverse an initial part of p_i in reverse direction until v_0, pass to a vertex z of R, follow R for a while and finally go back to K_i. A difficulty lies in the fact that we neither know lab(v_0, z) nor the labels of edges between R and K_i in advance. If lab$(v_0, z) = \oplus$ this edge can match the last $\oplus$ of the $(\ominus\oplus)^j$ part of pat(p_i). Otherwise, it can match the second $\ominus$ of the $\ominus^{j'}$ part. To deal with this difficulty, we show that we can always find z in R such that

—there exist x, y in R and u, u' in K_i with $d(z, x) = j' - 2$, $d(z, y) = j'$ (distances counted on R) and lab$(x, u) =$ lab$(y, u') = \oplus$, or

—there exist x, y in R and u in K_i with $d(z, x) = j' - 3$, $d(z, y) = j' - 1$ and lab$(x, u) =$ lab$(y, u') = \ominus$.

Then we choose $p_i' = *, v_{2j}, \ldots, v_0, z, \ldots, x, u, **$ if lab$(v_0, z) = \ominus$
or $p_i' = *, v_{2j-2}, \ldots, v_0, z, \ldots, y, u', **$, otherwise. See Figure 11 for an example of the second case.

In order to prove the existence of z, we consider three subcases.

(i) If there exist vertices x and y on R with distance 2 on R and neither positive(x, K_i) nor positive(y, K_i) holds, then we choose z on R that has distance $j' - 3$ from x and distance $j' - 1$ from y.

(ii) If there exist vertices x and y on R with distance 2 on R and neither negative(x, K_i) nor negative(y, K_i) holds, then we choose a vertex z on R that has distance $j' - 2$ from x and distance j' from y.

(iii) Finally, assume that neither of the above two cases holds. This implies that, for each pair of vertices x, y of distance 2 on R, either negative(x, K_i) and positive(y, K_i) or positive(x, K_i) and negative(y, K_i). We conclude that R can be written as a sequence $a_0, a_1, a_2, \ldots, a_g$ such that, for each h, negative(a_{4h}, K_i), negative(a_{4h+1}, K_i), positive(a_{4h+2}, K_i), and positive(a_{4h+3}, K_i). In particular, if the distance of two vertices x, y is a multiple of 4, then they are connected to K_i in the same way.

In this case, we will choose z between x and y on R (in the natural sense). The choice of x, y, z depends on the modulus of j' with respect to 4. If

(a) $j' = 4s + 3$, we choose x and y of distance $8s + 4$ with positive(x, K_i) and positive(y, K_i) and z of distance $4s + 3$ from x and $4s + 1$ from y;
(b) $j' = 4s + 1$, we choose x and y of distance $8s$ with positive(x, K_i) and positive(y, K_i) and z of distance $4s + 1$ from x and $4s - 1$ from y;
(c) $j' = 4s + 2$, we choose x and y of distance $8s$ with negative(x, K_i) and negative(y, K_i) and z of distance $4s + 1$ from x and $4s - 1$ from y;
(d) $j' = 4s$, we choose x and y of distance $8s - 4$ with negative(x, K_i) and negative(y, K_i) and z of distance $4s - 1$ from x and $4s - 3$ from y.

In analogy with the cases (i) and (ii) above it can be seen that p'_i can be constructed in each of the four cases.

This completes the description of the construction of C'.

6.3.1.2. FROM C' TO D. To summarize, C' has the following properties.

—Bad(C) $\subseteq$ Good(C') $\cup$ Neu(C'), that is, all bad vertices with respect to C are no longer bad with respect to C'.
—On the other hand, it might be the case that some vertices from Good(C) and Neu(C) are in Bad(C'). We know that there are not many good vertices with respect to C, as by assumption $|$Good(C)$| \leq \log(|$Bad(C)$|)$. We only have to worry about the vertices from Neu(C). Many of them may be bad with respect to C', in fact it might even be the case that $|$Bad(C')$| > |$Bad(C)$|$. It will turn out in the following that the existence of certain patterns in C guarantees that no vertices from Neu(C) are in Bad(C'). If none of these patterns occurs in C, we have to modify one or at most two subpaths of C'.

Below, we show that at least one of the following statements holds.

—The coloring col of C can be extended to a legal coloring of G.
—$|$Bad(C')$| < |$Bad(C)$|$.
—There is a self-saturating cycle D with pat(D) $=$ pat(C') $=$ pat(C) such that $|$Bad(D)$| < |$Bad(C)$|$.

We note again that, in the latter case, D will differ from C' only in at most two subpaths.

The main idea to ensure the saturation of the vertices in Neu(C) is as follows. Assume there are vertices x_0 and x_1 in $C \cap D$, such that x_0 is a $\ominus$-assistant and x_1 is a $\oplus$-assistant with respect to C and both saturate in D via edges of opposite polarity than in C (we say, that *their polarity is reversed*). As each vertex in Neu(C) is saturated with respect to C by at most one of x_0 and x_1, it follows that all vertices in Neu(C) are saturated by at least one of x_0 and x_1 with respect to D, that is, $\text{Neu}(C) \subseteq \text{Neu}(D) \cup \text{Good}(D)$. Therefore, it is sufficient to find a $\ominus$-assistant x_0 of C and a $\oplus$-assistant x_1 of C that have reversed polarity in D.

There are some easy cases in which such x_0 and x_1 can be found in C'. In these cases, we can simply set $D := C'$.

—If in the construction of C' case (2) occurs at least once or case (3) occurs with a pattern $(\ominus\oplus)^j\ominus^{j'}\oplus^{j''}$ and $j > 1$, we can choose $x_0 = v_2$ and $x_1 = v_1$.
—If case (3) occurs with $j = 1$ and $j' = 2$, then we can also choose $x_0 = v_2$ and $x_1 = v_1$.
—If case (3) occurs with $j' = 1$, $j' > 2$ and v_0 is connected by a negative edge to z, then we can again choose $x_0 = v_2$ and $x_1 = v_1$.

Otherwise, we have to modify C' to obtain D. Hence, in the following, we can assume that only case (1) and case (3) with $j = 1$, $j' > 2$ and a positive edge between v_0 and z occur in the construction of C'. An inspection of the construction shows that in all these cases the vertex v_0 saturates positively with respect to C but negatively with respect to C'. Hence, if the set of vertices $\text{first}(p_i)$ contains $\ominus$-assistants as well as $\oplus$-assistants with respect to C, then we are done again.

Otherwise, we distinguish the two cases that all vertices $\text{first}(p_i)$ are $\ominus$-assistants or that all of them are $\oplus$-assistants.

We write, in the following, $\text{Neu}_\ominus(C)$ for the vertices of $\text{Neu}(C)$ that are saturated by some $\ominus$-assistants of C and $\text{Neu}_\oplus(C)$ for those that are saturated by some $\oplus$-assistants of C.

—First, assume all vertices $\text{first}(p_i)$ are $\ominus$-assistants.
Hence, all vertices in $\text{Neu}_\oplus(C)$ are saturated by each single of these vertices with respect to C'.

If, for each vertex $v \in \text{Bad}(C)$, there is a vertex in $\text{Neu}_\ominus(C)$ that is connected negatively to v, then C can be extended to a legal coloring of G. If this is not the case, then there is a vertex $v \in \text{Bad}(C)$ such that $\text{positive}(v, \text{Neu}_\ominus(C))$. Note that v does not need to be a member of the clique K.

If a subpath of pattern (1) was used in the construction of C', then, as all $\text{first}(p_i)$ are $\ominus$-assistants, we can conclude that the pattern is of the form $\ominus^j\oplus$. Hence, we can construct D by replacing v'_m of p'_i by v. As v is in $\text{Bad}(C)$, we can be sure that we can use it also as the first vertex of p'_{i+1}. All vertices from $\text{Neu}_\ominus(C)$ are saturated by v.

Hence, we can assume that all subpaths are of the form (3) with $j = 1$ and $j'' = 1$.

If there is a subpath p_i of this kind with $j' = 2$ (hence, $p_i = v_0, \ldots, v_5$), then we can replace p'_i by $*, v_2, v_1, v_0, *$ and we can choose $x_0 = v_2$ and $x_1 = v_1$ $(= w_1)$.

It remains the case where all p_i are of type (3) with $j' \geq 3$.

If all of them have $j' = 3$, then C is symmetric and we can simply choose D as the reversal of C. In analogy to Lemma 6.7, it follows that G is in $\text{SATU}(P)$.

The same can be done if C consists of only one subpath p_i at all and this is of the form (3).

The only remaining case is when there are at least two subpaths of type (3) and at least one of them has $j' \geq 4$.

Let, without loss of generality, $p_1 = v_0, \ldots, v_m$ be a subpath of C with maximal j' and let $p_2 = u_0, \ldots, u_{m'}$. Let z_1 and z_2 denote the vertices that played the role of z in the construction of p'_1 and p'_2, respectively.

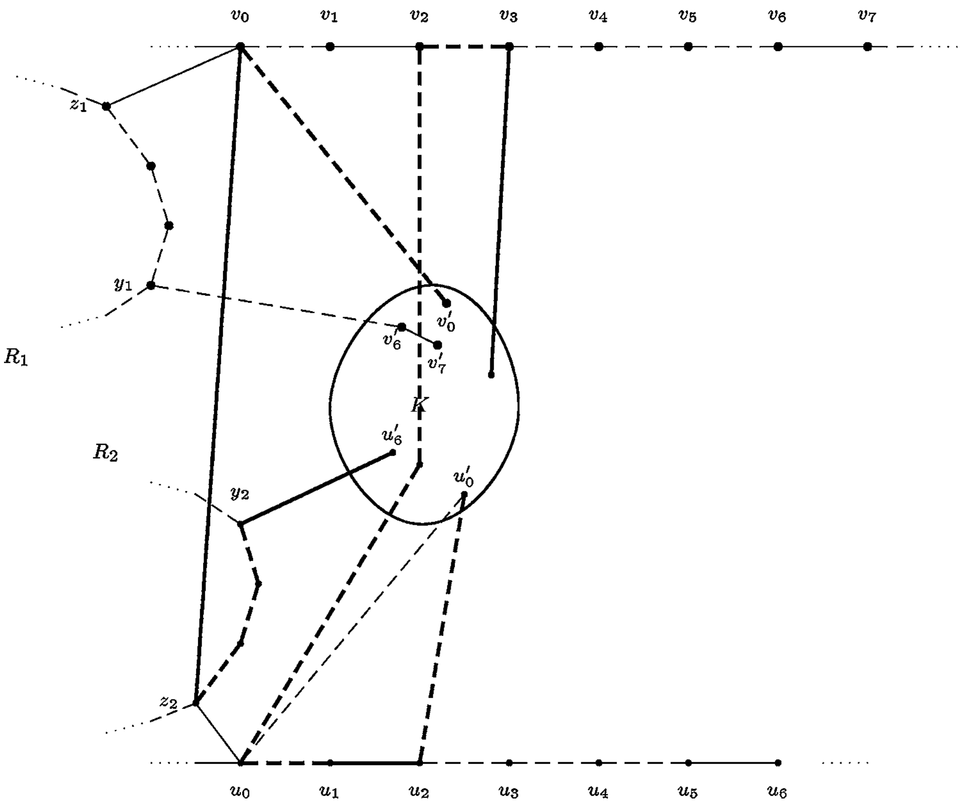

FIG. 12. Modification of two subpaths $p_1' = v_0', v_0, z_1, \ldots, y_1, v_6', v_7'$ and $p_2' = u_0', u_0, z_2, \ldots, y_2, u_6'$. The new paths are $p_1'' = u_0', u_2, u_1, u_0, *, v_2, v_3, *$ and $p_2'' = v_0', v_0, z_2, \ldots, y_2, u_6'$.

—If negative(first(p_1), z_2) or negative(first(p_2), z_1), then we can construct paths p_1'' and p_2'' in accordance with our general strategy but by using the prefix of p_1 in the construction of p_2'' and the prefix of p_2 in the construction of p_1''. Hence, we can replace p_1' and p_2' by p_1'' and p_2'', respectively. Then, at least one of first(p_1'') and first(p_2'') is connected negatively with its corresponding z. Hence, one of these paths gives us suitable x_0 and x_1.

—Otherwise, we can replace the second vertex of p_2' (that is, first(p_2)) by first(p_1) and we can replace p_1' by $*, u_2, u_1, u_0, *, v_2, \ldots, v_{m-4}, *$. We can choose $x_0 = u_2$ and $x_1 = u_1$. See Figure 12 for an illustrating example.

This finishes the description of the case where all w_0 are $\ominus$-assistants.

—Now assume that all vertices first(p_i) are $\oplus$-assistants. This case is less complicated than the one before. We may have subpaths of type (1) with $m - j \geq 2$ and of type 3 with $j = 1$ and $j'' \geq 2$.

We distinguish five subcases.

—If there is a subpath $p_i = v_0, \ldots, v_m$ of type (1) with $j = 2$, then we replace p_i' by $*, v_0, *, v_2, *^*$ and we can set $x_0 = v_2$ and $x_1 = v_0$.

—If there is a subpath of type (1) with $j = 3$, we proceed as follows. First it should be observed that none of the paths p_i' makes use of the $\oplus^{m-j}$ part (including the first vertex of this part). Let the first vertex of the $\oplus$-part of p_{i-1} be denoted z_0 (this one is connected negatively with its predecessor in p_{i-1}), the second z_1 and the third z_2 (recall that there are at least two positive edges). We replace p'_i by $*, z_2, *, z_1, z_0, *^*$. We can set $x_0 = z_1$ and $x_1 = z_2$.

—If there is a subpath of type (1) with $j \geq 4$, then we replace p'_i by $*, v_{j+1}, *, v_0, \ldots, v_{j-3}, *^*$ and set $x_0 = v_{j+1}$ and $x_1 = v_0$.

—If there are no subpaths of type (1) and there is a subpath of type (3) with $j' = 2$, we replace p'_i by $*, v_2, v_1, v_0, *^*$ and set $x_0 = v_2$ and $x_1 = v_1$.

—Otherwise, there must be a subpath of type (3) with $j' \geq 3$. In this case, we replace p'_i by $*, v_{-1}, v_0, *, v_2, *^*$, if $j' = 3$ and by $*, v_{-1}, v_0, *, v_2, \ldots, v_{j'}, *^*$, if $j' > 3$. In either case, we can set $x_0 = v_2$ and $x_1 = v_0$.

This completes the proof of Lemma 6.8. □

6.3.2. *Dealing with a Constant Number of Bad Vertices.* From Lemma 6.8, we can conclude that, whenever a nonspecial graph G has a self-saturating mixed cycle of some size l then it has such a cycle C with $|\text{Bad}(C)| \leq c$ for some constant c that only depends on l. In this section, we show that we can always extend the coloring of C to a legal coloring of G.

We start with a combinatorial lemma that states a relationship between being nonspecial and the existence of long paths of pattern $(\ominus\oplus)^*$ or $(\oplus\ominus)^*$ for complete bipartite $\ominus$-$\oplus$-graphs. Let G be a complete bipartite graph with vertex partition (V_1, V_2) and edge labels from $\{\ominus, \oplus\}$. We call two vertices $u, u' \in V_1$ *equivalent with respect to a subset* $T \subseteq V_2$, if for each $w \in T$ $\text{lab}(u, w) = \text{lab}(u', w)$.

LEMMA 6.9. *Let $l > 0$ and let G be a complete bipartite graph with vertex partition (V_1, V_2) and edge labels from $\{\ominus, \oplus\}$. Then at least one of the following two conditions holds.*

(*a*) *There is a path of pattern $(\oplus\ominus)^l$ starting from (and ending in) a vertex from V_1.*

(*b*) *The vertices in V_1 can be partitioned into sets $W_1, \ldots, W_{2^l}$ and there is a set $T \subseteq V_2$ of size at most 2^{2l} such that for all $i \leq l$ the vertices in the set W_i are equivalent with respect to $V_2 - T$.*

PROOF. Consider the execution of the following algorithm on input G.

```
(1) $T := \emptyset$, choose $v_1 \in V_1$ arbitrarily
(2) FOR $i := 2$ TO $2^l$ DO
      —IF there exists $w \in V_1$ not equivalent to any $v_j$, $j < i$, with respect to $V_2 - T$
         —$v_i := w$
         —Pick for each $j < i$ a vertex $x_{ij} \in V_2 - T$ such that $\text{lab}(v_i, x_{ij}) \neq \text{lab}(v_j, x_{ij})$
         —$T := T \cup \{x_{ij} \mid j < i\}$
      —OTHERWISE Stop
```

There are two possibilities how this algorithm might terminate. The first is that no furher vertex w is found. Then every vertex in V_1 is equivalent to one of the v_i with respect to $V_2 - T$. As $|T| < i^2$ after the execution of i steps, condition (b) follows. In the other case, we get 2^l vertices $v_1, \ldots, v_{2^l}$ that are pairwise not equivalent with respect to V_2. Let H be the directed graph with vertex set $\{v_1, \ldots, v_{2^l}\}$ and edges defined as follows. For $j < i$ there is an edge (v_j, v_i) in H if $\text{lab}(v_j, x_{ij}) = \oplus$. If $\text{lab}(v_j, x_{ij}) = \ominus$, then there is an edge (v_j, v_i) in H. A directed path $v_{i_0}, \ldots, v_{i_m}$ in

H corresponds to a path $v_{i_0}, x_{i_0 i_1}, v_{i_1}, x_{i_1 i_2}, \ldots, v_{i_m}$ with pattern $(\oplus\ominus)^m$ in G (where $x_{ij} := x_{ji}$, for $i < j$). Note that, by construction, all the vertices $x_{i_j i_{j+1}}$ are distinct. It is easy to show that, as H is complete and has 2^l vertices, it has a directed path of length l. Hence, condition (a) holds in G. □

Now we come back to the saturation problem.

LEMMA 6.10. *Let P be a pattern graph and let c, l be integers. Then there are constants k and t such that the following holds. If G is a $\ominus$-$\oplus$graph that is not (k, t)-special and has a mixed self-saturating cycle C of length l of pattern different from $(\ominus\oplus)^*$ and $(\ominus\oplus)^*$ with $|Bad(C)| \leq c$ and $|Good(C)| \leq log(|Bad(C)|)$, then $G \in SATU(P)$ via a coloring that extends the coloring of C.*

PROOF. Let c_1 be large enough such that a graph of tree-width $>c_1$ always contains a path of length l (cf. Proposition 5.11). Let $c_2 = l + \log(c) + c + 2^{2l+1}$ and $c_3 = 2^{l+1}2^{c_2}$. We prove by induction on c that a mixed self-saturating coloring of a cycle C of length l can be extended to a legal coloring of G. If $c = 0$, then $\text{Bad}(C)$ is empty and there is nothing to prove. Let therefore $c > 0$. Let k' and t' be the constants obtained from the statement of the Lemma for $c' = c - 1$ and $l' = l$. We set $k = \max(c_3, 2^l k')$ and $t = \max(c_1 + c_2, t' + l')$.

Let now G be a graph which is not (k, t)-special and let $C = w_1, \ldots, w_l$ be a cycle with self-saturating coloring col. If $|\text{Bad}(C)| < c$, we are done by induction. Hence, we can assume $|\text{Bad}(C)| = c$. We show that col can be extended to a legal coloring of G. Let therefore $v \in \text{Bad}(C)$.

By $p_m(w_j)$ we denote the path $w_{j-m}, w_{j-m+1}, \ldots, w_j$, where indices are modulo l. We construct a *saturating path* $p = v_0, \ldots, v_m$ *for* v, for some $m \leq l$, that is, a path such that, for some j,

—$v_0 = v$,

—$v_m = w_j \in C$,

—$\{v_0, \ldots, v_{m-1}\} \cap C = \emptyset$, and

—$\text{pat}(p) = \text{pat}(p_m(w_j))$.

Given such a path, we can color the vertices v_i via $\text{col}(v_i) = \text{col}(w_{j-m+i})$. Hence, each vertex v_i becomes saturated by v_{i+1}. Let $G' = G - \{v_0, \ldots, v_{m-1}\}$. Toward a contradiction, assume that G' is (k', t')-special with partition $A, A_1 \ldots, A_{k'}$. Then it follows that G is $(2^l k', t + l)$-special by adding the vertices $v_0, \ldots, v_{m-1}$ to A and partitioning each set A_i, $i > 0$, into at most $2^m \leq 2^l$ subsets with respect to the polarity of their edges to $v_0, \ldots, v_{m-1}$. By the choice of k and t, G would be (k, t)-special, a contradiction. Therefore, G' is not (k', t')-special. As $|\text{Bad}(C)| \leq c - 1$ in G', G' can be legally colored with a coloring which extends col, by induction. By adding back $\{v_1, \ldots, v_m\}$ with their chosen colors, we get a legal coloring of G.

It remains to describe the construction of p. First, we choose a fixed extension of col to the vertices of $\text{Neu}(C)$ such that all of them are saturated and all of them are colored with colors that also appear in $\text{col}(C)$. For each $u \in \text{Neu}(C)$, let $j(u)$ be such that u is saturated by $w_{j(u)+1}$, in particular, $\text{col}(u) = \text{col}(w_{j(u)})$.

For a $\ominus$-$\oplus$-string s let B_s be the set of vertices from $\text{Neu}(C)$ which saturate via paths of pattern s. More formally, B_s is the set of vertices $u \in \text{Neu}(C)$ such that $\text{pat}(p_i(w_{j(u)})) = s$, where i is the length of s. Note that, by definition, $B_\ominus \cap B_\oplus = \emptyset$ and that $B_s \subseteq B_{s'}$ if s' is a suffix of s.

If, for some $\ominus$-$\oplus$-string s, there is a path p' of pattern s from v to a vertex u in B_s, consisting only of vertices in Neu(C), then one can obtain p by adding $w_{j(u)+1}$ to p'. We call such a path p' *useful for* v. Our goal is to show that a useful path for v exists.

In any of the following situations, the existence of a useful path p' follows.

—If v has a negative edge to a vertex u from $B_\ominus$ or a positive edge to a vertex u from $B_\oplus$, then $p' := v, u$.

—If there is a positive edge between vertices $u_1 \in B_{\ominus\ominus}$ and $u_2 \in B_{\oplus\oplus}$ and v is connected by positive edges to the vertices in $B_{\ominus\ominus}$, then p' can be chosen as v, u_1, u_2. Analogously, if there is a negative edge between $B_{\ominus\ominus}$ and $B_{\oplus\oplus}$. Hence, in the following we can assume that there is *no* edge between $B_{\ominus\ominus}$ and $B_{\oplus\oplus}$ which implies that $B_{\ominus\ominus} = \emptyset$ or $B_{\oplus\oplus} = \emptyset$.

—If there is a path $q = z_0, \ldots, z_l$ of pattern $\oplus^l$ in $B_\oplus$ and v is connected by negative edges to the vertices in $B_\oplus$, then we can construct p as follows. Let $j = j(z_l)$. As C is a mixed cycle, there is a j' such that pat$(p_{j'}(w_j)) = \ominus\oplus^{j'}$. Hence, we can choose p' as $v, z_{l-j'}, \ldots, z_l$. An analogous construction can be done if there is a path q of length l of pattern $\ominus^l$ in $B_\ominus$.

—If there is a positive edge between two vertices, $u_1, u_2 \in B_{\ominus\oplus}$ and v is connected by negative edges to the vertices in $B_{\ominus\oplus}$, then we can choose $p' = v, u_1, u_2$.

—Analogously, if there is a negative edge between two vertices in $B_{\oplus\ominus}$ and v is connected by positive edges to the vertices in $B_{\ominus\oplus}$, then we can choose $p' = v, u_1, u_2$.

Let us therefore assume that none of these situations occur, that is,

—positive(v, $B_\ominus$), negative(v, $B_\oplus$),

—without loss of generality $B_{\oplus\oplus} = \emptyset$,

—there is neither a positive path of length l in $B_\oplus$ nor a negative path of length l in $B_\ominus$, in particular, the positive graph induced by $B_{\ominus\ominus}$ has tree-width at most c_1, and

—$B_{\ominus\oplus}$ is a negative clique and $B_{\oplus\ominus}$ is a positive clique.

Now we apply Lemma 6.9 to the bipartite graph G' with $V_1 := B_{\oplus\ominus}$ and $V_2 := B_{\ominus\oplus}$ and the edges between these two sets induced from G. Hence, there is a path of pattern $(\oplus\ominus)^l$ starting from and ending in a vertex from $B_{\oplus\ominus}$ or condition (b) of Lemma 6.9 holds.

Assume first the existence of such a path $q = z_0, \ldots, z_{2l}$ and let $j = j(z_{2l})$. Recall that z_i is in $= B_{\oplus\ominus}$, for even i, and in $B_{\ominus\oplus}$, for odd i. As the pattern of C neither is of the form $(\ominus\oplus)^*$ nor of the form $(\ominus\oplus)^*$ we can assume that there is a j', with $2j' \leq l$, such that pat$(p_{2j'+1}(w_{j-1})) = \oplus(\oplus\ominus)^{j'}$ or pat$(p_{2j'+2}(w_{j-1})) = \ominus\ominus(\oplus\ominus)^{j'}$. In the first case, we can choose $p' = v, z_{2l-2j'-1}, \ldots, z_{2l}$ as a useful path, in the second case $p' = v, z_{2l-2j'-2}, \ldots, z_{2l}$.

By switching V_1 and V_2 as well as $\ominus$ and $\oplus$, we get, again from Lemma 6.9, that there is a path of pattern $(\ominus\oplus)^l$ starting from and ending in a vertex from $B_{\ominus\oplus}$ or the analogue of condition (b) holds for $B_{\ominus\oplus}$. If there is such a path, we can construct p', analogously.

Hence, it remains to consider the situation where in both cases condition (b) holds. Let T, $W_1, \ldots, W_{2^l}$ be the sets obtained from the first application of Lemma 6.9

and let $T', W'_1, \ldots, W'_{2^l}$ be the respective sets obtained from the second application. Let

$$S := C \cup \text{Good}(C) \cup \text{Bad}(C) \cup T \cup T'.$$

Note that $|S| \leq l + \log(c) + c + 2^{2l+1} = c_2$. For each subset $U \subseteq S$ and each $i \leq 2^l$, let $W_i(U)$ be the set of vertices of W_i that are connected positively with the vertices in U and negatively with the vertices in $S - U$. Let $W'_i(U)$ be defined analogously. Then, for each i and $U \subseteq S$, the vertices in $W_i(U)$ (as well as those in $W'_i(U)$) are equivalent with respect to V. Therefore, the set $A := B_{\ominus\ominus} \cup S$ and the $(\leq 2^{l+1}2^{c_2} = c_3)$ sets $W_i(U)$ and $W'_i(U)$ partition V and actually witness that G is $(c_3, c_1 + c_2)$-special, the desired contradiction.

This completes the proof of the lemma. □

ACKNOWLEDGMENTS. We thank Nicole Schweikardt for her comments on an earlier version of this paper, which led to important improvements. In addition, we wish to thank Martin Grohe, Stefan Wöhrle, and the anonymous referees for their useful comments.

REFERENCES

BODLAENDER, H. L. 1996. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM J. Comput. 25*, 6, 1305–1317.

BÖRGER, E., GRÄDEL, E., AND GUREVICH, Y. 1997. *The Classical Decision Problem*. Springer, Berlin, Germany.

BÜCHI, J. R. 1960. Weak second-order arithmetic and finite automata. *Z. Math. Logik Grund. Math. 6*, 66–92.

COURCELLE, B. 1990. The monadic second-order logic of graphs I: Recognizable sets of finite graphs. *Inf. Comput. 85*, 12–75.

DOWNEY, R. G., AND FELLOWS, M. R. 1999. *Parametrized Complexity*. Springer-Verlag, New York.

EITER, TH., GOTTLOB, G., AND GUREVICH, Y. 2000. Existential second order logic over strings. *J. ACM 47*, 1, 77–131. (Preliminary version in *LICS'98*.)

EITER, TH., GOTTLOB, G., AND SCHWENTICK, TH. 2002. Second order logic over strings: Regular and nonregular fragments. In *Proceedings of the 5th International Conference on Developments in Language Theory* (Vienna, Austria, July 16–21, 2001). Lecture Notes in Computer Science, vol. 2295. Springer-Verlag, New York, pp. 21–36.

FAGIN, R. 1974. Generalized first-order spectra and polynomial-time recognizable sets. In *Complexity of Computation*, R. M. Karp, ed. AMS, providence, R.F., pp. 43–74.

GAREY, M. R., AND JOHNSON, D. S. 1979. *Computers and Intractability. A Guide to the Theory of* NP-*Completeness*. Freeman, New York.

GRÄDEL, E. 1991. The Expressive power of second-order Horn Logic. In *Proceedings STACS-91*. Lecture Notes in Computer Science, vol. 480. Springer-Verlag, New York, pp. 466–477.

GRÄDEL, E. 1992. Capturing complexity classes with fragments of second order logic. *Theoret. Comput. Sci. 101*, 35–57.

IMMERMAN, N. 1988. Nondeterministic space is closed under complementation. *SIAM J. Comput. 17*, 935–939.

IMMERMAN, N. 1998. *Descriptive Complexity Theory*. Springer-Verlag, New York, 1999.

JONES, N. D., LIEN, Y. E., AND LAASER, W. T. 1976. New problems complete for nondeterministic log space. *Math. Syst. Theory 10*, 1–17.

LADNER, R. 1975. On the structure of polynomial time reducibility. *J. ACM 22*, 155–171.

PAPADIMITRIOU, C. H. 1994. *Computational Complexity*, Addison-Wesley, Reading, Pa.

PAPADIMITRIOU, C. H., AND YANNAKAKIS, M. 1991. Optimization, approximation, and complexity classes. *J. Comput. Syst. Sci. 43*, 425–440.

ROBERTSON, N., AND SEYMOUR, P. D. 1986. Graph minors II. Algorithmic aspects of tree-width. *J. Algorithms 7*, 309–322.

ROBERTSON, N., AND SEYMOUR, P. D. 1984. Graph minors III. Planar tree-width. *J. Combinat. Theory, Ser. B 36*, 49–64.

SZELEPCSÈNYI, R. 1988. The method of forced enumeration for nondeterministic automata. *Acta Inf., 26*, 279–284.

TARJAN, R. 1972. Depth-first search and linear graph algorithms. *SIAM J. Comput. 1*, 146–160.

THOMASSEN, C. 1988. On the presence of disjoint subgraphs of a specified type. Algorithmic aspects of tree-width. *J. Graph Theory 12,* 1, 101–111.

TRAHTENBROT, B. 1963. The impossibility of an algorithm for the decision problem for finite domains (Russian). *Doklady Academii Nauk SSR*, 70:569–572, 1950. English translation: *American Mathematical Society Translation Series 2*, 23, 1–5.

WANKE, E. 1994. Bounded tree-width and LOGCFL. *J. Algorithms 16*, 470–491.

RECEIVED JANUARY 2002; REVISED JULY 2003; ACCEPTED JULY 2003