Unification Algorithms Cannot be Combined in Polynomial Time

Miki Hermann[1] and Phokion G. Kolaitis[2] *

[1] CRIN (CNRS) and INRIA-Lorraine
BP 239, 54506 Vandœuvre-lès-Nancy, France. hermann@loria.fr
[2] Computer and Information Sciences, University of California, Santa Cruz
Santa Cruz, CA 95064, U.S.A. kolaitis@cse.ucsc.edu

Abstract. We establish that there is no polynomial-time general combination algorithm for unification in finitary equational theories, unless the complexity class #P of counting problems is contained in the class FP of function problems solvable in polynomial-time. The prevalent view in complexity theory is that such a collapse is extremely unlikely for a number of reasons, including the fact that the containment of #P in FP implies that P = NP. Our main result is obtained by establishing the intractability of the counting problem for general AG-unification, where AG is the equational theory of Abelian groups. Specifically, we show that computing the cardinality of a minimal complete set of unifiers for general AG-unification is a #P-hard problem. In contrast, AG-unification with constants is solvable in polynomial time. Since an algorithm for general AG-unification can be obtained as a combination of a polynomial-time algorithm for AG-unification with constants and a polynomial-time algorithm for syntactic unification, it follows that no polynomial-time general combination algorithm exists, unless #P is contained in FP.

1 Introduction and summary of results

Unification in equational theories is the keystone of automated deduction. It is used extensively in several areas of computer science, including theorem proving, database systems, natural language processing, logic programming, computer algebra, and program verification. Plotkin [Plo72] was the first to formulate explicitly the idea that theorem provers should have built-in algorithms for unification in equational theories. His pioneering article provided the impetus for the development of the entire field of equational unification.

Since there are equational theories with an undecidable unification problem, no general algorithm for unification in an arbitrary equational theory exists. Instead, different special-purpose unification algorithms or procedures have to be designed for equational theories with a decidable unification problem. Nevertheless, one may still hope to obtain a unification algorithm for a given equational theory as a *combination* of existing unification algorithms for the components of the theory. More precisely, let $\mathcal{F}$ be a signature, E a finite set of equational

* Research of this author was partially supported by NSF Grants No. CCR-9307758

axioms generating the theory, and $\mathrm{Th}(\mathcal{F}, \mathrm{E})$ the equational theory generated by E. Suppose that the signature $\mathcal{F}$ and the equational axioms in E can be partitioned into disjoint sets $\mathcal{F}_1$, $\mathcal{F}_2$ and E_1, E_2 such that the theories $\mathrm{Th}(\mathcal{F}_1, \mathrm{E}_1)$ and $\mathrm{Th}(\mathcal{F}_2, \mathrm{E}_2)$ have decidable unification problems. The question is: does there exist a general method for combining unification algorithms for $\mathrm{Th}(\mathcal{F}_1, \mathrm{E}_1)$ and $\mathrm{Th}(\mathcal{F}_2, \mathrm{E}_2)$ into a new unification algorithm for the entire theory $\mathrm{Th}(\mathcal{F}, \mathrm{E})$?

By comparing the signature $\mathcal{F}$ with the symbols $sig(\mathrm{E})$ occurring in the set E of equational axioms, we distinguish between three kinds of equational unification. If $sig(\mathrm{E}) = \mathcal{F}$, which means that a unification problem may contain only symbols occurring in the equations E, then we speak about *elementary* E-*unification*. If the signature $\mathcal{F}$ contains additional free constant symbols, but no free function symbols, then we speak about E-*unification with constants*. Finally, if the signature $\mathcal{F}$ contains both additional free constant and free function symbols, then we speak about *general* E-*unification*. Quite often, it is much easier to design an algorithm for elementary E-unification or E-unification with constants than an algorithm for general E-unification. Note, however, that general E-unification can be viewed as the combination of E-unification with constants and syntactic unification, where syntactic unification is general unification in the empty theory. Thus, a general method for combining unification algorithms makes it possible to produce a general E-unification algorithm in a uniform way, provided an algorithm for E-unification with constants exists.

The development of combination algorithms originated with Stickel's algorithm for general associative-commutative (AC) unification [Sti81]. Stickel first constructed an algorithm for elementary AC-unification and then introduced a special-purpose combination algorithm for general AC-unification (actually, with several AC-symbols) that used the algorithm for elementary AC-unification and the algorithm for syntactic unification as subroutines. The termination of Stickel's algorithm was proved by Fages [Fag87]. Similar work was carried out by Herold and Siekmann [HS87]. More general combination problems were treated by Yelick, Kirchner, Herold, Tidén, Boudet, Jouannaud, and Schmidt-Schauß, who designed algorithms for combination of equational theories that satisfy certain restrictions on the syntactic form of their axioms. Kirchner [Kir85] requires E_1 and E_2 to be sets of simple axioms. Yelick [Yel87] gives a solution for the combination of regular and collapse-free theories. Similar results with the same restriction were obtained by Herold [Her86]. Tidén [Tid86] extended Yelick's result to collapse-free theories. Boudet, Jouannaud & Schmidt-Schauß [BJSS89] gave an algorithm for combining an arbitrary theory with a simple theory. The problem of how to combine unification algorithms for arbitrary disjoint theories was finally solved by Schmidt-Schauß [SS89]. A more efficient version of this combination method was given by Boudet [Bou93]. Using a new approach, Baader and Schulz [BS92] presented a combination method for decision problems in disjoint equational theories; a slight modification gives rise to a method for combining algorithms for unification in two disjoint equational theories. This method is based on linear constant restriction, a notion that generalizes Schmidt-Schauß' approach, where constant elimination problems have to be solved. Recently, an

attempt has been made to relax the condition that the equational theories must have disjoint signatures in the combination problem [KR94]. Although there are classes of non-disjoint equational theories for which a combination algorithm exists, the main problem with non-disjoint theories is that provably no general combination algorithm exists for them, even if one restricts attention to finitary theories generated by a finite set of simple linear equational axioms (cf. [DKR94]).

Every existing combination algorithm has an exponential running time. In particular, even if there exist polynomial-time unification algorithms A_1 and A_2 for the disjoint theories $\mathrm{Th}(\mathcal{F}_1, \mathrm{E}_1)$ and $\mathrm{Th}(\mathcal{F}_2, \mathrm{E}_2)$, every known general combination method will give rise to an exponential algorithm A for unification in the theory $\mathrm{Th}(\mathcal{F}_1 \cup \mathcal{F}_2, \mathrm{E}_1 \cup \mathrm{E}_2)$. In this paper we demonstrate that this exponential-time behaviour is not a deficiency of the known combination algorithms, but rather it is caused by the inherent intractability of the combination problem. More precisely, we show that there is no polynomial-time general combination algorithm for unification in finitary equational theories, unless the complexity class #P of counting problems is contained in the class FP of function problems solvable in polynomial time.

#P is the class of all functions f for which there is a nondeterministic Turing machine M that runs in polynomial time and has the property that $f(x)$ equals the number of accepting computation paths of M on every input x. The class #P was introduced and studied in depth by Valiant [Val79a,Val79b], who showed that several counting problems from graph theory, logic, and algebra are #P-complete. The prevalent view in complexity theory is that #P-complete problems are highly intractable and that, in particular, they are not contained in FP. Note that one of the reasons for this belief is the fact that if #P were contained in FP, then P = NP. In [HK95a,HK95b], we showed that the theory of #P-completeness can be applied to the analysis of equational matching and unification. For this, we introduced a class of counting problems that arise naturally in equational matching and unification, namely to compute the cardinality of a minimal complete set of E-matchers or E-unifiers, where E is a given finitary equational theory. We proved that counting the number of E-matchers or E-unifiers is a #P-hard problem for essentially every important equational theory E studied in the literature. It should be pointed out that a lower bound for counting the number of E-matchers or E-unifiers yields immediately a lower bound on all algorithms for computing minimal complete sets of E-matchers or E-unifiers, since any algorithm for E-matching or E-unification can be used to solve the associated counting problem within the same time bounds.

We derive the main result of this paper by analyzing the counting complexity of unification in the equational theory AG of Abelian groups. We exploit the fact that AG-unification with constants is unitary, whereas general AG-unification is finitary, but not unitary. Indeed, AG-unification with constants reduces to the problem of solving linear Diophantine systems over the integers (positive, negative, or zero); such systems are known to have a unique general solution obtained from the Hermite normal form of the corresponding integer matrix. Moreover,

this solution can be computed in polynomial time. Since an algorithm for general AG-unification can be obtained as a combination of a polynomial-time algorithm for AG-unification with constants and a polynomial-time algorithm for syntactic unification, it follows that if the counting problem for general AG-unification is intractable, then no polynomial-time general combination algorithm exists. We show this to be the case by establishing that computing the cardinality of a minimal complete set of unifiers for general AG-unification is a #P-hard problem.

We also establish that the counting problem for general BR-unification is #P-hard. This result yields a lower bound on the performance of all algorithms for general BR-unification.

2 Counting & combination problems in equational unification

In this section, we define the basic concepts, describe the family of counting problems arising in equational unification, and review the solution to the combination problem for unification algorithms. We also present here a minimum amount of the necessary background material from computational complexity and unification. Additional material for each of these topics can be found in [Pap94,JK91,BS94].

2.1 Counting problems and the complexity class #P

A *counting Turing machine* is a non-deterministic Turing machine equipped with an auxiliary output device on which it prints in binary notation the number of its accepting computations on a given input. The class #P consists of all functions that are computable by polynomial-time counting Turing machines, that is, machines for which there is a polynomial $p(n)$ such that the longest accepting computation of the machine over all inputs of size n is at most $p(n)$. These concepts were introduced and studied in depth by Valiant in his seminal papers [Val79a,Val79b].

Let Σ, Γ be nonempty alphabets and let $w\colon \Sigma^* \longrightarrow \mathcal{P}(\Gamma^*)$ be a function from the set Σ^* of strings over Σ to the power set $\mathcal{P}(\Gamma^*)$ of Γ^*. If x is a string in Σ^*, then we refer to $w(x)$ as the *witness set* for x and to the elements of $w(x)$ as *witnesses* for x. Every such function can be identified with the following *counting problem* w: given a string x in Σ^*, find the number of witnesses for x, i.e., find the cardinality of the witness set $w(x)$. Using these concepts, the class #P can be also described as the collection of all counting problems w such that the two conditions below hold: (1) there is a polynomial-time algorithm to tell, given strings x and y, whether $y \in w(x)$; (2) there is a $k \geq 1$ (which depends on w) such that $|y| \leq |x|^k$ for all $y \in w(x)$.

#SAT is the archetypal counting problem in #P: given a propositional formula φ, find the number of truth assignments that satisfy it. Here, the witness set $w(\varphi)$ consists of all truth assignments satisfying φ.

Counting problems relate to each other via *counting* and *parsimonious reductions*, which are stronger than the polynomial-time reductions between NP-

problems. Let $v: \Pi^* \to \mathcal{P}(\Delta^*)$ and $w: \Sigma^* \to \mathcal{P}(\Gamma^*)$ be two counting problems. A *polynomial-time many-one counting* (or, simply, *counting reduction*) from v to w is a pair of polynomial-time computable functions $\sigma: \Pi^* \to \Sigma^*$ and $\tau: N \to N$ such that $|v(x)| = \tau(|w(\sigma(x))|)$. Such reductions are often called *weakly parsimonious*. A *parsimonious reduction* from v to w is a counting reduction σ, τ from v to w such that τ is the identity function. A counting problem w is #P-*hard* if for each counting problem v in #P there is a counting reduction from v to w. If in addition w is a member of #P, then we say that w is #P-*complete*.

The proof of Cook's theorem [Coo71] that SAT is NP-complete can be modified to show that #SAT is #P-complete. Since many reductions of SAT to other NP-hard problems turn out to be parsimonious, it follows that the counting versions of many NP-complete problems are #P-complete. Valiant [Val79a] made also an unexpected, but fundamental, discovery by establishing that there are #P-complete problems whose underlying decision problem is solvable in polynomial time. The first and perhaps most well known among them is the following problem, which will be of particular use to us in the sequel.

#PERFECT MATCHINGS [Val79a]
Input: Bipartite graph G with $2n$ nodes.
Output: Number of *perfect matchings* of G , i.e., sets of n edges such that no pair of edges shares a common node.

#P-complete problems are considered to be truly intractable. Actually, in some sense they are substantially more intractable than NP-complete problems. To make this precise, one needs to bring in complexity classes of function problems, since #P is a collection of problems that require more complicated answers than the mere "yes" or "no" answers to decision problems. Let FP denote the class of all functions computable by a deterministic Turing machine in polynomial time; thus FP is the functional analog of P, the class of decision problems solvable in polynomial time. FP forms the first and lowest level of FPH, the functional analog of the polynomial hierarchy PH (cf. [Joh90, section 4.1]). The next level of FPH is the class $\mathrm{FP}^{\mathrm{NP}}$ of all functions that are computable in polynomial time using NP-oracles. In general, for each $k \geq 1$, the $(k+1)$-st level of FPH is the class of all functions computable in polynomial time with oracles from the k-th level of the polynomial hierarchy PH. There is strong evidence that #P is *not* contained in FPH, although this remains an oustanding open problem in complexity theory. First, it should be pointed out that if #P were contained in FP, then P = NP. Moreover, it is known that there are oracles relative to which #P is *not* contained in $\mathrm{FP}^{\mathrm{NP}}$. Finally, evidence of a different kind was provided by Toda [Tod89], who showed that the polynomial hierarchy PH is contained in the class $\mathrm{P}^{\#\mathrm{P}}$ of all decision problems computable in polynomial time using #P-oracles. As Johnson [Joh90] writes, this result indicates a precise sense in which #P dominates the entire polynomial hierarchy PH.

2.2 Equational theories and unification

If $\mathcal{F}$ is a signature and $\mathcal{X}$ is a countable set of variables, then $\mathcal{T}(\mathcal{F},\mathcal{X})$ denotes the set of all terms over the signature $\mathcal{F}$ and the variables in $\mathcal{X}$. If E is a set of equational axioms, then the *equational theory* $\mathrm{Th}(\mathcal{F},\mathrm{E})$ *induced by* E is the smallest congruence relation over $\mathcal{T}(\mathcal{F},\mathcal{X})$ containing E and closed under substitutions. We write $s =_\mathrm{E} t$ to denote that the pair (s,t) of terms is a member of $\mathrm{Th}(\mathcal{F},\mathrm{E})$. An E-*unifier of* s *and* t is a substitution ρ such that $s\rho =_\mathrm{E} t\rho$ holds; equivalently, an E-unifier of s and t is a solution of the equation $s \doteq_\mathrm{E} t$ in the algebra $\mathcal{T}(\mathcal{F},\mathcal{X})/=_\mathrm{E}$. If a minimal complete set of E-unifiers of s and t exists, then it is unique up to $\equiv^V_\mathrm{E}$ (cf. [FH86]). In this case, we let $\mu\mathrm{CSU}_\mathrm{E}(s,t)$ denote the minimal complete set of E-unifiers of s and t, if s and t are unifiable, or the empty set, otherwise. A theory E is said to be *unitary* if for every pair of terms (s,t) the set $\mu\mathrm{CSU}_\mathrm{E}(s,t)$ exists and $|\mu\mathrm{CSU}_\mathrm{E}(s,t)| \leq 1$. Similarly, E is said to be *finitary* if for every pair of terms (s,t) the set $\mu\mathrm{CSU}_\mathrm{E}(s,t)$ exists and is finite.

Every finitary equational theory E gives rise to the following E-*unification problem*: given two terms s and t, produce a (minimal) complete set $\mu\mathrm{CSU}_\mathrm{E}(s,t)$ of E-unifiers of s and t. The E-*matching problem* is the restriction of the E-unification problem to terms s and t such that t is a ground term. We write $s \doteq_\mathrm{E} t$ to denote an instance of the E-unification problem; this way we differentiate an instance of the E-unification problem from an E-equality $s =_\mathrm{E} t$. If E is the empty theory, then we speak about the *syntactic* unification problem and the *syntactic* matching problem, and we write $s \doteq t$.

2.3 Unification in Abelian groups and Boolean rings

Let $\mathbf{G} = (G,+,-,e)$ be an algebraic structure such that $+$ is a binary operation on the carrier G of $\mathbf{G}$, $-$ is a unary operation on G, and e is an element of G. We say that $\mathbf{G} = (G,+,-,e)$ is an *Abelian group* if it satisfies the following equational axioms AG:

$$\begin{array}{cc} x+e = x & x+y = y+x \\ x+(-x) = e & (x+y)+z = x+(y+z). \end{array}$$

It is important to note that AG-unification is equivalent to AG-matching, since every AG-unification problem $s \doteq_\mathrm{AG} t$ is equivalent to $s+(-t) \doteq_\mathrm{AG} e$.

Let E be an arbitrary equational theory. In the case of general E-unification, there is no difference between a single equation $s \doteq_\mathrm{E} t$ and a system of equations $\{s_1 \doteq_\mathrm{E} t_1, \ldots, s_n \doteq_\mathrm{E} t_n\}$, since the E-unifiers of $\{s_1 \doteq_\mathrm{E} t_1, \ldots, s_n \doteq_\mathrm{E} t_n\}$ coincide with the E-unifiers of the equation $f(s_1,\ldots,s_n) \doteq_\mathrm{E} f(t_1,\ldots,t_n)$, where f is a free function symbol in $\mathcal{F} \setminus \mathit{sig}(\mathrm{E})$. In contrast, there are equational theories E for which in the case of elementary E-unification or in the case of E-unification with constants there are computational differences between single equations and systems of equations (cf. [BS94,HK95b]). Note that systems of AG-unification problems with constants are not always equivalent to single AG-unification problems. Nevertheless, we can take advantage of the Abelian group axioms and bring such systems into a special form. We replace n occurrences

of the term t in $t + \cdots + t$ by the expression nt; we also replace k occurrences of t in $(-t) + \cdots + (-t)$ by the expression $-kt$. Thus, every system of AG-unification problems with constants can be brought into the form $A\mathbf{x} = \Gamma\mathbf{c}$, where $A = \left(\alpha_i^j\right)_k^m$ and $\Gamma = \left(\gamma_i^j\right)_k^n$ are integer matrices, $\mathbf{x} = (x_1, \ldots, x_m)$ is a vector of formal variables, and $\mathbf{c} = (c_1, \ldots, c_n)$ is a vector of free constants. It follows that every system of AG-unification problems with constants can be transformed to a system of linear Diophantine equations that must be solved over the integers (positive, negative, or zero). The solution of the latter is computed as the Hermite normal form of the corresponding integer matrix. The Hermite normal form yields a general parametric expression for all solutions; moreover, this expression is unique up to a linear combination. As a result, AG-unification with constant is unitary (cf. [BS94]).

Let $\mathbf{B} = (B, \oplus, \wedge, 0, 1)$ be an algebraic structure such that $\oplus$ (exclusive or) and $\wedge$ (conjunction) are binary operations on the carrier B of $\mathbf{B}$, and 0 (false) and 1 (true) are elements of B. We say that $\mathbf{B} = (B, \oplus, \wedge, 0, 1)$ is a *Boolean ring* if it satisfies the following equational axioms BR:

$$\begin{array}{cc}
x \oplus 0 = x & x \oplus y = y \oplus x \\
x \oplus x = 0 & (x \oplus y) \oplus z = x \oplus (y \oplus z) \\
x \wedge 0 = 0 & x \wedge y = y \wedge x \\
x \wedge 1 = x & (x \wedge y) \wedge z = x \wedge (y \wedge z) \\
x \wedge x = x & x \wedge (y \oplus z) = (x \wedge y) \oplus (x \wedge z).
\end{array}$$

BR-unification is equivalent to BR-matching, since every BR-unification problem $s \doteq_{\mathrm{BR}} t$ is equivalent to $s \oplus t \doteq_{\mathrm{BR}} 0$. Moreover, every BR-unification problem $s \doteq_{\mathrm{BR}} 0$ is equivalent to $s \oplus 1 \doteq_{\mathrm{BR}} 1$; therefore, it makes no difference whether we consider a problem $s \doteq_{\mathrm{BR}} 0$ or $s \doteq_{\mathrm{BR}} 1$. When it comes to BR-unification with constants, there is no difference between a single equation and a system of equations, since every system of BR-unification problems $s_1 \doteq_{\mathrm{BR}} 1, \ldots, s_n \doteq_{\mathrm{BR}} 1$ can be transformed to the equivalent problem $s_1 \wedge \cdots \wedge s_n \doteq_{\mathrm{BR}} 1$.

Martin and Nipkow [MN89] showed that BR-unification with constants is unitary. This follows from Löwenheim's theorem, which provides a way to obtain the most general BR-unifier from any particular BR-unifier. Indeed, let $\mathcal{F}$ be a signature consisting of $\oplus$, $\wedge$, 0, 1, and free constant symbols, and let t be a term over $\mathcal{F}$ whose variables are $x_1, \ldots, x_n$. Löwenheim's theorem implies that if the substitution $x_i \mapsto b_i$, $1 \leq i \leq n$, is a BR-unifier of $t \doteq_{\mathrm{BR}} 0$, then the substitution $x_i \mapsto x_i \oplus (t \wedge (x_i \oplus b_i))$, $1 \leq i \leq n$, is the only element of $\mu\mathrm{CSU}_{\mathrm{BR}}(t, 0)$.

2.4 Combination algorithm for equational unification

Let $\mathrm{Th}(\mathcal{F}_1, \mathrm{E}_1)$ and $\mathrm{Th}(\mathcal{F}_2, \mathrm{E}_2)$ be finitary equational theories with disjoint signatures. Baader and Schulz [BS92] presented an algorithm for unification in the combined theory $\mathrm{Th}(\mathcal{F}_1 \cup \mathcal{F}_2, \mathrm{E}_1 \cup \mathrm{E}_2)$, under the assumption that the *unification problem with linear constant restrictions* is solvable for each of the theories $\mathrm{Th}(\mathcal{F}_1, \mathrm{E}_1)$ and $\mathrm{Th}(\mathcal{F}_2, \mathrm{E}_2)$. If E is an equational theory and P is an E-unification problem, then a *linear constant restriction* of P is a linear ordering $\prec$ on a finite set V of variables and a finite set C of free constants (i.e., the constants in

C are not members of $sig(\mathrm{E})$). A *solution* of an E-unification problem P *with linear constant restriction* is an E-unifier σ of P with the property that if $c \in C$ and $x \in V$ are such that $x \prec c$, then c does not occur in $x\sigma$. It is known that there are algorithms for both AG-unification with linear constant restriction and BR-unification with linear constant restriction (cf. [SS89,BS92]).

Assume that A_i is an algorithm for the E_i-unification problem with linear constant restriction, $i = 1, 2$. Baader and Schulz [BS92] give an algorithm A for unification in the combined theory $\mathrm{Th}(\mathcal{F}_1 \cup \mathcal{F}_2, \mathrm{E}_1 \cup \mathrm{E}_2)$ that uses the two algorithms A_1 and A_2 as subroutines. The crucial part of this combination algorithm A is a decomposition algorithm that takes as input an a system P of elementary $\mathrm{E}_1 \cup \mathrm{E}_2$-unification problems and, after several (possibly non-deterministic) steps, transforms this system into separate E_1-unification problems and E_2-unification problems. Before outlining the combination algorithm, several auxiliary concepts have to be introduced. The elements of the signature $\mathcal{F}_1$ are called 1-*symbols* and the elements of $\mathcal{F}_2$ are called 2-*symbols*. If a term t is of the form $f(t_1, \ldots, t_n)$ and f is an i-symbol, then we say that t is an i-*term*. A subterm s of an i-term t is called an *alien subterm* of t if it is a j-term, $j \neq i$, such that every proper superterm of s in t is an i-term. An i-term is *pure* if it contains only i-symbols and variables. A *pure* i-equation, $i = 1, 2$, is an equation $s \doteq_{\mathrm{E}} t$ such that s and t are pure i-terms. An equation $s \doteq_{\mathrm{E}} t$ is *pure* if it is 1-pure or 2-pure.

The main steps of the combination algorithm A are as follows:

Variable abstraction: Successively replace all alien subterms by new variables until all terms in P are pure. This means that every equation $s \doteq_{\mathrm{E}} t$, where s contains an alien subterm s_1, is replaced by two equations $s' \doteq_{\mathrm{E}} t$ and $x \doteq_{\mathrm{E}} s_1$, where s' is the term obtained from s by replacing s_1 by x.

Impure equation split: Replace each impure equation $s \doteq_{\mathrm{E}} t$ by two new equations $x \doteq_{\mathrm{E}} s$ and $x \doteq_{\mathrm{E}} t$, where x is a new variable. After this step has been carried out, the resulting system contains pure equations only.

Variable identification: In a non-deterministic way, choose a partition of the set of variables occurring in the system obtained in the previous step. For each equivalence class of this partition, choose a variable as canonical representative of the class and replace in the system all occurrences of other variables in the class by its canonical representative.

Variable ordering and labelling: In a non-deterministic way, choose a linear ordering $\prec$ on the variables of the system and assign label 1 or label 2 to each of these variables.

Split of the problem: Split the system into two systems P_1 and P_2, where the P_1 contains all 1-equations and P_2 contains all 2-equations. Only the i-variables are considered as variables in the system P_i, whereas the j-variables in P_i, with $i \neq j$, are treated as constants. For $i = 1, 2$, use algorithm A_i to solve the E_i-unification problem P_i with the linear constant restriction induced by the linear ordering of the previous step. If both P_1 and P_2 are solvable, combine the complete sets U_1 and U_2 returned by A_1 and A_2 to obtain a solution to the original system.

Regrouping: The complete set of unifiers for the original $E_1 \cup E_2$-unification problem P is the union of the solutions of all systems generated by all possible choices in the earlier non-deterministic steps.

Note that if both equational theories E_1 and E_2 are finitary, then the combination algorithm A computes a finite complete set of unifiers for every unification problem in the theory $E_1 \cup E_2$, since every nondeterministic choice is done from a finite set. This implies that the combination $E_1 \cup E_2$ of two finitary theories E_1 and E_2 is also finitary, assuming that E_i-unification with linear constant restriction is solvable, $i = 1, 2$. On the other hand, if both E_1 and E_2 are unitary theories, then the combination algorithm may compute a complete set of unifiers with more than one element, since the combination algorithm consists of several non-deterministic steps. This does not necessarily mean that the equational theory $E_1 \cup E_2$ is not unitary. Indeed, assume that E_1 and E_2 are empty theories with finite disjoint signatures. It is obvious that the empty theory $E_1 \cup E_2$ is unitary, but the combination algorithm may produce a complete set of unifiers with more than one element, due to the non-deterministic choices.

3 Unification with constants vs. general unification

In this section, we derive inherent lower bounds for the running time of all combination algorithms for equational unification. More precisely, we show that, unless #P is contained in FP, there does not exist a polynomial-time combination algorithm for $E_1 \cup E_2$-unification with oracles for the E_1-unification problem and the E_2-unification problem. This result is obtained by analyzing the complexity of AG-unification with constants and the complexity of the counting problem for general AG-unification. As stated earlier, AG-unification with constants is a unitary theory. Baader and Siekman [BS94] pointed out that the most general unifier for AG-unification with constants can be computed in polynomial time. This is based on a transformation of the AG-unification problems with constants to an equivalent linear Diophantine system of equations that must be solved over the integers, followed by the computation of the Hermite normal form of the corresponding integer matrix.

Proposition 1. AG*-unification with constants is solvable in polynomial time.*

Proof. Every system of AG-unification problems with constants $A\mathbf{x} = \Gamma\mathbf{c}$ can be transformed to an equivalent linear Diophantine system over the integers as follows. Assume that every formal variable x_i gets assigned y_i^j copies of the constant c_j, and z_i^j copies of a residual term u_j, $1 \leq j \leq n$. Therefore, we write

$$x_i = y_i^1 c_1 + \cdots + y_i^n c_n + z_i^1 u_1 + \cdots + z_i^n u_n$$

where y_i^j, z_i^j are integer variables, c_j are free constants, and u_j are formal variables (one for each free constant) representing residual terms that are cancelled to the neutral element e in the original system $A\mathbf{x} = \Gamma\mathbf{c}$. After substitution and regrouping, we obtain two linear Diophantine systems $AZ = \mathbf{0}$ and $AY = \Gamma$ over

the integers, where $Y = (y_i^j)_m^n$ and $Z = (z_i^j)_m^n$ are matrices of integer variables. The first system is derived from the equations $A(Z\mathbf{u}) = \mathbf{e}$, where $\mathbf{e} = (e, \ldots, e)$ is a vector of neutral elements, expressing the fact that the residual terms u_j are cancelled to the neutral element e in the original system. It is clear that this transformation can be carried out in polynomial time. Since the unique integer solutions of the linear Diophantine systems $AZ = \mathbf{0}$ and $AY = \Gamma$ can be computed in polynomial time (cf. [KB79]), it follows that the unique solution of each AG-unification problem with constants can be computed in polynomial time. □

We now introduce the counting problems for general AG-unification and general BR-unification.

#General AG-Unification
Input: A set $\mathcal{F}$ of free constant and function symbols, and two terms $s, t \in \mathcal{T}(sig(\mathrm{AG}) \cup \mathcal{F}, \mathcal{X})$.
Output: Cardinality of the set $\mu\mathrm{CSU}_{\mathrm{AG}}(s, t)$.

#General BR-Unification
Input: A set $\mathcal{F}$ of free function and constant symbols, and two terms $s, t \in \mathcal{T}(sig(\mathrm{BR}) \cup \mathcal{F}, \mathcal{X})$.
Output: Cardinality of the set $\mu\mathrm{CSU}_{\mathrm{BR}}(s, t)$.

The following result yields a lower bound for the computational complexity of the counting problem for general AG-unification and general BR-unification.

Proposition 2. *The counting problems* #General AG-Unification *and* #General BR-Unification *are both* #P*-hard.*

Proof. We give a parsimonious reduction from #Perfect Matchings that works for both #General AG-unification and #General BR-unification. In [HK95a], we used the same reduction to show that #AC1-matching is #P-hard, where AC1-matching is the restriction of AC-matching to linear terms. It should be noted, however, that the proof of correctness we give here is substantially different than the proof for #AC1-matching; actually, in what follows the combination algorithm for equational unification is used in a crucial way, while the proof for #AC1-matching made no use of the combination algorithm.

Suppose that we are given a bipartite graph $G = (S, T, E)$ with $2n$ nodes, where the sets $S = \{s_1, \ldots, s_n\}$ and $T = \{t_1, \ldots, t_n\}$ form the partition of the nodes. Let a be a constant symbol, f a unary function symbol, and g a (n+1)-ary function symbol. We also consider two disjoint sets of variables $X = \{x_{ij} \mid i, j = 1, \ldots, n\}$ and $Y = \{y_1, \ldots, y_n\}$. With each node s_i in the set S we associate the term $s_i^* = g(s_i^1, \ldots, s_i^n, s_i^{n+1})$, where

$$s_i^j = \begin{cases} f(x_{ii}) & \text{if } 1 \leq i, j \leq n \text{ and } i = j \\ x_{ij} & \text{if } 1 \leq i, j \leq n \text{ and } i \neq j \\ y_i & \text{if } 1 \leq i \leq n \text{ and } j = n + 1 \end{cases}$$

Intuitively, we view the nodes $s_1, \ldots, s_n$ in S as vectors of a "matrix":

$$\begin{aligned} s_1^* &= g(f(x_{11}), x_{12}, x_{13}, \ldots, x_{1n}, y_1) \\ s_2^* &= g(x_{21}, f(x_{22}), x_{23}, \ldots, x_{2n}, y_2) \\ &\vdots \\ s_n^* &= g(x_{n1}, x_{n2}, \ldots, x_{n,n-1}, f(x_{nn}), y_n) \end{aligned}$$

in which the subterms $f(x_{ii})$ occupy the main diagonal, while the variables $y_1, \ldots, y_n$ are along the last column. Next, with each node t_i in T we associate the ground term $t_i^* = g(t_i^1, \ldots, t_i^n, t_i^{n+1})$, where

$$t_i^j = \begin{cases} f(a) & \text{if } 1 \leq i, j \leq n \text{ and } (s_j, t_i) \in E \\ a & \text{if } 1 \leq i, j \leq n \text{ and } (s_j, t_i) \notin E \\ f^i(a) & \text{if } j = n+1 \end{cases}$$

Thus, we view the nodes $t_1, \ldots, t_n$ in T as vectors of another "matrix"

$$\begin{aligned} t_1^* &= g(t_1^1, \ldots, t_1^n, f(a)) \\ t_2^* &= g(t_2^1, \ldots, t_2^n, f^2(a)) \\ &\vdots \\ t_n^* &= g(t_n^1, \ldots, t_n^n, f^n(a)). \end{aligned}$$

The intuition is that the second matrix represents the adjacency matrix (extended by the column $f^k(a)$, $1 \leq k \leq n$) of the edge relation E of the graph G, where the terms $f(a)$ and a are used to encode the presence and the absence, respectively, of an edge between two nodes. Note that the terms s_i^*, $i \leq n$, are linear and have pairwise disjoint variables, while the terms t_i^*, $i \leq n$, are ground.

Consider now the E-unification problem $s_1^* \circ \cdots \circ s_n^* \doteq_{\mathrm{E}} t_1^* \circ \cdots \circ t_n^*$, where, if $\mathrm{E} = \mathrm{AG}$, the symbol $\circ$ stands for $+$, while, if $\mathrm{E} = \mathrm{BR}$, it stands for $\wedge$. This problem can be viewed as a $\mathrm{E}_1 \cup \mathrm{E}_2$-unification problem, where $\mathrm{E}_1 \in \{\mathrm{AG}, \mathrm{BR}\}$ and E_2 is the empty theory. Thus, the problem $s_1^* \circ \cdots \circ s_n^* \doteq_{\mathrm{E}} t_1^* \circ \cdots \circ t_n^*$ can be solved by applying to it the combination algorithm with the algorithm for E_1-unification with constants and the algorithm for syntactic unification algorithm as subroutines. The variable abstraction transforms it to the system

$$\{u_1 \circ \cdots \circ u_n \doteq_{\mathrm{E}} v_1 \circ \cdots \circ v_n, \quad u_i \doteq_{\mathrm{E}} s_i^*, \quad v_j \doteq_{\mathrm{E}} t_j^* \mid i, j = 1, \ldots, n\}$$

where u_i, v_j are new variables. Every equation is pure so we do not need to split them. We cannot identify two variables v_i and v_j, where $i \neq j$, since the equality $v_i = v_j$ implies the equality $t_i^* = t_j^*$, which is evidently incorrect because of the different subterms in the last column $t_i^{n+1} = f^i(a)$ and $t_j^{n+1} = f^j(a)$. Hence, every variable v_j can be either identified with a variable u_i or it can form a singleton equivalence class $[v_j]$. Identifying the variables X and Y is not necessary, their value will be determined later. It is also not necessary to choose a linear ordering $\prec$ on the variables, since every choice of the ordering is correct.

Indeed, the terms s_i^* are linear and have pairwise disjoint variables, whereas the terms t_j^* are ground; therefore, no variable cycles can occur.

Note that $u_i \doteq_{\mathrm{E}} s_i^*$ and $v_j \doteq_{\mathrm{E}} t_j^*$ are 2-equations, and s_i^*, t_j^* are 2-terms, for $1 \leq i, j \leq n$. Therefore the variables u_i, v_j must be labelled as 2-variables, otherwise none of the 2-equations $u_i \doteq s_i^*$ and $v_j \doteq t_j^*$ would have a solution. Since $u_1 \circ \cdots \circ u_n \doteq_{\mathrm{E}} v_1 \circ \cdots \circ v_n$ is a 1-equation, the 2-variables u_i, v_j are considered here as constants. Since no "constant" appears twice among $v_1, \ldots, v_n$, the axioms $(x + (-x) = e)$ and $(x \wedge x = x)$ are not used in the equivalence proof of the Skolemized terms $u_1 \circ \cdots \circ u_n$ and $v_1 \circ \cdots \circ v_n$. Only the associativity and commutativity of the symbol $\circ \in \{+, \wedge\}$ is used in the equivalence proof, therefore the vector of "constants" $(u_1, \ldots, u_n)$ must be a permutation of the vector $(v_1, \ldots, v_n)$. This implies that the only variable identifications that generate a solution for $u_1 \circ \cdots \circ u_n \doteq_{\mathrm{E}} v_1 \circ \cdots \circ v_n$ are the ones for which every class in the partition of $\{u_i, v_j \mid i, j = 1, \ldots, n\}$ consists of two variables, one u_i and the other v_j, for some $i, j \in \{1, \ldots, n\}$. No partition with singleton equivalence classes $[u_i]$ or $[v_j]$ gives a solution for the aforementioned 1-equation. Its solution is always the identity $\top$.

Consider now the 2-equations $u_i \doteq s_i^*$ and $v_j \doteq t_j^*$, $1 \leq i, j \leq n$. If the variables u_i and v_j are identified within a class $[u_i, v_j]$ of the partition, then the syntactic unification algorithm merges the respective equations to $s_i^* \doteq t_j^*$. We claim that for each i and j, $1 \leq i, j \leq n$, there is an edge $(s_i, t_j) \in E$ if and only if the terms s_i^* and t_j^* are unifiable in the empty theory. Indeed, if $(s_i, t_j) \in E$, then by the above construction we have that $t_j^i = f(a)$ and $s_i^i = f(x_{ii})$. Since $s_i^k = x_{ik}$ for $1 \leq i \neq k \leq n$ and $s_i^{n+1} = y_i$, we have that the terms s_i^* and t_j^* are unifiable via the most general unifier

$$\sigma_{ij} = [x_{i1} \mapsto t_j^1, \ldots, x_{i,i-1} \mapsto t_j^{i-1},\ x_{ii} \mapsto a,\ x_{i,i+1} \mapsto t_j^{i+1}, \ldots, x_{in} \mapsto t_j^n,\ y_i \mapsto f^j(a)].$$

Conversely, if $(s_i, t_j) \notin E$, then $t_j^i = a$ and $s_i^i = f(x_{ii})$. Consequently, the terms s_i^* and t_j^* are not unifiable. Observe that for each pair of terms s_i^* and t_j^* the unifier σ_{ij} is unique, because of the forced substitution $y_i \mapsto f^j(a)$. As a result, every perfect matching $E' \subseteq E$ of the graph G gives rise to the unifier

$$\sigma = \bigcup_{(i,j) \in E'} \sigma_{ij}$$

of the system $P_2 = \{u_i \doteq s_i^*,\ v_j \doteq t_j^* \mid 1 \leq i, j \leq n\}$, provided that the partition identified the variables u_i and v_j if and only if there exists and edge $(s_i, t_j) \in E$. Moreover, the uniqueness of each substitution σ_{ij} implies the uniqueness of the unifier σ, since the term $s_1^* \circ \cdots \circ s_n^*$ is linear and the term $t_1^* \circ \cdots \circ t_n^*$ is ground. It follows that each partition of the variables $\{u_i, v_j \mid 1 \leq i, j \leq n\}$ that encodes a perfect matching of the graph G corresponds to one unifier of the system P_2. Conversely, if the partition identified the variables u_i and v_j such that $(s_i, t_j) \notin E$, then the system P_2 has no solution.

The regrouping step returns the complete set U of unifiers σ that corresponds to the perfect matchings of the graph G, since the solution of the Skolemized equation $u_1 \circ \cdots \circ u_n \doteq_{\mathrm{E}} v_1 \circ \cdots \circ v_n$ is the identity $\top$. Note that every substitution in the complete set of unifiers U is ground. Assume that the set U is not minimal, i.e., that there exists two unifiers $\sigma, \rho \in U$ and a substitution η, such that $\sigma\eta =_{\mathrm{E}}^{V} \rho$ where $V = \mathcal{D}om(\sigma) = \mathcal{D}om(\rho)$ is the set of variables occurring in the E-unification problem. The fact that the substitution σ is ground implies that the identity $\sigma\eta = \sigma$ holds for every substitution η with $\mathcal{D}om(\eta) \subseteq V$. Since the substitutions σ and ρ have the same domains V and since for all variables $x \in V$ the instances $x\sigma$ and $x\rho$ contain free symbols only, we have that $\sigma =_{\mathrm{E}}^{V} \rho$ implies $\sigma = \rho$. Hence, the computed complete set of unifiers is minimal. This concludes the construction of a parsimonious reduction from #Perfect Matchings to #General AG-unification and to #General BR-Unification. □

By Proposition 1, AG-unification with constants is solvable in polynomial time. Moreover, it is well known that the same holds true for syntactic unification. Since general AG-unification is the combination of AG-unification with constants and syntactic unification, Proposition 2 implies now immediately the main result of this paper.

Theorem 3. *Unless* #P *is contained in* FP*, there does not exist a combination algorithm A for $\mathrm{E}_1 \cup \mathrm{E}_2$-unification, where E_1 and E_2 are disjoint equational theories, such that A runs in polynomial time using oracles for the E_1-unification problem and the E_2-unification problem.*

We conclude this section with some comments on the proof of Proposition 2. First, as a byproduct of this proof, we see that general AG-unification and general BR-unification are not unitary, since there exist bipartite graphs with more than one perfect matching. Note also that this proof makes use of three free (uninterpreted) symbols that are not present in the signature of the Boolean ring axioms BR, namely the constant a, the unary function symbol f, and the $(n+1)$-ary symbol g. Using a more complicated proof, we can reduce the number of the free symbols and their arity. Clearly, the constant a is not necessary and can be replaced by the Boolean constant 0. This results in some additional variable abstraction steps. The unary symbol f can be replaced by the negation operator $\neg$ on the main diagonal of the "matrix" S and at the positions in T encoding the edges $(s_j, t_i) \in E$, but it cannot be eliminated from the last column where it serves to distinguish the terms t_j^*. The ground term $\neg 0$ can be replaced by the equivalent Boolean value 1. The $(n+1)$-ary symbol g can be replaced by an iteration of the exclusive-or connective $\oplus$, provided that we apply the unary symbol f to the subterms as an index to express their previous positions under the symbol g. Hence, after these transformations have been carried out, we have the new terms $s_i^* = s_i^1 \oplus \cdots \oplus s_i^n \oplus s_i^{n+1}$, where

$$s_i^j = \begin{cases} f^j(\neg x_{ii}) & \text{if } 1 \le i, j \le n \text{ and } i = j \\ f^j(x_{ij}) & \text{if } 1 \le i, j \le n \text{ and } i \ne j \\ f^{n+1}(y_i) & \text{if } 1 \le i \le n \text{ and } j = n+1 \end{cases}$$

and $t_i^* = t_i^1 \oplus \cdots \oplus t_i^n \oplus t_i^{n+1}$ where

$$t_i^j = \begin{cases} f^j(1) & \text{if } 1 \leq i,j \leq n \text{ and } (s_j, t_i) \in E \\ f^j(0) & \text{if } 1 \leq i,j \leq n \text{ and } (s_j, t_i) \notin E \\ f^{n+1+i}(0) & \text{if } j = n+1 \end{cases}$$

The unification problem $s_1^* \wedge \cdots \wedge s_n^* \doteq_{\mathrm{BR}} t_1^* \wedge \cdots \wedge t_n^*$ encodes, as before, the problem of finding all perfect matchings in the bipartite graph G. Thus, the counting problem #General BR-Unification is #P-hard, even in the presence of a single free unary function symbol.

4 Concluding remarks

We proved that there is no polynomial-time general combination algorithm for unification in finitary equational theories, unless the counting class #P collapses to the class FP of function problems solvable in polynomial-time. Such a collapse is considered highly unlikely for a number of reasons; in particular, as mentioned earlier, the collapse of #P to FP implies that P = NP. As a matter of fact, the prevalent view in complexity theory is that #P is contained neither in $\mathrm{FP}^{\mathrm{NP}}$ nor in any other level of the functional polynomial hierarchy FPH. Under the hypothesis that #P is not contained in FPH (which is also widely believed to be true), our results imply a stronger lower bound on the performance of all general combination algorithms, namely that no such algorithm can be found in the class FPH.

We end by describing two open problems that are motivated from the work reported here. Note that the equational theory AG of Abelian groups is collapsing (axiom $x + e = x$) and non-regular (axiom $x + (-x) = e$). Are there equational theories E that are regular, or non-collapsing, or both regular and non-collapsing, and such that a similar gap in computational complexity exists between E-unification with constants and the counting problem for general E-unification? We conjecture that such equational theories exist, although none of the well-studied ones, e.g. AC, appears to be such a candidate. Finally, can a lower bound on the performance of all combinations algorithms be derived without appealing to any complexity-theoretic hypotheses? In other words, is there an equational theory E for which the gap in computational complexity between E-unification with constants and the counting problem for general E-unification can be enlarged to two provably different complexity classes?

Acknowledgement We thank Christophe Ringeissen for discussions on the combination problem for finitary unification.

References

[BJSS89] A. Boudet, J.-P. Jouannaud, and M. Schmidt-Schauß. Unification in Boolean rings and Abelian groups. *Journal of Symbolic Computation*, 8:449–477, 1989.

[Bou93] A. Boudet. Combining unification algorithms. *Journal of Symbolic Computation*, 16:597–626, 1993.

[BS92] F. Baader and K. Schulz. Unification in the union of disjoint equational theories: combining decision procedures. In D. Kapur, editor, *Proceedings 11th International Conference on Automated Deduction (CADE'92), Saratoga Springs (New York, USA)*, volume 607 of *Lecture Notes in Computer Science (in Artificial Intelligence)*, pages 50–65. Springer-Verlag, June 1992.

[BS94] F. Baader and J.H. Siekmann. Unification theory. In D.M. Gabbay, C.J. Hogger, and J.A. Robinson, editors, *Handbook of Logic in Artificial Intelligence and Logic Programming*, volume 2: Deduction Methodologies, pages 41–125. Oxford University Press, Oxford (UK), 1994.

[Coo71] S.A. Cook. The complexity of theorem-proving procedures. In *Proceedings 3rd Symposium on Theory of Computing (STOC'71), Shaker Heights (Ohio, USA)*, pages 151–158. Association for Computing Machinery, May 1971.

[DKR94] E. Domenjoud, F. Klay, and Ch. Ringeissen. Combination techniques for non-disjoint equational theories. In A. Bundy, editor, *Proceedings 12th International Conference on Automated Deduction (CADE'94), Nancy (France)*, volume 814 of *Lecture Notes in Computer Science (in Artificial Intelligence)*, pages 267–281. Springer-Verlag, 1994.

[Fag87] F. Fages. Associative commutative unification. *Journal of Symbolic Computation*, 3(3):257–275, 1987.

[FH86] F. Fages and G. Huet. Complete sets of unifiers and matchers in equational theories. *Theoretical Computer Science*, 43(1):189–200, 1986.

[Her86] A. Herold. Combinations of unification algorithms. In J.H. Siekmann, editor, *Proceedings 8th International Conference on Automated Deduction (CADE'86), Oxford (England)*, volume 230 of *Lecture Notes in Computer Science*, pages 450–469. Springer-Verlag, 1986.

[HK95a] M. Hermann and P.G. Kolaitis. The complexity of counting problems in equational matching. *Journal of Symbolic Computation*, 20(3):343–362, 1995.

[HK95b] M. Hermann and P.G. Kolaitis. Computational complexity of simultaneous elementary matching problems. In J. Wiedermann and P. Hájek, editors, *Proceedings 20th International Symposium on Mathematical Foundations of Computer Science (MFCS'95), Prague (Czech Republic)*, volume 969 of *Lecture Notes in Computer Science*, pages 359–370. Springer-Verlag, August 1995.

[HS87] A. Herold and J.H. Siekmann. Unification in Abelian semigroups. *Journal of Automated Reasoning*, 3:247–283, 1987.

[JK91] J.-P. Jouannaud and C. Kirchner. Solving equations in abstract algebras: A rule-based survey of unification. In J.-L. Lassez and G. Plotkin, editors, *Computational Logic. Essays in honor of Alan Robinson*, chapter 8, pages 257–321. MIT Press, Cambridge (MA, USA), 1991.

[Joh90] D.S. Johnson. A catalog of complexity classes. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science, Volume A: Algorithms and Complexity*, chapter 2, pages 67–161. North-Holland, Amsterdam, 1990.

[KB79] R. Kannan and A. Bachem. Algorithms for computing the Smith and Hermite normal forms of an integer matrix. *SIAM Journal on Computing*, 8(4):499–507, 1979.

[Kir85] C. Kirchner. Méthodes et outils de conception systématique d'algorithmes d'unification dans les théories équationnelles. Thèse d'Etat, Université de Nancy 1, France, 1985.

[KR94] H. Kirchner and Ch. Ringeissen. Combining symbolic constraint solvers on algebraic domains. *Journal of Symbolic Computation*, 18(2):113–155, 1994.

[MN89] U. Martin and T. Nipkow. Boolean unification — the story so far. *Journal of Symbolic Computation*, 7(3 & 4):275–294, 1989.

[Pap94] C.H. Papadimitriou. *Computational complexity*. Addison-Wesley, 1994.

[Plo72] G.D. Plotkin. Building-in equational theories. In B. Meltzer and D. Mitchie, editors, *Machine Intelligence*, volume 7, pages 73–90. Edinburgh University Press, Edinburgh, UK, 1972.

[SS89] M. Schmidt-Schauß. Unification in a combination of arbitrary disjoint equational theories. *Journal of Symbolic Computation*, 8:51–99, 1989.

[Sti81] M. Stickel. A unification algorithm for associative-commutative functions. *Journal of the Association for Computing Machinery*, 28(3):423–434, 1981.

[Tid86] E. Tidén. Unification in combinations of collapse-free theories with disjoint sets of function symbols. In J.H. Siekmann, editor, *Proceedings 8th International Conference on Automated Deduction (CADE'86), Oxford (England)*, volume 230 of *Lecture Notes in Computer Science*, pages 431–449. Springer-Verlag, 1986.

[Tod89] S. Toda. On the computational power of PP and $\oplus$P. In *Proceedings 30th IEEE Symposium on Foundations of Computer Science (FOCS'89), Research Triangle Park (North Carolina, USA)*, pages 514–519, 1989.

[Val79a] L.G. Valiant. The complexity of computing the permanent. *Theoretical Computer Science*, 8(2):189–201, 1979.

[Val79b] L.G. Valiant. The complexity of enumeration and reliability problems. *SIAM Journal on Computing*, 8(3):410–421, 1979.

[Yel87] K. Yelick. Unification in combinations of collapse-free regular theories. *Journal of Symbolic Computation*, 3(1 & 2):153–182, 1987.