Combining Constraint Solving

Franz Baader[1] & Klaus U. Schulz[2]

[1] Theoretical Computer Science, RWTH Aachen, Germany
E-mail: baader@informatik.rwth-aachen.de
[2] CIS, University of Munich, Germany
E-mail: schulz@cis.uni-muenchen.de

1 Introduction

In many areas of Logic, Computer Science, and Artificial Intelligence, there is a need for specialized formalisms and inference mechanisms to solve domain-specific tasks. For this reason, various methods and systems have been developed that allow for an efficient and adequate treatment of such restricted problems. In most realistic applications, however, one is faced with a complex combination of different problems, which means that a system tailored to solving a single problem can only be applied if it is possible to combine it both with other specialized systems and with general purpose systems.

This general problem of combining systems can be addressed on various conceptual levels. At one end, combination of logical systems is studied with an emphasis on formal properties, using tools from mathematics and logics. Examples of results obtained on this level are transfer results for modal logics [43, 28] and modularity results for term rewriting systems [74, 52]. On the other end of the spectrum, the combination of software tools necessitates considering physical connections and appropriate communication languages [19, 44]. Between these two extremes lies the combination of constraint systems and the respective solvers, which is the topic of this paper. On the one hand, defining a semantics for the combined system may depend on methods and results from formal logic and universal algebra. On the other hand, an efficient combination of the actual constraint solvers often requires the possibility of communication and cooperation between the solvers.

Since there is a great variety of constraint systems and of approaches for combining them, we will start with a short classification of the different approaches. Subsequently, we will describe two of the most prominent combination approaches in this area:

- the Nelson-Oppen scheme [49, 54] for combining decision procedures for the validity of quantifier-free formulae in first-order theories, which was originally motivated by program verification;
- methods for combining E-unification algorithms [60, 15, 6] and more general constraint solvers [7], which are of interest in theorem proving, term rewriting, and constraint programming.

Our treatment of the Nelson-Oppen method can be seen as a warm-up exercise for the second approach since it is simpler both w.r.t. the actual combination algorithm and w.r.t. the (algebraic) tools required for proving its correctness. The problem of combining unification algorithms will be treated in more detail. First, we will describe the combination algorithm as introduced in [2, 6] and briefly sketch how to prove its correctness. We will then give a logical reformulation of the combination algorithm, and describe an algebraic approach for proving its correctness. This approach has the advantage that it can be generalized to constraints more general than the equational constraints of unification problems and to solution structures more general than the E-free algebras of unification problems. Finally, we will sketch approaches for optimizing the combination algorithm, and comment on principal limitations for optimizations.

2 Classification of constraint systems and combination approaches

Before classifying different approaches for combining constraint system, we must explain what we mean by constraint systems and constraint solvers. We will also consider two examples, which already introduce the different forms of constraint systems whose combination will be considered in more detail in this paper.

2.1 Constraint systems and solvers

Informally, a *constraint system* is given by a constraint language and a "semantics" for this language. The *constraint language* determines which formal expressions are considered to be admissible constraints, and the *semantics* provides us with a notion of constraint satisfaction: given an admissible constraint, it is uniquely determined by the semantics whether this constraint is satisfiable or not. In general, this does not mean that this question is also effectively decidable. In its most basic form, a *constraint solver* for a given constraint system is a procedure that decides satisfiability.

The constraint language To put these notions on a more formal footing, we assume that constraints are formulae of first-order predicate logic, and that the semantics is provided by the usual semantics of first-order logic. To be more precise, we consider an (at most countable) *signature* Σ consisting of function symbols and predicate symbols, and a (countably infinite) set V of (individual) variables, and build first-order terms, called *Σ-terms*, and first-order formulae, called *Σ-formulae*, from these ingredients in the usual way. Usually, a *Σ-constraint* is a Σ-formula $\varphi(v_1, \ldots, v_n)$ with free variables $v_1, \ldots, v_n$, and a solution of the constraint replaces the variables such that the resulting expression is "true" under the given semantics. If we are interested only in solvability of the constraints and not in actually computing a solution, we may also use closed formulae (e.g., the existential closure of the open formula) as constraints.

It should also be noted that a constraint language usually does not allow for all Σ-formulae, but only for a certain subclass of formulae, to be used as constraints. Thus, a constraint language is characterized by a signature Σ and a class of Σ-formulae, which may be open or closed.

The semantics Given such a constraint language, its semantics can be defined in two different ways: by a Σ-theory T or a Σ-structure $\mathcal{A}$. A Σ-theory is given by a set T of closed Σ-formulae, and a Σ-structure $\mathcal{A}$ is a Σ-interpretation, i.e., a nonempty set A, the domain of $\mathcal{A}$, together with an interpretation of the n-ary predicate (function) symbols as n-ary relations (functions) on A.

For a given Σ-structure $\mathcal{A}$, a *solution* of the Σ-constraint $\varphi(v_1, \ldots, v_n)$ in $\mathcal{A}$ is a mapping $\{v_1 \mapsto a_1, \ldots, v_n \mapsto a_n\}$ of the free variables of the constraint to elements of A such that $\mathcal{A} \models \varphi(a_1, \ldots, a_n)$, i.e., $\varphi(v_1, \ldots, v_n)$ is true in $\mathcal{A}$ under the evaluation $\{v_1 \mapsto a_1, \ldots, v_n \mapsto a_n\}$. The constraint $\varphi(v_1, \ldots, v_n)$ is *satisfiable* in $\mathcal{A}$ iff it has a solution in $\mathcal{A}$. This is equivalent to saying that its existential closure $\exists v_1. \cdots \exists v_n. \varphi(v_1, \ldots, v_n)$ is valid in $\mathcal{A}$.

For a given Σ-theory T, there are two different ways of defining satisfiability of constraints, depending on whether we want the constraints to be satisfiable (in the sense introduced above) in all models or in some model of the theory T. In the first case, which is the more usual case in constraint solving, the Σ-constraint $\varphi(v_1, \ldots, v_n)$ is *satisfiable* in (all models of) T iff its existential closure $\exists v_1. \cdots \exists v_n. \varphi(v_1, \ldots, v_n)$ is valid in T. The second case coincides with the usual definition of satisfiability of (open) formulae in predicate logic: the Σ-constraint $\varphi(v_1, \ldots, v_n)$ is *satisfiable* in (some model of) T iff its existential closure $\exists v_1. \cdots \exists v_n. \varphi(v_1, \ldots, v_n)$ is satisfiable in T, i.e., valid in some model of T. In both cases, one does not have a natural notion of solution since there is more than one solution structure involved, though there may be specific instances where such a notion can be defined.

To sum up, we can define satisfiability of a constraint in three different ways: as validity of its existential closure in (i) a fixed solution structure $\mathcal{A}$; (ii) all models of a fixed theory T; (iii) some model of a fixed theory T.

Note that (i) is a special case of (ii) since we can take as theory T the theory of $\mathcal{A}$, i.e., the set of all Σ-formulae valid in $\mathcal{A}$. In general, (ii) is not a special case of (i). This is the case, however, if there exists a Σ-structure $\mathcal{A}$ that is *canonical* for T and the constraint language in the sense that a constraint is satisfiable in T iff it is satisfiable in $\mathcal{A}$.

The constraint solver Given a constraint language and a semantics, a constraint solver is a procedure that is able to decide satisfiability of the constraints. In this paper, we will mostly restrict our attention to the combination of such decision procedures. It should be noted, however, that in many cases constraint solvers produce more than just the answer "yes" or "no".

If there is the notion of a solution available, one may also want to have a solver that not only decides satisfiability, but also computes a solution, if one exists. Since a given constraint may have more than one solution one may even

be interested in obtaining a complete set of solutions, i.e., a set of solutions from which all solutions can be generated in a simple way. A prominent example of such a complete representation of all solutions is Robinson's most general unifier, from which all unifiers of a syntactic unification problem can be generated by instantiation.

Instead of actually computing a solution, the solver may transform the constraint into an equivalent one in "solved form." Such a solved form should, in some way, be simpler than the original constraint; in particular, the existence of a solved form should indicate satisfiability of the constraint, and it should be "easy" to read off an actual solution. The advantage of using solved forms is twofold. On the one hand, computing the solved form may be less complex than computing a solution. An example of this phenomenon is the so-called dag solved form of syntactic unification problems [33], which is linear in the size of the problem, whereas the most general unifier may be exponential in the size of the problem. On the other hand, a solver that computes a solved form is usually incremental: if the constraint is strengthened, then not all the work done during the satisfiability test needs to be re-done. To be more precise, assume that we have already tested φ for satisfiability, and then need to test $\varphi \wedge \psi$, a situation that frequently occurs, e.g., in constraint logic programming and theorem proving. If all we know after the satisfiability test is that φ is satisfiable, then we must start from scratch when testing $\varphi \wedge \psi$. However, if we have computed a solved form φ' of φ, then we can test $\varphi' \wedge \psi$ instead, which hopefully is easier.

2.2 More notation and two examples

As examples that illustrate the notions introduced above, we consider quantifier-free formulae and E-unification problems. Before introducing these types of constraint systems more formally, we define some subclasses of first-order formulae, which will be of interest later on.

We consider logic *with equality*, i.e., the binary predicate symbol $=$, which is interpreted as equality on the domain, is always available, without being explicitly contained in the signature. A Σ-atom is of the form $P(t_1, \ldots, t_n)$, where P is an n-ary predicate symbol of $\Sigma \cup \{=\}$ and $t_1, \ldots, t_n$ are Σ-terms. If the atom is of the form $t_1 = t_2$, i.e., P is the equality symbol $=$, then it is called an *equational atom*; otherwise, it is called a *relational atom*. Negated equational atoms are written $t_1 \neq t_2$ rather than $\neg(t_1 = t_2)$. A Σ-matrix is a Boolean combination of Σ-atoms, and a positive Σ-matrix is built from Σ-atoms using conjunction and disjunction only. A *positive Σ-formula* is a quantifier prefix followed by a positive Σ-matrix. The formula is called *positive existential (positive AE)* iff the quantifier prefix consists of existential quantifiers (universal quantifiers followed by existential quantifiers). A *universal formula* is given by a universal quantifier prefix followed by a quantifier-free formula. A universal formula is called *conjunctive universal* iff its matrix is a conjunctions of Σ-atoms and negated Σ-atoms. *Σ-sentences* (of either type) are Σ-formulae without free variables. Given a Σ-structure $\mathcal{A}$ (a Σ-theory T), the *positive theory* of $\mathcal{A}$ (T) consists of the set

of all positive Σ-sentences valid in $\mathcal{A}$ (T). The *positive existential, positive AE, universal, and conjunctive universal theories* of $\mathcal{A}$ (T) are defined analogously.

Quantifier-free formulae The Nelson-Oppen combination method [49, 54] applies to constraint systems of the following form:

- For a given signature Σ, the constraint language consists of all quantifier-free Σ-formulae, i.e., all Σ-matrices.
- The semantics is defined by an arbitrary Σ-theory T.
- One is interested in satisfiability in *some* model of T.

Thus, the constraint solver must be able to decide whether the existential closure of a quantifier-free Σ-formula is valid in some model of T. Since a formula is valid in some model of T iff its negation is not valid in all models of T, a decision procedure for the universal theory of T can be used as a constraint solver for this type of constraint systems.

Unification problems Unification modulo equational theories is a subfield of automated deduction which is very well-investigated (see [33, 8, 9] for survey papers of the area).

An *equational theory* is given by a set E of identities $s = t$ between terms s, t. The *signature of* E is the set of all function symbols occurring in E. With $=_E$ we denote the least congruence relation on the term algebra $\mathcal{T}(\Sigma, V)$ that is closed under substitutions and contains E. Equivalently, $=_E$ can be introduced as the reflexive, transitive, and symmetric closure $\overset{*}{\leftrightarrow}_E$ of the rewrite relation $\rightarrow_E$ induced by E, or as the set of equational consequences of E, i.e., $s =_E t$ iff the universal closure of the atom $s = t$ is valid in all models of E. An equational theory E is *trivial* iff $x =_E y$ holds for two distinct variables x, y. In the following, we consider only non-trivial equational theories.

Given an equational theory E with signature Σ, an *elementary E-unification problem* is a finite system $\Gamma := \{s_1 =^? t_1, \ldots, s_n =^? t_n\}$ of equations between Σ-terms. In E-unification problems *with constants*, these terms may contain additional "free" constant symbols, i.e., constant symbols not contained in the signature Σ of E, and in *general* E-unifications problems, these terms may contain additional "free" function symbols, i.e., function symbols not contained in the signature Σ of E.

A *solution* (or *E-unifier*) of the E-unification problem $\{s_1 =^? t_1, \ldots, s_n =^? t_n\}$ is a substitution σ such that $\sigma(s_i) =_E \sigma(t_i)$ for $i = 1, \ldots, n$. If there exists such a solution, then Γ is called *unifiable*. Recall that a *substitution* is a mapping from variables to terms which is almost everywhere equal to the identity. It can be extended to a mapping from terms to terms in the obvious way. Substitutions will be written in the form $\sigma = \{x_1 \mapsto u_1, \ldots, x_k \mapsto u_k\}$, where $x_1, \ldots, x_k$ are the finitely many variables that are changed by the substitution, and $u_1, \ldots, u_k$ are their respective σ-images. The set $\{x_1, \ldots, x_k\}$ is called the *domain* of the substitution σ, and $\{u_1, \ldots, u_k\}$ is called its *range*.

Let X be the set of all variables occurring in Γ. The E-unifier θ is an instance of the E-unifier σ iff there exists a substitution λ such that $\theta(x) =_E \lambda(\sigma(x))$ for all $x \in X$. A *complete set of E-unifiers of Γ* is a set C of E-unifiers of Γ such that every E-unifier of Γ is an instance of some unifier in C. Finite complete sets of E-unifiers yield a finite representation of the usually infinite sets of solutions of E-unification problems. Since this representation should be as small as possible, one is usually interested in *minimal complete sets of E-unifiers*, i.e., complete sets in which different elements are not instances of each other.

For a given equational theory E, elementary E-unification problems can be seen as constraints of the following constraint system:

- The constraint language consists of all conjunctions of equational atoms $s = t$. We call such a conjunction an *E-unification constraint*.
- The semantics is defined by the E-free algebra in countably many generators, i.e., the quotient term algebra $\mathcal{T}(\Sigma, V)/=_E$ for the countably infinite set of variables V.

Since the E-unification problem $\{s_1 =^? t_1, \ldots, s_n =^? t_n\}$ has a solution iff the existential closure of $s_1 = t_1 \wedge \ldots \wedge s_n = t_n$ is valid in $\mathcal{T}(\Sigma, V)/=_E$, the notion of satisfiability of E-unification constraints induced by this semantics coincide with the notion of unifiability of E-unification problems introduced above.

Alternatively, the semantics of E-unification constraints can also be defined w.r.t. the equational theory E. In fact, the E-free algebra in countably many generators is a canonical solution structure for E-unification constraints: the existential closure of $s_1 = t_1 \wedge \ldots \wedge s_n = t_n$ is valid in $\mathcal{T}(\Sigma, V)/=_E$ iff it is valid in *all* models of E.

If we are interested only in decidability and not in complexity, then we can also consider general positive existential sentences instead of the conjunctive sentences obtained as existential closures of unification constraints. In fact, we can simply write the positive matrix in disjunctive normal form, and then distribute the existential quantifiers over disjunctions. This yields a disjunction of conjunctive sentences, which is valid iff one of its disjuncts is valid.

These observations are summed up in the following theorem:

Theorem 1. *Let E be an equational theory with signature Σ, and V a countably infinite set of variables. Then the following are equivalent:*

1. *Elementary E-unification is decidable.*
2. *The positive existential theory of E is decidable.*
3. *The positive existential theory of $\mathcal{T}(\Sigma, V)/=_E$ is decidable.*

In order to obtain a class of first-order formulae that correspond to E-unification problems with constants, but are built over the signature of E, we note that free constants can be generated via Skolemization. Since we are interested in validity of the formulae, we must Skolemize universal quantifiers, and since we want to obtain Skolem constants, these universal quantifiers should not be in the scope of an existential quantifier.

Theorem 2. *Let E be an equational theory with signature Σ, and V a countably infinite set of variables. Then the following are equivalent:*

1. *E-unification with constants is decidable.*
2. *The positive AE theory of E is decidable.*
3. *The positive AE theory of $\mathcal{T}(\Sigma, V)/=_E$ is decidable.*

These theorems can be seen as folk theorems in the unification community. Explicit proofs can be found in [14]. For general E-unification, a similar characterization is possible. However, the proof of this result (which is not as straightforward as the ones for the above theorems) depends on results concerning the combination of E-unification algorithms. For this reason, we defer presenting the characterization to Section 5.1.

2.3 Combining constraint systems and solvers

Given two constraint systems, it is not a priori clear what their combination is supposed to be, and in some cases there are several sensible candidates. Since constraint systems consist of a constraint language and a corresponding semantics, one must first define the combined language, and then introduce a combined semantics for this language.

Let us first consider the problem of defining the combined constraint language. This is quite simple if we restrict ourselves to the case where the two constraint languages consist of formulae of the same type, but over different signatures. In this case, the most natural candidate for the combined constraint language appears to be the language consisting of formulae of the given type, but over the union of the signatures. For example, if the constraint languages allow for quantifier-free formulae over the signatures Σ_1 and Σ_2, respectively, then the combined constraint language consists of all quantifier-free formulae over the signature $\Sigma_1 \cup \Sigma_2$. Similarly, combined unification problems consist of equations between terms over the union of the signatures of the component constraint systems. In this paper, we will consider only this simple case.

Once the combined constraint language is defined, it must be equipped with an appropriate semantics. If the semantics is defined via a theory, this is again quite simple: the natural candidate is the union of the component theories. Thus, given equational theories E_1 and E_2 with the respective signatures Σ_1 and Σ_2, the combined unification constraint system consists of all $(E_1 \cup E_2)$-unification problems (with the usual semantics). If the semantics of the component constraint systems is defined with the help of solution structures, things are more difficult. One must combine the solution structures of the components into a solution structure for the combined constraint language, and it not always obvious how this combined solution structure should look like. We will briefly come back to this problem of combining solution structures in Subsection 6.2.

Finally, given the combined constraint language with an appropriate semantics, one needs a constraint solver for this combined constraint system. For two specific constraint systems and their combination, one can of course try to construct an ad hoc constraint solver for the combined system, which may or may

not employ the single solvers as subprocedures. A more satisfactory approach, however, is to design a combination scheme that applies to a whole class of constraint systems. The combination procedures that we will consider in the next two sections are of this form. For example, the Nelson-Oppen combination procedure can be used to combine decision procedures for the universal theories of T_1 and T_2 into one for the universal theory of $T_1 \cup T_2$. This holds for arbitrary theories T_1 and T_2 with decidable universal theory (and not just for two specific theories), provided that the signatures of T_1 and T_2 are disjoint. The combination result for E-unification algorithms is of the same type.

In both cases, the combination approach treats the solvers of the single constraint systems as black boxes, i.e., it does not make any assumptions on how these solvers work. This distinguishes these approaches from others that assume the existence of constraint solvers of a certain type. For example, a semi-complete (i.e., confluent and weakly normalizing) term rewriting system can be used to decide the word problem of the corresponding equational theory. Since confluence and weak normalization are modular properties, the union of two semi-complete term rewriting systems over disjoint signatures is again semi-complete [53], and thus the word problem in the combined equational theory is decidable as well. However, this result is of no help at all if the decision procedures for the word problem in the component equational theories are not based on rewriting.

3 The Nelson-Oppen combination procedure

This procedure, which was first introduced in [49], is concerned with combining decision procedures for the validity of universal sentences in first-order theories, or equivalently with combining constraint solvers that test satisfiability of quantifier-free formulae in some model of the theory. To be more precise, assume that Σ_1 and Σ_2 are two *disjoint* signatures, and that T is obtained as the union of a Σ_1-theory T_1 and a Σ_2-theory T_2. How can decision procedures for validity (equivalently: satisfiability) in T_i $(i = 1, 2)$ be used to obtain a decision procedure for validity (equivalently: satisfiability) in T?

When considering the satisfiability problem, as done in Nelson and Oppen's method, we may without loss of generality restrict our attention to *conjunctive* quantifier-free formulae, i.e., conjunctions of Σ-atoms and negated Σ-atoms. In fact, a given quantifier-free formula can be transformed into an equivalent formula in disjunctive normal form (i.e., a disjunction of conjunctive quantifier-free formulae), and this disjunction is satisfiable in T iff one of its disjuncts is satisfiable in T.

Given a conjunctive quantifier-free formula φ over the combined signature $\Sigma_1 \cup \Sigma_2$, it is easy to generate a conjunction $\varphi_1 \wedge \varphi_2$ that is equivalent to φ, where φ_i is a pure Σ_i-formula, i.e., contains only symbols from Σ_i $(i = 1, 2)$. Here equivalent means that φ and $\varphi_1 \wedge \varphi_2$ are satisfiable in exactly the same models of T. This is achieved by *variable abstraction*, i.e., by replacing alien subterms by variables and adding appropriate equations.

Variable abstraction Assume that t is a term whose topmost function symbol is in Σ_i, and let j be such that $\{i, j\} = \{1, 2\}$. A subterm s of t is called *alien subterm* of t iff its topmost function symbol belongs to Σ_j and every proper superterm of s in t has its top symbol in Σ_i.

Given a conjunctive quantifier-free formula φ, the variable abstraction process iteratively replaces terms by variables and adds appropriate equations to the conjunction:

- If φ contains an equational conjunct $s = t$ such that the topmost function symbols of s and t belong to different signatures, then replace $s = t$ by the conjunction $u = s \wedge u = t$, where u is a new variable, i.e., a variable not occurring in φ.
- If φ contains a negated equational conjunct $s \neq t$ such that the topmost function symbols of s and t belong to different signatures, then replace $s = t$ by the conjunction $u \neq v \wedge u = s \wedge v = t$, where u, v are new variables.
- If φ contains a relational conjunct $P(\ldots, s_i, \ldots)$ such that the topmost function symbol of s_i does not belong to the signature of P, then replace $P(\ldots, s_i, \ldots)$ by the conjunction $P(\ldots, u, \ldots) \wedge u = s_i$, where u is a new variable. Conjuncts of the form $\neg P(\ldots, s_i, \ldots)$ are treated analogously.
- If φ contains a (relational or equational) conjunct $P(\ldots, s_i, \ldots)$ such that s_i contains an alien subterm t, then replace $P(\ldots, s_i, \ldots)$ by the $P(\ldots, s_i', \ldots) \wedge u = t$, where u is a new variable and s_i' is obtained from s_i by replacing the alien subterm t by u. Conjuncts of the form $\neg P(\ldots, s_i, \ldots)$ are treated analogously.

Obviously, this abstraction process always terminates and the resulting formula can be written in the form $\varphi_1 \wedge \varphi_2$, where φ_i is a pure Σ_i-formula $(i = 1, 2)$. In addition, it is easy to see that the original formula φ and the new formula $\varphi_1 \wedge \varphi_2$ are satisfiable in exactly the same models of $T = T_1 \cup T_2$. Consequently, if φ is satisfiable in a model of T, then both φ_1 and φ_2 are satisfiable in a model of T, which is also a model of T_1 and of T_2. This shows that satisfiability of φ in a model of T implies satisfiability of φ_i in a model of T_i for $i = 1, 2$. Unfortunately, the converse need not hold, i.e., satisfiability of φ_i in a model of T_i $(i = 1, 2)$ does not necessarily imply satisfiability of $\varphi_1 \wedge \varphi_2$ in a model of T, and thus also not satisfiability of φ in a model of T.

The reason for this problem is that φ_1 and φ_2 may share variables, and one formula may force some of these variables to be interpreted by the same element of the model, whereas the other is only satisfiable if they are interpreted by distinct elements of the model. To overcome this problem, Nelson and Oppen's procedure propagates equalities between variables from the formula φ_1 to φ_2, and vice versa.

The combination procedure Given a conjunctive quantifier-free $(\Sigma_1 \cup \Sigma_2)$-formula φ to be tested for satisfiability (in some model of $T_1 \cup T_2$), Nelson and Oppen's method for combining decision procedures proceeds in three steps:

1. *Use variable abstraction to generate a conjunction $\varphi_1 \wedge \varphi_2$ that is equivalent to φ, where φ_i is a pure Σ_i-formula ($i = 1, 2$).*
2. *Test the pure formulae for satisfiability in the respective theories.*
   If φ_i is unsatisfiable in T_i for $i = 1$ or $i = 2$, then return "unsatisfiable." Otherwise proceed with the next step.
3. *Propagate equalities between different shared variables (i.e., distinct variables u_i, v_i occurring in both φ_1 and φ_2), if a disjunction of such equalities can be deduced from the pure parts.*
   A disjunction $u_1 = v_1 \vee \ldots \vee u_k = v_k$ of equations between different shared variables can be deduced from φ_i in T_i iff $\varphi_i \wedge u_1 \neq v_1 \wedge \ldots \wedge u_k \neq v_k$ is unsatisfiable in T_i. Since the satisfiability problem in T_i was assumed to be decidable, and since there are only finitely many shared variables, it is decidable whether there exists such a disjunction.
   If no such disjunctions can be deduced, return "satisfiable." Otherwise, take any of them,[1] and propagate its equations as follows. For every disjunct $u_j = v_j$, proceed with the second step for the formula $\sigma_j(\varphi_1) \wedge \sigma_j(\varphi_2)$, where $\sigma_j := \{u_j \mapsto v_j\}$ (for $j = 1, \ldots, k$). The answer is "satisfiable" iff one of these cases yields "satisfiable."

Obviously, the procedure *terminates* since there are only finitely many shared variables to be identified. In addition, it is easy to see that satisfiability is preserved at each step. This implies *completeness* of the procedure, that is, if it answers "unsatisfiable" (since already one of the pure subformulae is unsatisfiable in its theory), the original formula was indeed unsatisfiable. Before showing soundness of the procedure (which is more involved), we illustrate the working of the procedure by an example.

Example 1. Consider the equational[2] theories $T_1 := \{f(x, x) = x\}$ and $T_2 := \{g(g(x)) = g(x)\}$ over the signatures $\Sigma_1 := \{f\}$ and $\Sigma_2 := \{g\}$. Assume that we want to know whether the quantifier-free (mixed) formula $g(f(g(z), g(g(z)))) = g(z)$ is valid in $T_1 \cup T_2$. To this purpose we apply the Nelson-Oppen procedure to its negation $g(f(g(z), g(g(z)))) \neq g(z)$.

In *Step 1*, $f(g(z), g(g(z)))$ is an alien subterm in $g(f(g(z), g(g(z))))$ (since $g \in \Sigma_2$ and $f \in \Sigma_1$). In addition, $g(z)$ and $g(g(z))$ are alien subterms in $f(g(z), g(g(z)))$. Thus, variable abstraction yields the conjunction $\varphi_1 \wedge \varphi_2$, where

$$\varphi_1 := u = f(v, w) \quad \text{and} \quad \varphi_2 := g(u) \neq g(z) \wedge v = g(z) \wedge w = g(g(z)).$$

In *Step 2*, it is easy to see that both pure formulae are satisfiable in their respective theories. The equation $u = f(v, w)$ is obviously satisfiable in the trivial model of T_1 (of cardinality 1). The formula φ_2 is, for example, satisfiable in the T_2-free algebra with two generators, where u is interpreted by one generator, z by the other, and v, w as required by the equations.

[1] For efficiency reasons, one should take a disjunction with minimal k.
[2] Recall that the identities in equational theories are (implicitly) universally quantified.

In *Step 3*, we can deduce $w = v$ from φ_2 in T_2 since φ_2 contains $v = g(z) \wedge w = g(g(z))$ and T_2 contains the (universally quantified) identity $g(g(x)) = g(x)$. Propagating the equality $w = v$ yields the pure formulae

$$\varphi_1' := u = f(v, v) \quad \text{and} \quad \varphi_2' := g(u) \neq g(z) \wedge v = g(z) \wedge v = g(g(z)),$$

which again turn out to be separately satisfiable in *Step 2* (with the same models as used above).

In *Step 3*, we can now deduce the equality $u = v$ from φ_1' in T_1, and its propagation yields

$$\varphi_1'' := v = f(v, v) \quad \text{and} \quad \varphi_2'' := g(v) \neq g(z) \wedge v = g(z) \wedge v = g(g(z)).$$

In *Step 2*, it turns out that φ_2'' is not satisfiable in T_2, and thus the answer is "unsatisfiable," which shows that $g(f(g(z), g(g(z)))) = g(z)$ is valid in $T_1 \cup T_2$. In fact, $v = g(z)$ and the identity $g(g(x)) = g(x)$ of T_2 imply that $g(v) = g(z)$, which contradicts $g(v) \neq g(z)$.

Soundness of the procedure As mentioned above, termination and completeness of the procedure are quite trivial. Soundness of the procedure, i.e., if the procedure answers "satisfiable," then the input formula is indeed satisfiable, is less trivial. In fact, for arbitrary theories T_1 and T_2, the combination procedure need not be sound. One must assume that each T_i is *stably infinite*, that is, such that a quantifier-free formula φ_i is satisfiable in T_i iff it is satisfiable in an infinite model of T_i. This restriction was not mentioned in Nelson and Oppen's original article [49]; it was introduced by Oppen in [54].

The following example demonstrates that the Nelson-Oppen combination procedure need not be sound for theories that are not stably infinite, even if the theories in question are non-trivial[3] equational theories.

Example 2. Let $E_1 := \{f(g(x), g(y)) = x,\ f(g(x), h(y)) = y\}$ and $E_2 := \{k(x) = k(x)\}$. The theory E_2 is obviously non-trivial, and it is easy to see that E_1 is also non-trivial: by orienting the equations from left to right, one obtains a canonical term rewriting system, in which any two distinct variables have different normal forms.

First, we show that E_1 is not stably infinite. To this purpose, we consider the quantifier-free formula $g(x) = h(x)$. Obviously, this formula is satisfiable in the trivial (one-element) model of E_1. In every model $\mathcal{A}$ of E_1 that satisfies $g(x) = h(x)$, there exists an element e such that $g^{\mathcal{A}}(e) = h^{\mathcal{A}}(e)$. Here, $g^{\mathcal{A}}, h^{\mathcal{A}}$ denote the interpretations of the unary function symbols f, g by functions $A \to A$. But then we have for any element a of $\mathcal{A}$ that

$$a = f^{\mathcal{A}}(g^{\mathcal{A}}(a), g^{\mathcal{A}}(e)) = f^{\mathcal{A}}(g^{\mathcal{A}}(a), h^{\mathcal{A}}(e)) = e,$$

[3] Recall that non-trivial means that the theory has a model of cardinality greater than 1.

i.e., all elements of $\mathcal{A}$ are equal to e, which shows that $\mathcal{A}$ is the trivial model. Thus, $g(x) = h(x)$ is satisfiable only in the trivial model of E_1, which show that the (non-trivial) equational theory E_1 is not stably infinite.

To show that this really leads to an unsound behavior of the Nelson-Oppen method, we consider the mixed conjunction $g(x) = h(x) \wedge k(x) \neq x$. Clearly, $k(x) \neq x$ is satisfiable in E_2 (for instance, in the E_2-free algebra with 1 generator) and, as we saw earlier, $g(x) = h(x)$ is satisfiable in E_1. In addition, no equations between distinct shared variables can be deduced (since there is only one shared variable). It follows that Nelson and Oppen's procedure would answer "satisfiable" on input $g(x) = h(x) \wedge k(x) \neq x$. However, since $g(x) = h(x)$ is satisfiable only in the trivial model of E_1, and no disequation can be satisfied in a trivial model, $g(x) = h(x) \wedge k(x) \neq x$ is unsatisfiable in $E_1 \cup E_2$.

Nelson and Oppen's original proof of soundness of the procedure as well as a more recent one by Tinelli and Harandi [72] use Craig's Interpolation Theorem [20]. In the following, we sketch a proof that uses a very elementary model theoretic construction: the fusion of structures. It goes back to Ringeissen [58] and was further refined by Ringeissen and Tinelli [73, 71].[4]

In the following, let Σ_1 and Σ_2 be disjoint signatures, and $\Sigma := \Sigma_1 \cup \Sigma_2$ their union. A given Σ-structure $\mathcal{A}$ can also be viewed as a Σ_i-structure, by just forgetting about the interpretation of the symbols not contained in Σ_i. We call this Σ_i-structure the *Σ_i-reduct* of $\mathcal{A}$, and denote it by $\mathcal{A}^{\Sigma_i}$.

Definition 1. *The Σ-structure $\mathcal{A}$ is a fusion of the Σ_1-structure $\mathcal{A}_1$ and the Σ_2-structure $\mathcal{A}_2$ iff the Σ_i-reduct $\mathcal{A}^{\Sigma_i}$ of $\mathcal{A}$ is Σ_i-isomorphic to $\mathcal{A}_i$ $(i = 1, 2)$.*

Since the signatures Σ_1 and Σ_2 are disjoint, the existence of a fusion depends only on the cardinality of the structures $\mathcal{A}_1$ and $\mathcal{A}_2$.

Lemma 1. *Let $\mathcal{A}_1$ be a Σ_1-structure and $\mathcal{A}_2$ a Σ_2-structure. Then $\mathcal{A}_1$ and $\mathcal{A}_2$ have a fusion iff their domains A_1 and A_2 have the same cardinality.*

Proof. The only-if direction of the lemma is an immediate consequence of the definition of fusion. The if direction can be seen as follows: if A_1 and A_2 have the same cardinality, then there exists a bijection $\pi : A_1 \to A_2$. This bijection can be used to transfer the interpretation of the elements of Σ_2 from $\mathcal{A}_2$ to $\mathcal{A}_1$. To be more precise, let $\mathcal{A}$ be the Σ-structure that has domain A_1, interprets the elements of Σ_1 like $\mathcal{A}_1$, and interprets the elements of Σ_2 as follows:

- If f is an n-ary function symbol in Σ_2, and $a_1, \ldots, a_n \in A_1$, then we define $f^{\mathcal{A}}(a_1, \ldots, a_n) := \pi^{-1}(f^{\mathcal{A}_2}(\pi(a_1), \ldots, \pi(a_n)))$.
- If P is an n-ary predicate symbol in Σ_2, and $a_1, \ldots, a_n \in A_1$, then $(a_1, \ldots, a_n) \in P^{\mathcal{A}}$ iff $(\pi(a_1), \ldots, \pi(a_n)) \in P^{\mathcal{A}_2}$.

[4] Actually, these papers consider the more general situation where the signatures need not be disjoint. Here, we restrict our attention to the disjoint case.

Then $\mathcal{A}$ is a fusion of $\mathcal{A}_1$ and $\mathcal{A}_2$ since $\mathcal{A}^{\Sigma_1}$ is identical to $\mathcal{A}_1$, and π is a Σ_2-isomorphism from $\mathcal{A}^{\Sigma_2}$ to $\mathcal{A}_2$ by construction of $\mathcal{A}$. □

There is an interesting connection between the union of theories and fusions of models of the theories.

Proposition 1. *For $i = 1, 2$, let T_i be a Σ_i-theory. The Σ-structure $\mathcal{A}$ is a model of $T_1 \cup T_2$ iff it is a fusion of a model $\mathcal{A}_1$ of T_1 and a model $\mathcal{A}_2$ of T_2.*

Proof. The only-if direction is an immediate consequence of the facts that $\mathcal{A}$ is a fusion of its Σ_1-reduct $\mathcal{A}^{\Sigma_1}$ and its Σ_2-reduct $\mathcal{A}^{\Sigma_2}$, and that $\mathcal{A}^{\Sigma_i}$ is a model of T_i $(i = 1, 2)$.

The if direction is also trivial since, if $\mathcal{A}$ is a fusion of a model $\mathcal{A}_1$ of T_1 and a model $\mathcal{A}_2$ of T_2, then its Σ_i-reduct is isomorphic to $\mathcal{A}_i$, and thus a model of T_i $(i = 1, 2)$. Consequently, $\mathcal{A}$ is a model of $T_1 \cup T_2$. □

We are now ready to show a result from which soundness of the Nelson-Oppen procedure follows immediately. For a finite set of variables X, let $\Delta(X)$ denote the conjunction of all negated equations $x \neq y$ for distinct variables $x, y \in X$.

Proposition 2. *Let T_1 and T_2 be two stably infinite theories over the disjoint signatures Σ_1 and Σ_2, respectively; let φ_i be a quantifier-free Σ_i-formula $(i = 1, 2)$, and let X be the set of variables occurring in both φ_1 and φ_2. If $\varphi_i \wedge \Delta(X)$ is satisfiable in a model $\mathcal{A}_i$ of T_i for $i = 1, 2$, then $\varphi_1 \wedge \varphi_2$ is satisfiable in a fusion of $\mathcal{A}_1$ and $\mathcal{A}_2$, and thus in a model of $T_1 \cup T_2$.*

Proof. Since the theories T_1 and T_2 are stably infinite and signatures are at most countable, we may without loss of generality assume that the structures $\mathcal{A}_1$ and $\mathcal{A}_2$ are both countably infinite. Since $\varphi_i \wedge \Delta(X)$ is satisfiable in $\mathcal{A}_i$, there is an evaluation $\alpha_i : X \to A_i$ that satisfies φ_i and replaces the variables in X by distinct elements of A_i $(i = 1, 2)$. This implies that there exists a bijection $\pi : A_1 \to A_2$ such that $\pi(\alpha_1(x)) = \alpha_2(x)$. As shown in the proof of Lemma 1, this bijection induces a fusion $\mathcal{A}$ of $\mathcal{A}_1$ and $\mathcal{A}_2$. It is easy to see that $\varphi_1 \wedge \varphi_2$ is satisfiable in $\mathcal{A}$. The evaluation α showing satisfiability is defined as follows: on the variables in φ_1 it coincides with α_1, and for the non-shared variables x in φ_2 we define $\alpha(x) := \pi^{-1}(\alpha_2(x))$. □

It is easy to see that this proposition yields soundness of the Nelson-Oppen combination procedure, i.e, if the procedure answers "satisfiable," then the original formula was indeed satisfiable. In fact, if in Step 3 no disjunction of equalities between shared variables can be derived from the pure formulae, the prerequisite for the proposition is satisfied: since the disjunction of all equations $x = y$ for distinct variables $x, y \in X$ cannot be deduced from φ_i in T_i, we know that $\varphi_i \wedge \Delta(X)$ is satisfiable in T_i. Thus, we can deduce that $\varphi_1 \wedge \varphi_2$ is satisfiable in $T_1 \cup T_2$, and since each step of the procedure preserves satisfiability, the input formula was also satisfiable.

To sum up, we have shown correctness of Nelson and Oppen's combination procedure, which yields the following theorem:

Theorem 3. *Let T_1 and T_2 be two stably infinite theories over disjoint signatures such that the universal theory of T_i is decidable for $i = 1, 2$. Then the universal theory of $T_1 \cup T_2$ is also decidable.*

Complexity of the procedure The main sources of complexity are (i) the transformation of the quantifier-free formula into a disjunction of *conjunctive* quantifier-free formulae, and (ii) Step 3 of the combination procedure for conjunctive quantifier-free formulae. It is well-known that the transformation of an arbitrary Boolean formula into disjunctive normal form may cause an exponential blow-up. Step 3 of the procedure has again two sources of complexity. First, there is the problem of deciding whether there is a disjunction of equalities between distinct variables that can be derived from φ_i in T_i. If one must really test all possible disjunctions using the satisfiability procedure for T_i, then already a single such step needs exponential time. However, even if we assume that there is a polynomial procedure that determines an appropriate disjunction (if there is one), then the overall algorithm is still not polynomial unless all these disjunctions consist of a single disjunct. Otherwise, the algorithm must investigate different branches, and since this may happen each time Step 3 is performed, an exponential number of branches may need to be investigated.

Nelson and Oppen [49] introduce the notion of a convex theory, and Oppen [54] shows that, for convex theories, the two sources of complexity in Step 3 of the procedure can be avoided. A theory T is *convex* iff the following the following holds: if a disjunction of equalities between distinct variables can be deduced from a quantifier-free formula in T, then a single equality between distinct variables can already be deduced. For convex theories, Step 3 of the procedure can thus be modified as follows: it is only tested whether a single equation between distinct shared variables can be deduced. Since there are only a polynomial number of such equations, this can be tested by a polynomial number of calls to the satisfiability procedure for T_i. In addition, there is no more branching in Step 3. This shows that the modified combination procedure runs in polynomial time, if applied to conjunctive quantifier-free input formulae.

Theorem 4. *Let T_1 and T_2 be two* convex *and stably infinite theories over disjoint signatures such that the* conjunctive *universal theory of T_i is decidable in polynomial time for $i = 1, 2$. Then the* conjunctive *universal theory of $T_1 \cup T_2$ is also decidable in polynomial time.*

In the general case, the Nelson-Oppen combination approach yields a *nondeterministic* polynomial procedure. First, given an arbitrary quantifier-free formula, the nondeterministic procedure chooses from each disjunction one of the disjuncts. This yields a conjunctive quantifier-free formula. This formula is then treated by the following nondeterministic variant of the combination procedure:

1. *Use variable abstraction to generate a conjunction $\varphi_1 \wedge \varphi_2$ that is equivalent to φ, where φ_i is a pure Σ_i-formula ($i = 1, 2$).*

2. *Nondeterministically choose a variable identification, i.e., choose a partition* $\Pi = \{\pi_1, \ldots, \pi_k\}$ *of the variables shared by* φ_1 *and* φ_2.
   For each of the classes π_i, let $x_i \in \pi_i$ be a representative of this class, and let $X_\Pi := \{x_1, \ldots, x_k\}$ be the set of these representatives. The substitution that replaces, for all $i = 1, \ldots, k$, each element of π_i by its representative x_i is denoted by σ_Π.
3. *Test the pure formulae for satisfiability in the respective theories.*
   If $\sigma_\Pi(\varphi_i) \wedge \Delta(X_\Pi)$ is unsatisfiable in T_i for $i = 1$ or $i = 2$, then return "unsatisfiable;" otherwise return "satisfiable."[5]

Obviously, it is possible to choose one partition using only a polynomial number of binary choices, which shows that the above procedure is indeed nondeterministic polynomial. Completeness is again easy to see, and soundness is an immediate consequence of Proposition 2.

Theorem 5. *Let* T_1 *and* T_2 *be two stably infinite theories over disjoint signatures such that the universal theory of* T_i *is decidable in NP for* $i = 1, 2$. *Then the universal theory of* $T_1 \cup T_2$ *is also decidable in NP.*

Extensions and related work Shostak [64] describes a more efficient combination procedure, which is based on congruence closure [50]. However, unlike the Nelson-Oppen procedure, this approach assumes that decision procedures of a specific form (so-called "canonizers" and "solvers") exist for the component theories, i.e., is does not treat the component decision procedures as black boxes. A formally more rigorous presentation of the method can be found in [25].

Above, we have always assumed that the theories to be combined are over disjoint signatures. Without any such restriction, a general combination procedure cannot exist. In fact, it is very easy to find examples of decidable theories whose (non-disjoint) combination is undecidable. Nevertheless, it is worthwhile to try to weaken the disjointness assumption. The first extension of Nelson and Oppen's approach in this direction is due to Ringeissen [58]. It was further extended by Ringeissen and Tinelli [73, 71] to the case of theories sharing "constructors."

Baader and Tinelli [10] consider the application of the Nelson-Oppen procedure to equational theories. Although equational theories need not be stably infinite (see Example 2), Nelson and Oppen's procedure can be applied, after some minor modifications, to combine decision procedures for the validity of quantifier-free formulae in equational theories. It is also shown that, contrary to a common belief, the method cannot be used to combine decision procedures for the word problem. The paper then presents a method that solves this kind of combination problem. In [11, 12] it is shown that this approach can also be extended to the case of theories sharing constructors.

[5] Recall that $\Delta(X_\Pi)$ denotes the conjunction of all negated equations $x \neq y$ for distinct variables $x, y \in X_\Pi$.

4 Combination of E-unification algorithms

The constraints that we treat in this section are E-unification problems, i.e., systems of term equations that must be solved modulo a given equational theory E. From a more logical point of view, this means that we are interested in the positive existential or the positive AE theory of E (see Theorems 1 and 2), depending on whether we consider elementary unification or unification with constants. During the last three decades, research in unification theory has produced E-unification algorithms (i.e., constraint solvers for E-unification constraints) for a great variety of equational theories E (see [33, 8, 9]). Such an algorithm either actually computes solutions of the E-unification constraints (usually complete sets of E-unifiers), or it just decides satisfiability of E-unification constraints. Using decision procedures instead of algorithms computing complete sets of unifiers may be advantageous for theories where the complete sets are large or even infinite.

E-unification algorithms that compute complete sets of unifiers are, for example, applied in theorem proving with "built in" theories (see, e.g., [55, 68]), in generalizations of the Knuth-Bendix completion procedure to rewriting modulo theories (see, e.g., [34, 13]), and in logic programming with equality (see, e.g., [32]). With the development of constraint approaches to theorem proving (see, e.g., [18, 51]), term rewriting (see, e.g., [41]), and logic programming (see, e.g., [31, 22]), decision procedures for E-unification have been gaining in importance.

The combination problem for E-unification algorithms is directly motivated by these applications. In this section, we first motivate the problem and briefly review the research on this topic, which led to a complete solution for the case of theories over disjoint signatures. Subsequently, we describe the combination algorithm developed in [2, 6], and sketch how to prove its correctness. In Section 5, we derive an algebraic and logical reformulation of the combination problem and the combination algorithm. This leads to a more abstract proof, which can also be generalized to other classes of constraints (Section 6). Finally, in Section 7 we comment on the complexity of the combination problem, and describe possible optimizations of the combination procedure.

4.1 The problem and its history

Basically, the problem of combining E-unification algorithms can be described as follows:

> *Assume we are given unification algorithms for solving unification problems modulo the equational theories E_1 and E_2. How can we obtain a unification algorithm for the union of the theories, $E_1 \cup E_2$?*

Here, unification algorithms may either be algorithms computing complete sets of unifiers or decision procedures for unifiability.

The relevance of this problem relies on the observation that, quite often, a given E-unification algorithm can only treat unification problems where the

terms occurring in the problem are composed over the signature of E (elementary E-unification), possibly enriched by some free constants (E-unification with constants). This is, for example, the case for the "natural" unification algorithms for the theory AC of an associative-commutative function symbol [67, 30], which depend on solving linear Diophantine equations in the natural numbers. However, in the applications mentioned above, unification problems often contain "mixed" terms, i.e, terms that are constructed from function symbols belonging to different theories.

For example, in automated theorem proving, free function symbols of arbitrary arity are frequently introduced by Skolemization. Thus, the E-unification problems that must be solved there are usually *general* E-unification problems. If the given E-unification algorithm can treat only E-unification problems with constants, the question arises whether it is always possible to construct an algorithm for general E-unification from a given algorithm for E-unification with constants. This can be seen as an instance of the combination problem where E_1 is the theory E, and E_2 is the free theory for the free function symbols (e.g., consisting of the "dummy" identities $f(x_1, \ldots, x_n) = f(x_1, \ldots, x_n)$ for the free function symbols f). The combination problem in its general form arises if the semantic properties of several function symbols are to be integrated into the unification; for example, one may want to build in an associative symbol representing concatenation of lists, and an associative-commutative symbol representing addition of numbers.

Similarly as in the case of the Nelson-Oppen procedure, there cannot be a general solution to the combination problem as stated above: there exist equational theories E_1 and E_2 where unification with constants is decidable both for E_1 and for E_2, but solvability of unification problems with constants modulo $E_1 \cup E_2$ is undecidable. For example, both unification with constants modulo left-distributivity of the binary symbol f over the binary symbol g [70] and unification with constants modulo associativity of the binary symbol g [46] are decidable, but unification with constants modulo the union of these theories is undecidable [65]. Again, the restriction to theories over disjoint signatures avoids this problem. Until now, most of the research was concentrated on this restricted case.

A first important instance of the problem was considered by Stickel [66, 67]. Stickel's AC-unification algorithm allowed for the presence of several AC-symbols and free symbols. However, termination of this algorithm could only be shown for restricted cases and it took almost a decade until Fages [27] could close this gap.

Subsequently, more general combination problems were, for example, treated in [40, 69, 29, 75, 16], but the theories considered in these papers always had to satisfy certain restrictions (such as collapse-freeness or regularity) on the syntactic form of their defining identities. Recall that a theory E is called collapse-free if it does not contain an identity of the form $x = t$ where x is a variable and t is a non-variable term, and it is called regular if the left- and right-hand sides of the identities contain the same variables. Such restrictions simplify the combina-

tion problem, both from the conceptual and from the computational complexity point of view.

An important break-through in the research on the combination problem was the combination algorithm by Schmidt-Schauß [60]. The algorithm applies to arbitrary equational theories over disjoint signatures, provided that an additional algorithmic requirement is fulfilled: in addition to algorithms for unification with constants, one needs algorithms that solve so-called constant elimination problems. A more efficient version of this highly nondeterministic algorithm has been described by Boudet [15]. Basically, whereas the algorithm by Schmidt-Schauß performs two nondeterministic steps right at the beginning, Boudet's algorithm tries to defer nondeterministic steps as long as possible; nondeterminism is only used "on demand" to resolve certain conflicts. For restricted classes of theories (e.g., collapse-free theories) some of these conflicts cannot occur, and thus the corresponding nondeterministic steps can be avoided.

The combination procedures mentioned until now all considered the problem of combining algorithms that compute (finite) complete sets of E-unifiers. The problem of how to combine decision procedures is not solved by these approaches, in particular not for theories like associativity, where unification problems need not have a *finite* complete set of unifiers. Actually, the paper by Schmidt-Schauß [60] also considered the problem of combining decision procedures. It showed that decision procedures can be combined, provided that solvability of *general* unification problems is decidable in the component theories. The drawback of this result was that for many theories (e.g., associativity) one already needs to employ combination methods to show that general unification (i.e., unification in the combination of the given theory with syntactic equality of the free function symbols) is decidable.

The problem of how to combine decision procedures was finally solved in a very general form in [2, 6], where a combination algorithm was given that can be used both for combining decision procedures and for combining algorithms computing complete sets of unifiers. This algorithm applies to arbitrary equational theories over disjoint signatures, but it requires as a prerequisite that algorithms solving so-called unification problems *with linear constant restrictions*[6] are available for these theories (which was, e.g., the case for associativity). In the sequel, we describe this combination algorithm.

All the combination results that will be presented in the following are restricted to the case of disjoint signatures. There are some approaches that try to weaken the disjointness assumption, but the theories to be combined must satisfy rather strong conditions [57, 26].

4.2 A combination algorithm for E-unification algorithms

In the following, we consider equational theories E_1 and E_2 over the disjoint signatures Σ_1 and Σ_2. We denote the union of the theories by $E := E_1 \cup E_2$ and

[6] In retrospect, if one looks at the combination method for decision procedures by Schmidt-Schauß, then one sees that he used the free function symbols in the general unification problems just to encode linear constant restrictions.

the union of the signatures by $\Sigma := \Sigma_1 \cup \Sigma_2$. The theories E_1, E_2 are called the component theories of the combined theory E.

Before describing the algorithm in detail, we motivate its central steps and the ideas underlying these steps by examples. In these examples, E_1 will be the theory of a binary associative function symbol f, i.e., $E_1 := \{f(f(x,y),z) = f(x,f(y,z))\}$, and E_2 will be the free theory for the unary function symbol g and the constant symbol a, .e., $E_2 := \{g(x) = g(x),\ a = a\}$.

Assume that we are given an elementary E-unification problem Γ. First, we proceed in the same way as in the Nelson-Oppen procedure: using variable abstraction, we decompose Γ into a union of two pure unification problems, i.e., we compute an equivalent system of equations of the form[7] $\Gamma_1 \uplus \Gamma_2$, where Γ_i contains only terms over the signature Σ_i $(i = 1, 2)$.

Again, it is easy to see that an E-unifier σ of the original problem Γ is also an E-unifiers of the problems Γ_1 and Γ_2. By using an appropriate projection technique, which replaces alien subterms by variables, σ can be turned into an E_i-unifier of Γ_i (see Subsection 4.4 for more details). As in the case of the Nelson-Oppen procedure, the other direction need not be true: given solutions σ_1 and σ_2 of Γ_1 and Γ_2, it is not clear how to combine them into a solution of $\Gamma_1 \uplus \Gamma_2$.

An obvious problem that one may encounter is that the substitutions σ_1 and σ_2 may assign different terms to the same variable.

Example 3. Consider the decomposed problem $\Gamma_1 \uplus \Gamma_2$, where

$$\Gamma_1 := \{x =^? f(z,z)\} \quad \text{and} \quad \Gamma_2 := \{x =^? a\}.$$

The substitution $\sigma_1 := \{x \mapsto f(z,z)\}$ solves Γ_1 and $\sigma_2 := \{x \mapsto a\}$ solves Γ_2. Thus, there are conflicting assignment for the variable x, and it is not clear how to combine the two substitutions into a single one. In fact, in the example, there is no solution for the combined problem $\Gamma_1 \uplus \Gamma_2$.

This problem motivates the following step of the combination algorithm.

> *Theory labeling.* Given $\Gamma_1 \uplus \Gamma_2$, we (nondeterministically) introduce for each variable occurring in this problem a label 1 or 2. The label 1 for variable x indicates that x may be instantiated by solutions of the E_1-unification problem Γ_1, whereas it must be treated as a constant by solutions of the E_2-unification problem Γ_2. Label 2 is interpreted in the dual way.

In the example, the variable x must be assigned either label 1 or 2. In the first case, Γ_2 does not have a solution, whereas in the second case Γ_1 does not have a solution.

Unfortunately, this step introduces a new problem. Some of the new free "constants" generated by the labeling step may need to be identified by a solution of the subproblem, but this is no longer possible since they are constants, and thus cannot be replaced.

[7] The symbol "$\uplus$" indicates that this union is disjoint.

Example 4. Consider the decomposed problem $\Gamma_1 \uplus \Gamma_2$, where

$$\Gamma_1 := \{x =^? f(z,z), y =^? f(z,z)\} \text{ and } \Gamma_2 := \{g(x) =^? g(y)\}.$$

Obviously, Γ_1 can only be solved if both x and y obtain label 1. However, then Γ_2 does not have a solution since x and y are viewed as different constants in Γ_2. Nevertheless, $\Gamma_1 \uplus \Gamma_2$ has a solution, namely $\sigma := \{x \mapsto f(z,z),\ y \mapsto f(z,z)\}$.

There is, however, a simple solution to this problem.

> *Variable identification.* Before introducing the theory labeling, variables are (nondeterministically) identified.[8]

In the example, we just identify x and y (e.g., by replacing all occurrences of y by x). The resulting problem $\Gamma_2' = \{g(x) =^? g(x)\}$ is now obviously solvable.

After we have performed the identification and the labeling step, we can be sure that given solutions σ_1 and σ_2 of Γ_1 and Γ_2 have a disjoint domain, and thus it makes sense to consider the substitution $\sigma_1 \cup \sigma_2$. Nevertheless, this substitution need not be a solution of $\Gamma_1 \uplus \Gamma_2$, as illustrated by the following example.

Example 5. Consider the decomposed problem $\Gamma_1 \uplus \Gamma_2$, where

$$\Gamma_1 := \{x = f(z,z)\} \text{ and } \Gamma_2 := \{z =^? a\}.$$

We assume that no variables are identified and that x obtains label 1 and z label 2. Then $\sigma_1 := \{x \mapsto f(z,z)\}$ solves Γ_1 and $\sigma_2 := \{z \mapsto a\}$ solves Γ_2. However, $\sigma := \sigma_1 \cup \sigma_2 = \{x \mapsto f(z,z),\ z \mapsto a\}$ does not solve $\Gamma_1 \uplus \Gamma_2$ since $\sigma(x) = f(z,z)$ and $\sigma(f(z,z)) = f(a,a)$.

To avoid this kind of problem, we must iteratively apply the substitutions σ_1 and σ_2 to each other, i.e., consider the sequence σ_1, $\sigma_2 \circ \sigma_1$, $\sigma_1 \circ \sigma_2 \circ \sigma_1$,[9] etc. until it stabilizes. In the above example, the correct combined substitution would be $\{x \mapsto f(a,a),\ z \mapsto a\} = \sigma_2 \circ \sigma_1 = \sigma_1 \circ \sigma_2 \circ \sigma_1 = \ldots$. In general, this process need not terminate since there may be cyclic dependencies between the variable instantiations.

Example 6. Consider the decomposed problem $\Gamma_1 \uplus \Gamma_2$, where

$$\Gamma_1 := \{x = f(z,z)\} \text{ and } \Gamma_2 := \{z =^? g(x)\}.$$

We assume that no variables are identified and that x obtains label 1 and z label 2. Then $\sigma_1 := \{x \mapsto f(z,z)\}$ solves Γ_1 and $\sigma_2 := \{z \mapsto g(x)\}$ solves Γ_2. Because x is replaced by a term containing y and y by a term containing x, iterated application of the substitutions to each other does not terminate. In fact, it is easy to see that the combined problem $\Gamma_1 \uplus \Gamma_2$ does not have a solution.

[8] This is a step that also occurs in the nondeterministic variant of the Nelson-Oppen procedure.

[9] The symbol "$\circ$" denotes composition of substitution, where the substitution on the right is applied first.

In the combination procedure, such cyclic dependencies between solutions of the component problems are prohibited by the following step:

> *Linear ordering.* We (nondeterministically) choose a linear ordering $<$ on the variables occurring in $\Gamma_1 \uplus \Gamma_2$. Given a labeling of variables as explained above, we ask for solutions σ_i of Γ_i $(i = 1, 2)$ that respect the labeling in the sense introduced above, and satisfy the following additional condition: if a variable y with label j occurs in $\sigma_i(x)$, where x has label $i \neq j$, then $y < x$.

As a consequence of these steps, the E_i-unification problems obtained as output of the combination procedure are no longer elementary E_i-unification problems. Because of the labeling, they contain free constants (the variables with label $j \neq i$), and the linear ordering imposes additional restrictions on the possible solutions. We call such a problem an E_i-unification problem with linear constant restrictions.

Definition 2. *Let F be an equational theory. An* F-unification problem with linear constant restriction *is a quadruple $(\Gamma, X, C, <)$. The first component, Γ, is an elementary F-unification problem. X and C are disjoint finite sets such that $X \cup C$ is a superset of the set of variables occurring in Γ. The last component, $<$, is a linear ordering on $X \cup C$. A* solution *of $(\Gamma, X, C, <)$ is a substitution σ that satisfies the following conditions:*

1. *σ solves the elementary F-unification problem Γ;*
2. *σ treats all elements of C as constants, i.e., $\sigma(x) = x$ for all $x \in C$;*
3. *for all $x \in X$ and $c \in C$, if $x < c$, then c must not occur in $\sigma(x)$.*

We are now ready to give a formal description of the combination algorithm in Fig. 1. As before, we restrict the description to the combination of two component algorithms. It should be noted, however, that the generalization to $n > 2$ theories would be straightforward. Since Steps 2–4 are nondeterministic, the procedure actually generates a finite set of possible output pairs. The following proposition shows that the combination algorithm is sound and complete if used as a scheme for combining E_i-unification algorithms that decide solvability. A sketch of a proof will be given later on.

Proposition 3. *The input problem, Γ, has a solution iff there exists an output pair of the Combination Algorithm, $((\Gamma_1', Y_1, Y_2, <), (\Gamma_2', Y_2, Y_1, <))$, such that both components are solvable.*

4.3 Consequences

The straightforward generalization of Proposition 3 to $n \geq 2$ theories yields the following combination result for decision procedures.

Theorem 6. *Let $E_1, \ldots, E_n$ be equational theories over pairwise disjoint signatures such that solvability of E_i-unification problems with linear constant restrictions is decidable for $i = 1, \ldots, n$. Then unifiability is decidable for the combined theory $E := E_1 \cup \ldots \cup E_n$.*

Input: A finite set Γ of equations between $(\Sigma_1 \cup \Sigma_2)$-terms. The following steps are applied in consecutive order.

1. Decomposition.
Using variable abstraction, we compute an equivalent system $\Gamma_1 \uplus \Gamma_2$ where Γ_1 only contains pure Σ_1-terms and Γ_2 only contains pure Σ_2-terms.

2. Choose Variable Identification.
A partition Π of the set of variables occurring in $\Gamma_1 \uplus \Gamma_2$ is chosen, and for each equivalence class of Π a representative is selected. If y is the representative of $\pi \in \Pi$ and $x \in \pi$ we say that y is the representative of x. Let Y denote the set of all representatives. Now each variable is replaced by its representative. We obtain the new system $\Gamma_1' \uplus \Gamma_2'$.

3. Choose Theory Labeling.
A labeling function $Lab : Y \to \{1, 2\}$ is chosen. Let Y_1 and Y_2 respectively denote the set of variables with label 1 and 2.

4. Choose Linear Ordering.
A linear ordering "$<$" on Y is selected.

Output: The pair $((\Gamma_1', Y_1, Y_2, <), (\Gamma_2', Y_2, Y_1, <))$.
Each component $(\Gamma_i', Y_i, Y_j, <)$ is treated as an E_i-unification problem with linear constant restriction $(i = 1, 2)$.

Fig. 1. The Combination Algorithm

By "unifiability" we mean here solvability of elementary E-unification problems. Since, for each set Ω of free function symbols, solvability of unification problems with linear constant restriction in the free theory $F_\Omega = \{f(\ldots) = f(\ldots) \mid f \in \Omega\}$ is decidable (see below), the result of Theorem 6 can also be lifted to general E-unification problems. In fact, given a general E-unification problem, Γ, we just have to apply the theorem to the theories $E_1, \ldots, E_n, F_\Omega$ where Ω denotes the set of free function symbols occurring in Γ. In Section 5.1 we shall see that E-unification problems with linear constant restrictions can always be encoded as E-unification problems with free function symbols. As a consequence, Theorem 6 also holds for E-unification problems with linear constant restrictions, which yields a modularity result for unification with linear constant restrictions.

A simple analysis of the (nondeterministic) steps of the Combination Algorithm also provides us with the following complexity result, which is analogous to the one of Theorem 5:

Theorem 7. *If solvability of E_i-unification problems with linear constant restrictions is decidable in NP, then unifiability in the combined theory $E_1 \cup E_2$ is also decidable in NP.*

Although it was designed for the purpose of combining decision procedures, the Combination Algorithm can also be used to compute complete sets of unifiers modulo the union of equational theories.

Theorem 8. *Let $E_1, \ldots, E_n$ be equational theories over pairwise disjoint signatures, and let $E := E_1 \cup \ldots \cup E_n$ be their union. Assume that we have unification algorithms that compute, for each E_i-unification problem with linear constant restrictions, a finite complete set of E_i-unifiers ($i = 1, \ldots, n$). Then we can compute a finite complete set of E-unifiers for each elementary E-unification problem.*

The main idea for proving this theorem (sketched here for the case of $n = 2$ theories) is as follows. In the proof of soundness of the combination algorithm (see Section 4.4 below), we will show how an arbitrary pair (σ_1, σ_2) of solutions of an output pair of the combination algorithm can be combined into a solution $\sigma_1 \oplus \sigma_2$ of the input problem (see also Example 5). Given a single output pair $((\Gamma_1', Y_1, Y_2, <), (\Gamma_2', Y_2, Y_1, <))$, one can compute complete sets of unifiers for the two component problems, and then combine the elements of these complete sets in all possible ways. If this is done for all output pairs, then the set of all combined solutions obtained this way is a complete set of unifiers for the input problem (see [6] for details).

The last two results can again be lifted from elementary unification problems to general unification problems and to unification problems with linear constant restrictions in the combined theory, which provides us with a modularity result.

In order to apply these general combination results to specific theories, one needs algorithms that can solve unification problems *with linear constant restrictions* for these theories. For regular theories, an algorithm for computing complete sets of unifiers for unification with constants can be used to obtain an algorithm for computing complete sets of unifiers for unification with linear constant restrictions: just remove the unifiers violating the constant restrictions from the complete set. In particular, since the free theory is obviously regular, one can test solvability of a unification problem with linear constant restrictions in the free theory by computing the most general unifier, and then checking whether this unifier satisfies the constant restrictions. For non-regular theories, this simple way of proceeding is not possible. However, the constant elimination procedures required by the approach of Schmidt-Schauß can be used to turn complete sets of unifiers for unification with constants into complete sets of unifiers for unification with linear constant restrictions (see [6], Section 5.2, for details).

With respect to decision procedures, it has turned out that, for several interesting theories (e.g., the theory AC of an associative-commutative symbol or the theory ACI of an associative-commutative-idempotent symbol), the known decision procedures for unification with constants can easily be modified into algorithms for unification with linear constant restrictions [3]. In particular, it is easy to show that Theorem 7 applies to AC and ACI, which yields a simple proof that the decision problem for general AC- and ACI-unification is in NP. For the theory A of an associative function symbol, decidability of unification problems with linear constant restrictions is an easy consequence (see [3]) of a result by Schulz [61] on a generalization of Makanin's decision procedure. As a

consequence, general A-unification is also decidable (this problem had been open before the development of the combination algorithm presented above).

There are, however, also theories for which unification with linear constant restrictions is considerably harder than unification with constants. For example, it can be shown [1] that Boolean unification with linear constant restrictions is PSPACE-complete whereas Boolean unification with constants is "only" Π_2^p-complete. Until now, it is not known whether there exists an equational theory for which unification with constants is decidable, but unification with linear constant restrictions is undecidable.

4.4 Correctness

In the remainder of this section, we sketch how to prove Proposition 3. The full proof can be found in [6].

To show *soundness* of the Combination Algorithm, it suffices to show that, for each output pair $((\Gamma_1', Y_1, Y_2, <), (\Gamma_2', Y_2, Y_1, <))$, a given pair of solutions (σ_1, σ_2) of the two components can be combined into a solution σ of $\Gamma_1' \uplus \Gamma_2'$, which is treated as an elementary E-unification problem here. In fact, this obviously implies that σ can be extended to a solution of the input problem Γ.

The combined solution σ is defined by induction on the linear ordering $<$. Assume that σ is defined for all variables $y \in Y$ that are smaller than $z \in Y$ with respect to $<$. Without loss of generality we may assume that $z \in Y_1$ has label 1 and that $\sigma_1(z)$ does not contain any variables from Y_1. Since σ_1 satisfies the linear constant restrictions, it follows that all labeled variables y occurring in $\sigma_1(z)$ are smaller than z with respect to "$<$", which implies that $\sigma(y)$ is defined by induction hypothesis. We define $\sigma(z) := \sigma(\sigma_1(z))$. It is easy to see that the substitution σ obtained in this way is an instance of both σ_1 and σ_2. It follows that σ is an E_i-unifier, and hence an E-unifier, of Γ_i' $(i = 1, 2)$. Consequently, σ is a solution of $\Gamma_1' \uplus \Gamma_2'$.

It is more difficult to prove the *completeness* part of Proposition 3. Basically, the proof proceeds as follows. A given solution σ of Γ is used to define suitable choices in the nondeterministic steps of the Combination Algorithm, i.e., choices that lead to an output pair where both components are solvable:

- at the variable identification step, two variables x and y are identified iff $\sigma(x) =_E \sigma(y)$. Obviously σ is a solution of the system $\Gamma_1' \uplus \Gamma_2'$ reached after this identification.
- at the labeling step, a representative y receives label 1 iff $\sigma(x)$ has a symbol from Σ_1 as topmost function symbol.
- the linear ordering "$<$" that is chosen is an arbitrary extension of the partial ordering that is induced by the subterm relationship of the σ-values of representatives.

However, this way of proceedings is correct only if the solution σ of the input problem is assumed to be normalized in a particular way. In [2], using so-called "unfailing completion," a (possibly infinite) canonical rewrite system R for the

combined theory E is defined. For each variable x in the system $\Gamma_1' \uplus \Gamma_2'$ it is then assumed that $\sigma(x)$ is in R-normal form. Another possibility is to assume that the terms $\sigma(x)$ are in the so-called layer-reduced form [60, 42]. In principle, this normal form is obtained by applying collapse-equations as much as possible.

It remains to find, given a normalized solution σ, suitable solutions σ_1 and σ_2 of the output pair determined by the choices induced by σ. To define these solutions, a "projection technique" is introduced that transforms possibly mixed solution terms of the form $\sigma(y)$ to a pure Σ_i-terms $\sigma_i(y)$. Basically, to define $\sigma_i(y)$, "alien" subterms of $\sigma(y)$ (i.e., maximal subterms starting with a symbol not belonging to Σ_i) are replaced by new variables, while ensuring that E-equivalent subterms are replaced by the same variable. If $\sigma(y)$ itself is alien, then $\sigma_i(y) := y$, which ensures that variables with label $j \neq i$ are treated as free constants.

5 The logical and algebraic perspective

In this section, we describe the problem of combining unification algorithms from a more logical and algebraic point of view. This leads to a modified description of the combination algorithm and to a new proof of its correctness. In the next section, we will show that the techniques developed in the present section allow us to lift the combination methodology to more general classes of constraints.

5.1 A logical reformulation of the Combination Algorithm

Theorems 1 and 2 show that elementary E-unification problems and E-unification problems with constants correspond to natural classes of logical decision problems. The question arises whether this classification can be extended to general E-unification problems and to E-unification problems with linear constant restrictions. The following theorem, which was first proved in [6], gives a positive answer to this question. In particular, it states that both problems correspond to the same class of logical formulae.

Theorem 9. *Let E be an equational theory with signature Σ, and V a countably infinite set of variables. Then the following statements are equivalent:*

1. *Solvability of E-unification problems with linear constant restrictions is decidable.*
2. *The positive theory of E is decidable.*
3. *The positive theory of $\mathcal{T}(\Sigma, V)/=_E$ is decidable.*
4. *Solvability of general E-unification problems is decidable.*

From a practical point of view, the theorem is interesting because it shows that any theory that can reasonably be integrated in a universal deductive machinery via unification can also be combined with other such theories. In fact, as mentioned at the beginning of Section 4.1, such an integration usually requires an algorithm for *general* unification. The theorem shows that such an algorithm also

makes sure that the precondition for our combination method to apply—namely, the existence of an algorithm for unification with linear constant restrictions—are satisfied.[10]

Theorem 9, together with our combination result for decision procedures, yields the following modularity result for the decidability of positive theories:

Theorem 10. *Let $E_1, \ldots, E_n$ be equational theories over disjoint signatures. Then the positive theory of $E_1 \cup \ldots \cup E_n$ is decidable iff the positive theories of the component theories E_i are decidable, for $i = 1, \ldots, n$.*

In the following, we motivate the equivalences stated in Theorem 9 by sketching how the respective problems can be translated into each other (see [6] for a detailed proof of the theorem):

- Any E-unification problem with linear constant restrictions $(\Gamma, X, C, <)$ can be translated into a positive Σ-sentence ϕ_Γ as follows: both variables (i.e., elements of X) and free constants (i.e., elements of C) are treated as variables in this formula; the matrix of ϕ_Γ is the conjunction of all equations in Γ; and in the quantifier prefix, the elements of X are existentially quantified, the elements of C are universally quantified, and the order of the quantifications is given by the linear ordering $<$.
- The equivalence between 2) and 3) is due to the well-known fact that the E-free algebra with countably many generators is canonical for the positive theory of E [48], i.e., a positive sentence is valid in $\mathcal{T}(\Sigma, V)/=_E$ iff it is valid in all models of E.
- Given a positive Σ-sentence ϕ, one first removes universal quantifiers by Skolemization. The positive existential sentence obtained this way may contain additional free function symbols, the Skolem functions. It can be transformed into a disjunction of conjunctive positive existential sentences, and each of the disjuncts can obviously be translated into a general E-unification problem.
- The combination method described in Section 4 can be used to reduce solvability of a given general E-unification problem to solvability of E-unification problems with linear constant restrictions.

As an example, consider the free theory $F_{\{g\}} := \{g(x) = g(x)\}$, and the $F_{\{g\}}$-unification problem with constants $\{x =^? g(c)\}$. If we add the constant restriction $x < c$, then this problem is not solvable (since any solution must substitute x by the term $g(c)$, which contains the constant c). However, under the restriction $c < x$ the problem is solvable. The following are the positive sentences and general unification problems obtained by translating these two unification

[10] Strictly speaking, the theorem makes this statement only for decision procedures. In [6] it is shown, however, that the equivalence between general unification and unification with linear constant restrictions also holds with respect to algorithms that compute complete sets of unifiers.

problems with linear constant restrictions:

unification with lcr	*positive sentence*	*general unification*
$\{x =^? g(c)\},\ x < c$	$\exists x.\forall y.\ x = g(y)$	$\{x =^? g(h(x))\}$
$\{x =^? g(c)\},\ c < x$	$\forall y.\exists x.\ x = g(y)$	$\{x =^? g(d)\}$

For example, $\exists x.\forall y.\ x = g(y)$ is not valid in all models of $F_{\{g\}}$ since this formula says that g must be a constant function, which obviously does not follow from $F_{\{g\}}$. Correspondingly, $\{x = g(h(x))\}$ does not have a solution because it causes an occur-check failure during syntactic unification.

Returning now to the combination problem, let E_1 and E_2 be two nontrivial equational theories over disjoint signatures Σ_1 and Σ_2, let $E := E_1 \cup E_2$ denote the union of the theories and $\Sigma := \Sigma_1 \cup \Sigma_2$ the union of the signatures. Using the correspondence between elementary E-unification problems and existentially quantified conjunctions of equations for the input of the algorithm, and the correspondence between E_i-unification problems with linear constant restriction and positive sentences for the output components we obtain the reformulation of the Combination Algorithm shown in Fig. 2. The advantage of the new formulation is that it does no longer rely on notions and concepts that are specific to unification problems modulo equational theories, such as linear constant restrictions, which are quite technical restrictions on the form of the allowed solutions. Correctness follows from the following proposition.

Proposition 4. *The input sentence $\exists \boldsymbol{u}.\,\gamma$ holds in the combined quotient term algebra $\mathcal{T}(\Sigma_1 \cup \Sigma_2, V)/{=_{E_1 \cup E_2}}$ iff there exists an output pair (α, β) such that α holds in $\mathcal{T}(\Sigma_1, V)/{=_{E_1}}$ and β holds in $\mathcal{T}(\Sigma_2, V)/{=_{E_2}}$.*

Since the new combination algorithm is just a reformulation of the earlier version, Proposition 4 is a trivial consequence of Proposition 3.

The remainder of this section is devoted to giving an independent correctness proof for the logical version of the Combination Algorithm. The new proof will have some significant advantages: it is more abstract and less technical, and thus easier to generalize to larger classes of constraints.

5.2 Fusions of free algebras

The proof of soundness of the Nelson-Oppen combination procedure that we have presented in Section 3 depends on a very simple algebraic construction: the fusion of structures. Our goal is to adapt this algebraic approach to the task of proving correctness of the (logical reformulation of the) combination procedure for unification algorithms. At first sight, the input problems considered in the case of unification look like a special case of the problems accepted by the Nelson-Oppen procedure: they are (existentially quantified) conjunctions of equations.[11]

[11] The Nelson-Oppen procedure additionally allows for negation and for non-equational atoms.

Input: A $(\Sigma_1 \cup \Sigma_2)$-sentence of the form $\exists \boldsymbol{u}.\, \gamma$, where γ is a conjunction of equations between $(\Sigma_1 \cup \Sigma_2)$-terms and $\boldsymbol{u}$ is a finite sequence consisting of the variables occurring in γ. The following steps are applied in consecutive order.

1. Decomposition.
Using variable abstraction, compute an equivalent sentence $\exists \boldsymbol{v}.\, (\gamma_1 \wedge \gamma_2)$*, where* γ_i *is a conjunction of equations between pure* Σ_i*-terms for* $i = 1, 2$*.*

2. Choose Variable Identification.
A partition Π *of the set of variables occurring in* $\boldsymbol{v}$ *is chosen, and for each equivalence class of* Π *a representative is selected. If* v *is the representative of* $\pi \in \Pi$ *and* $u \in \pi$*, then we say that* v *is the representative of* u*. Let* W *denote the set of all representatives. Now each variable is replaced by its representative both in the quantifier prefix and in the matrix. Multiple quantifications over the same variable in the prefix are discarded. We obtain the new sentence* $\exists \boldsymbol{w}.\, (\gamma_1' \wedge \gamma_2')$*.*

3. Choose Labeling.
A labeling function $Lab : W \to \{1, 2\}$ *is chosen.*

4. Choose Linear Ordering.
A linear ordering "$<$*" on* W *is selected.*

Output: The pair

$$\alpha = \forall \boldsymbol{u}_1.\exists \boldsymbol{v}_1. \cdots \forall \boldsymbol{u}_k.\exists \boldsymbol{v}_k.\, \gamma_1' \text{ and } \beta = \exists \boldsymbol{u}_1.\forall \boldsymbol{v}_1. \cdots \exists \boldsymbol{u}_k.\forall \boldsymbol{v}_k.\, \gamma_2'.$$

Here $\boldsymbol{u}_1 \boldsymbol{v}_1 \ldots \boldsymbol{u}_k \boldsymbol{v}_k$ is the unique re-ordering of W along $<$. The sequences $\boldsymbol{u}_i$ ($\boldsymbol{v}_i$) represent the blocks of variables with label 1 (label 2).

Fig. 2. The Combination Algorithm (Logical Reformulation)

The main difference between the two combination problems lies in the semantics of the constraints. In the case treated by Nelson and Oppen, the input constraint must be satisfied in *some* model of the combined theory $T_1 \cup T_2$, whereas in the case of unification algorithms the input constraint must be satisfied in the *free* model of the combined theory $E_1 \cup E_2$. In the proof of correctness this means that, for the Nelson-Oppen procedure, it is sufficient to show that the input constraint can be satisfied in an arbitrary fusion of a model of T_1 with a model of T_2. In the unification case, we must make sure that this fusion is in fact the $(E_1 \cup E_2)$-free algebra with countably infinitely many generators. Thus, given the E_1- and E_2-free algebras $\mathcal{B}_1 := \mathcal{T}(\Sigma_1, V)/=_{E_1}$ and $\mathcal{B}_2 := \mathcal{T}(\Sigma_2, V)/=_{E_2}$, respectively, we want to construct a fusion of both algebras that is (isomorphic to) the $(E_1 \cup E_2)$-free algebra $\mathcal{B} := T(\Sigma_1 \cup \Sigma_2, V)/=_{E_1 \cup E_2}$. This construction will be called the *amalgamation construction*.

In the sequel, as always in this section, we assume that the signatures Σ_1 and Σ_2 are disjoint, and that the theories E_1 and E_2 are nontrivial. For simplicity we shall identify each variable $x \in V$ with its equivalence class w.r.t. E_i in $\mathcal{B}_i$, i.e., write again x for the E_i-class $[x]_{E_i} = \{t \in T(\Sigma_i, V) \mid t =_{E_i} x\}$.

The construction starts with a preparatory step where we extend $\mathcal{B}_i$ to an E_i-free algebra $\mathcal{B}_i^\infty$ of the form $T(\Sigma_i, V \cup Y_i)/{=_{E_i}}$ where Y_i denotes a countably infinite set of additional variables ($i = 1, 2$). Since the sets $V \cup Y_i$ and V have the same cardinality, B_1^∞ and B_2^∞ are isomorphic to $\mathcal{B}_1$ and $\mathcal{B}_2$, respectively. We assume (without loss of generality) that $B_1^\infty \cap B_2^\infty = V$. These two algebras (as well as details of the amalgamation construction) are depicted in Fig. 3.

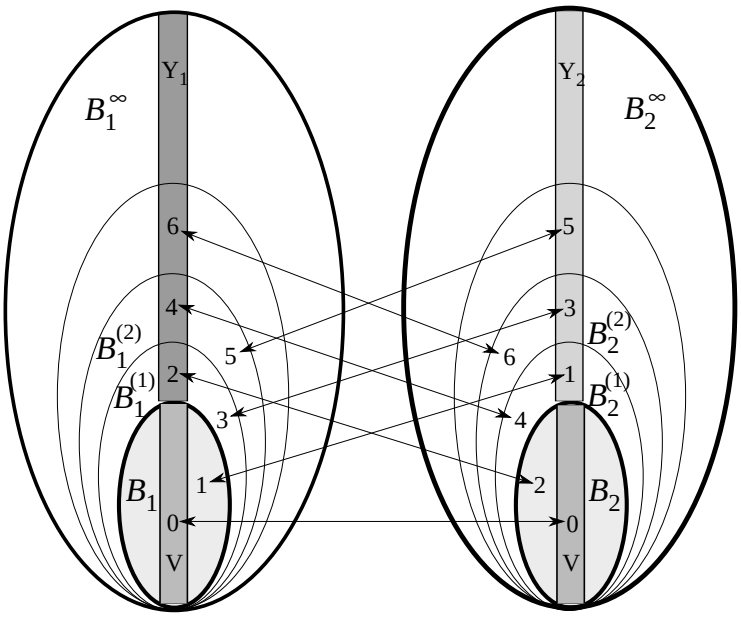


Fig. 3. The amalgamation construction.

We shall now construct a bijection between the domains B_1^∞ and B_2^∞ of the extended algebras $\mathcal{B}_1^\infty$ and $\mathcal{B}_2^\infty$. This bijection will then be used to define a fusion of $\mathcal{B}_1^\infty$ and $\mathcal{B}_2^\infty$ (see Lemma 1), which is also a fusion of $\mathcal{B}_1$ and $\mathcal{B}_2$. Note, however, that we cannot use an arbitrary bijection between B_1^∞ and B_2^∞ since we want this fusion to be the $(E_1 \cup E_2)$-free algebra with countably infinitely many generators.

In the following, let us call an element of $B_i^\infty \setminus (V \cup Y_i)$ a *non-atomic* element of $\mathcal{B}_i^\infty$. The elements of $V \cup Y_i$ are called *atomic*. The crucial property that we want to obtain is that *non-atomic* elements of one side are always mapped to *atomic* elements of the other side. The definition of the bijection proceeds in an infinite series of zig-zag steps: at each step an existing partial bijection is extended by adding a new partial bijection with domain and image sets disjoint to the sets already used.

In step 0 we use the identity mapping on V to obtain a bijection between the common set of generators of both sides (see areas 0 in Fig. 3). We say that the elements in V of both sides are now fibered.

In step 1 we assign suitable images to the elements of $B_1 \setminus V$ (see area 1 on the left-hand side). To this end, we select a set of atoms $Y_2^{(1)} \subset Y_2$ with the same cardinality as $B_1 \setminus V$ (area 1 on the right-hand side represents $Y_2^{(1)}$). The existing partial bijection is extended by adding a bijection between $B_1 \setminus V$ and $Y_2^{(1)}$ (indicated by a double arrow between the areas 1). We say that the elements in $B_1 \setminus V$ and $Y_2^{(1)}$ are now fibered as well. In step 1 and in the sequel, whenever we select a set of new atoms, we leave an infinite set of atoms untouched, which are thus available in subsequent steps of the construction.

In the symmetric step 2 we add a bijection between $B_2 \setminus V$ (area 2, right-hand side) and a suitable set of new atoms $Y_1^{(1)} \subset Y_1$ (area 2, left-hand side). With this step we say that now the elements in $B_2 \setminus V$ and $Y_1^{(1)}$ are fibered as well.

For $i = 1, 2$, let $\mathcal{B}_i^{(1)}$ denote the subalgebra of $\mathcal{B}_i^\infty$ that is generated by $V \cup Y_i^{(1)}$. The elements of $\mathcal{B}_1^{(1)}$ that do not yet have an image are fibered in step 3 using a fresh set of atoms $Y_2^{(2)}$ of the right-hand side (areas 3); in step 4 the elements of $B_2^{(1)}$ that do not yet have images are fibered using a fresh set of atoms $Y_1^{(2)}$ of left-hand side (areas 4).

For $i = 1, 2$, let $\mathcal{B}_i^{(2)}$ denote the subalgebra of $\mathcal{B}_i^\infty$ that is generated by $V \cup Y_i^{(1)} \cup Y_i^{(2)}$. We continue in the same way as above (areas 5, 6), etc. The construction determines for $i = 1, 2$ an ascending tower of Σ_i-subalgebras

$$\mathcal{B}_i = \mathcal{B}_i^{(0)} \subseteq \mathcal{B}_i^{(1)} \subseteq \mathcal{B}_i^{(2)} \subseteq \ldots$$

of $\mathcal{B}_i^\infty$. For simplicity we assume that the construction eventually covers each atom of both sides, hence we have $B_i^\infty = \bigcup_{k=0}^\infty B_i^{(k)}$. Since the limit bijection can be read in two directions we now have two inverse bijections

$$h_{1-2} : \mathcal{B}_1^\infty \to \mathcal{B}_2^\infty \quad \text{and} \quad h_{2-1} : \mathcal{B}_2^\infty \to \mathcal{B}_1^\infty.$$

As in the proof of Lemma 1, these bijections can be used to carry the Σ_i-structure of $\mathcal{B}_i^\infty$ to $\mathcal{B}_j^\infty$ (where $\{i, j\} = \{1, 2\}$). Let f be an n-ary function symbol of Σ_i and $b_1, \ldots, b_n \in B_j^\infty$. We define

$$f^{\mathcal{B}_j^\infty}(b_1, \ldots, b_n) := h_{i-j}(f^{\mathcal{B}_i^\infty}(h_{j-i}(b_1), \ldots, h_{j-i}(b_n))).$$

With this definition, the mappings h_{1-2} and h_{2-1} are inverse isomorphisms between the $(\Sigma_1 \cup \Sigma_2)$-algebras obtained from $\mathcal{B}_1^\infty$ and $\mathcal{B}_2^\infty$ by means of the above signature expansion. For this reason, it is irrelevant which of the two algebras we take as the combined algebra. We take, say, the $(\Sigma_1 \cup \Sigma_2)$-algebra obtained from $\mathcal{B}_1^\infty$, and denote it by $\mathcal{B}_1 \oplus \mathcal{B}_2$.

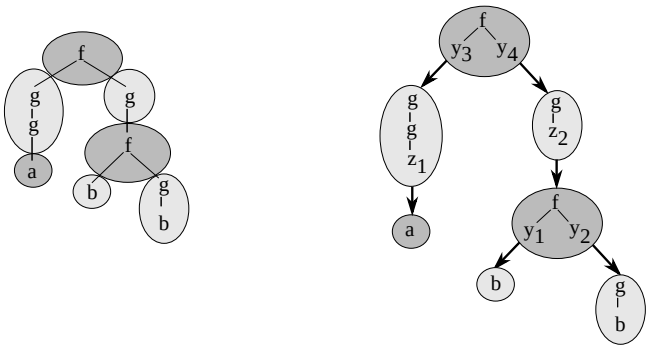
Recall that $\mathcal{B}_i$ and $\mathcal{B}_i^\infty$ are Σ_i-isomorphic algebras since both are free over a countably infinite set of generators for the class of all models of E_i. In addition, the construction makes sure that $\mathcal{B}_1 \oplus \mathcal{B}_2$ is Σ_i-isomorphic to $\mathcal{B}_i^\infty$ $(i = 1, 2)$, which yields the following lemma:

Lemma 2. *$\mathcal{B}_1 \oplus \mathcal{B}_2$ is a fusion of $\mathcal{B}_1$ and $\mathcal{B}_2$.*

More interestingly, the following theorem (whose proof can be found in [7]) shows that we have indeed found the desired description of $T(\Sigma_1 \cup \Sigma_2, V)/{=_{E_1 \cup E_2}}$ as a fusion of the component algebras $\mathcal{B}_1 = T(\Sigma_1, V)/{=_{E_1}}$ and $\mathcal{B}_2 = T(\Sigma_2, V)/{=_{E_2}}$:

Theorem 11. *$\mathcal{B}_1 \oplus \mathcal{B}_2$ is (isomorphic to) the $(E_1 \cup E_2)$-free algebra over a countably infinite set of generators.*

At first sight, it is perhaps not easy to see the relationship between the amalgamation construction and the usual description of $T(\Sigma_1 \cup \Sigma_2, V)/_{E_1 \cup E_2}$ in terms of $=_{(E_1 \cup E_2)}$-equivalence classes of terms (i.e., finite trees). In order to illustrate this connection, let us look at the simplest possible case where we combine two absolutely free algebras, i.e., where E_1 and E_2 are free theories. Let us assume that $\Sigma_1 = \{f, a\}$ and $\Sigma_2 = \{g, b\}$, where f is binary, g is unary and a, b are constants. The following figure depicts, on the left-hand side, an element of the combined domain using the conventional description as a finite tree. Subparts belonging to distinct signatures are highlighted accordingly.



The "leaf" elements a and $b, g(b)$ correspond to elements of $\mathcal{B}_1$ and $\mathcal{B}_2$ that are fibered in steps 1 and 2 of the construction, say, with atoms z_1 and y_1, y_2, respectively. Thus, the subtree $f(b, g(b))$ corresponds to the element $f^{\mathcal{B}_1^\infty}(y_1, y_2)$ of $B_1^{(1)}$, and $g(g(a))$ to the element $g^{\mathcal{B}_2^\infty}(g^{\mathcal{B}_2^\infty}(z_1))$ of $B_2^{(1)}$. These elements are fibered with new atoms (say z_2 and y_3) in the steps 2 and 3. The subtree $g(f(b, g(b)))$ corresponds to an element of $B_2^{(2)}$, which is fibered by a new atom (say y_4) in step 6. Finally, the complete tree $f(g(g(a)), g(f(b, g(b))))$ corresponds to an element of $B_1^{(3)}$. On the right-hand side of the figure, we have represented all elements of the fusion that are involved in the representation of the complete tree and made the fibering bijections explicit using arrows. Due to the inductive form of the construction, the elements of the fusion can be considered as generalized "finite trees" where nodes represent elements of the two components, and links represent ordered pairs of the fibering function.

If E_1 and E_2 are more interesting equational theories, the relationship between a given mixed term and the corresponding element of $\mathcal{B}_1 \oplus \mathcal{B}_2$ may be less obvious. For example, if E_2 contains the collapse axiom $g(x) = x$, then

$g(g(a))$ is equivalent to a, and thus the corresponding element belongs to B_1, and not to $B_2^{(1)}$. A similar phenomenon occurs in the presence of non-regular axioms. For example, if E_1 is the free theory, then $f(b, g(b))$ corresponds to an element of $B_1^{(1)}$. However, if E_1 contains the (non-regular) axiom $f(x, y) = a$, then $f(b, g(b))$ is equivalent to a, and the corresponding element belongs to B_1. This issue is closely related to the fact that, in the proof of completeness of Proposition 3, we needed a normalized substitution. Given a mixed term that is normalized by the (possibly infinite) canonical rewrite system R, the simple syntactic correspondence between subtrees of this term and elements of $\mathcal{B}_1 \oplus \mathcal{B}_2$ holds also for theories that are not regular and collapse-free.

In the next subsection, we will use the new description of the combined algebra $\mathcal{T}(\Sigma_1 \cup \Sigma_2, V)/{=_{E_1 \cup E_2}}$ as a fusion $\mathcal{B}_1 \oplus \mathcal{B}_2$ to show correctness of the combination algorithm in its logical reformulation.

5.3 Correctness of the Combination Algorithm

First, we show soundness of the Combination Algorithm (logical formulation). In the following, boldface letters like $\boldsymbol{u}, \boldsymbol{v}$ and $\boldsymbol{b}, \boldsymbol{d}$ (possibly with subscripts) will respectively denote finite sequences of variables and algebra elements. An expression like $\boldsymbol{b} \in \boldsymbol{B}$ expresses that $\boldsymbol{b}$ is a sequence of elements of B, which denotes the carrier set of the algebra $\mathcal{B}$. We denote by $h(\boldsymbol{b})$ the result of applying the homomorphism h to the sequence $\boldsymbol{b}$, i.e., the sequence consisting of the components $h(b)$ for all components b of $\boldsymbol{b}$.

Lemma 3. *Let $\exists \boldsymbol{u}.\, \gamma$ be an input sentence of the Combination Algorithm. Then $\mathcal{B}_1 \oplus \mathcal{B}_2 \models \exists \boldsymbol{u}.\, \gamma$ if $\mathcal{B}_1 \models \alpha$ and $\mathcal{B}_2 \models \beta$ for some output pair (α, β).*

Proof. Since $\mathcal{B}_1$ and $\mathcal{B}_1^\infty$ are isomorphic Σ_i-algebras, we know that $\mathcal{B}_1^\infty \models \alpha$. Accordingly, we also have $\mathcal{B}_2^\infty \models \beta$. More precisely, this means

$$(*) \quad \mathcal{B}_1^\infty \models \forall \boldsymbol{u}_1.\exists \boldsymbol{v}_1. \cdots \forall \boldsymbol{u}_k.\exists \boldsymbol{v}_k.\, \gamma_1'(\boldsymbol{u}_1, \boldsymbol{v}_1, \ldots, \boldsymbol{u}_k, \boldsymbol{v}_k),$$
$$(**) \quad \mathcal{B}_2^\infty \models \exists \boldsymbol{u}_1.\forall \boldsymbol{v}_1. \cdots \exists \boldsymbol{u}_k.\forall \boldsymbol{v}_k.\, \gamma_2'(\boldsymbol{u}_1, \boldsymbol{v}_1, \ldots, \boldsymbol{u}_k, \boldsymbol{v}_k).$$

Because of the existential quantification over $\boldsymbol{u}_1$ in $(**)$, there exists a sequence $\boldsymbol{b}_1 \in \boldsymbol{B}_2^\infty$ such that

$$(***) \quad \mathcal{B}_2^\infty \models \forall \boldsymbol{v}_1. \cdots \exists \boldsymbol{u}_k.\forall \boldsymbol{v}_k.\, \gamma_2'(\boldsymbol{b}_1, \boldsymbol{v}_1, \ldots, \boldsymbol{u}_k, \boldsymbol{v}_k).$$

We consider $\boldsymbol{a}_1 := h_{2-1}(\boldsymbol{b}_1)$. Because of the universal quantification over $\boldsymbol{u}_1$ in $(*)$ we have

$$\mathcal{B}_1^\infty \models \exists \boldsymbol{v}_1. \cdots \forall \boldsymbol{u}_k.\exists \boldsymbol{v}_k.\, \gamma_1'(\boldsymbol{a}_1, \boldsymbol{v}_1, \ldots, \boldsymbol{u}_k, \boldsymbol{v}_k).$$

Because of the existential quantification over $\boldsymbol{v}_1$ in this formula there exists a sequence $\boldsymbol{c}_1 \in \boldsymbol{B}_1^\infty$ such that

$$\mathcal{B}_1^\infty \models \forall \boldsymbol{u}_2.\exists \boldsymbol{v}_2. \cdots \forall \boldsymbol{u}_k.\exists \boldsymbol{v}_k.\, \gamma_1'(\boldsymbol{a}_1, \boldsymbol{c}_1, \boldsymbol{u}_2, \boldsymbol{v}_2, \ldots, \boldsymbol{u}_k, \boldsymbol{v}_k).$$

We consider $\boldsymbol{d}_1 := h_{1-2}(\boldsymbol{c}_1)$. Because of the universal quantification over $\boldsymbol{v}_1$ in $(***)$ we have

$$\mathcal{B}_2^\infty \models \exists \boldsymbol{u}_2.\forall \boldsymbol{v}_2.\cdots\exists \boldsymbol{u}_k.\forall \boldsymbol{v}_k.\,\gamma_2'(\boldsymbol{b}_1,\boldsymbol{d}_1,\boldsymbol{u}_2,\boldsymbol{v}_2,\ldots,\boldsymbol{u}_k,\boldsymbol{v}_k).$$

Iterating this argument, we thus obtain

$$\mathcal{B}_1^\infty \models \gamma_1'(\boldsymbol{a}_1,\boldsymbol{c}_1,\ldots,\boldsymbol{a}_k,\boldsymbol{c}_k),$$
$$\mathcal{B}_2^\infty \models \gamma_2'(\boldsymbol{b}_1,\boldsymbol{d}_1,\ldots,\boldsymbol{b}_k,\boldsymbol{d}_k),$$

where $\boldsymbol{a}_i = h_{2-1}(\boldsymbol{b}_i)$ and $\boldsymbol{d}_i = h_{1-2}(\boldsymbol{c}_i)$ (for $1 \leq i \leq k$). Since h_{1-2} and h_{2-1} are inverse $(\Sigma_1 \cup \Sigma_2)$-isomorphisms we also know that

$$\mathcal{B}_1^\infty \models\ \gamma_2'(\boldsymbol{a}_1,\boldsymbol{c}_1,\ldots,\boldsymbol{a}_k,\boldsymbol{c}_k).$$

It follows that

$$\mathcal{B}_1 \oplus \mathcal{B}_2 \models\ \gamma_1'(\boldsymbol{a}_1,\boldsymbol{c}_1,\ldots,\boldsymbol{a}_k,\boldsymbol{c}_k) \wedge\ \gamma_2'(\boldsymbol{a}_1,\boldsymbol{c}_1,\ldots,\boldsymbol{a}_k,\boldsymbol{c}_k).$$

Obviously, this implies that $\mathcal{B}_1 \oplus \mathcal{B}_2 \models \exists \boldsymbol{v}.\ (\gamma_1' \wedge \gamma_2')$, i.e., the sentences obtained after Step 1 of the algorithm holds in $\mathcal{B}_1 \oplus \mathcal{B}_2$. It is easy to see that this implies that $\mathcal{B}_1 \oplus \mathcal{B}_2 \models \exists \boldsymbol{u}.\,\gamma$. □

Before we can show completeness of the decomposition algorithm, we need one more prerequisite. The following lemma characterizes validity of positive sentences in free algebras in terms of satisfying assignments. The proof of this lemma, which uses the well-known fact that validity of positive formulae is preserved under surjective homomorphisms, is not difficult and can be found in [7] for the more general case of quasi-free structures.

Lemma 4. *Let $\mathcal{A} = \mathcal{T}(\Delta, V)/=_E$ be the E-free Δ-algebra over the countably infinite set of generators V, and let*

$$\gamma = \forall \boldsymbol{u}_1.\exists \boldsymbol{v}_1.\cdots\forall \boldsymbol{u}_k.\exists \boldsymbol{v}_k.\,\varphi(\boldsymbol{u}_1,\boldsymbol{v}_1,\ldots,\boldsymbol{u}_k,\boldsymbol{v}_k)$$

be a positive Δ-sentence. Then the following conditions are equivalent:

1. $\mathcal{A} \models \forall \boldsymbol{u}_1.\exists \boldsymbol{v}_1.\cdots\forall \boldsymbol{u}_k.\exists \boldsymbol{v}_k.\,\varphi(\boldsymbol{u}_1,\boldsymbol{v}_1,\ldots,\boldsymbol{u}_k,\boldsymbol{v}_k)$.
2. *There exist tuples $\boldsymbol{x}_1 \in \boldsymbol{V}, \boldsymbol{e}_1 \in \boldsymbol{A},\ldots,\boldsymbol{x}_k \in \boldsymbol{V}, \boldsymbol{e}_k \in \boldsymbol{A}$ and finite subsets $Z_1,\ldots,Z_k$ of V such that*
   (a) $\mathcal{A} \models \varphi(\boldsymbol{x}_1,\boldsymbol{e}_1,\ldots,\boldsymbol{x}_k,\boldsymbol{e}_k)$,
   (b) *all generators occurring in the tuples $\boldsymbol{x}_1,\ldots,\boldsymbol{x}_k$ are distinct,*
   (c) *for all $j, 1 \leq j \leq k$, the components of $\boldsymbol{e}_j$ are generated by Z_j, i.e., they belong to $\mathcal{T}(\Delta, Z_j)/=_E$*
   (d) *for all $j, 1 < j \leq k$, no component of $\boldsymbol{x}_j$ occurs in $Z_1 \cup \ldots \cup Z_{j-1}$.*

Using this lemma, we can now prove completeness of the Combination Algorithm (logical reformulation).

Lemma 5. *Let $\exists \boldsymbol{u}.\,\gamma$ be an input sentence of the Combination Algorithm. If $\mathcal{B}_1 \oplus \mathcal{B}_2 \models \exists \boldsymbol{u}.\,\gamma$ then there exists an output pair with components α and β such that $\mathcal{B}_1 \models \alpha$ and $\mathcal{B}_2 \models \beta$.*

Proof. Assume that $\mathcal{B}_1^\infty \simeq \mathcal{B}_1 \oplus \mathcal{B}_2 \models \exists \boldsymbol{u}_0.\,\gamma_0$.[12] Obviously, this implies that $\mathcal{B}_1^\infty \models \exists \boldsymbol{v}.\,(\gamma_1(\boldsymbol{v}) \wedge \gamma_2(\boldsymbol{v}))$, i.e., $\mathcal{B}_1^\infty$ satisfies the sentence that is obtained after Step 2 of the Combination Algorithm. Thus there exists an assignment $\nu : V \to B_1^\infty$ such that $\mathcal{B}_1^\infty \models \gamma_1(\nu(\boldsymbol{v})) \wedge \gamma_2(\nu(\boldsymbol{v}))$.

In Step 3 of the decomposition algorithm we identify two variables u and u' of $\boldsymbol{v}$ if, and only if, $\nu(u) = \nu(u')$. With this choice, the assignment ν satisfies the formula obtained after the identification step, i.e.,

$$\mathcal{B}_1^\infty \models \gamma_1'(\nu(\boldsymbol{w})) \wedge \gamma_2'(\nu(\boldsymbol{w})),$$

and all components of $\nu(\boldsymbol{w})$ are distinct.

In Step 4, a variable w in $\boldsymbol{w}$ is labeled with 2 if $\nu(w) \in Y_1$, and with 1 otherwise. In order to choose the linear ordering on the variables, we partition the range B_1^∞ of ν as follows:

$$B_1^{(0)},\ \ Y_1^{(1)},\ \ B_1^{(1)} \setminus (B_1^{(0)} \cup Y_1^{(1)}),\ \ Y_1^{(2)},\ \ B_1^{(2)} \setminus (B_1^{(0)} \cup Y_1^{(2)}),$$
$$Y_1^{(3)},\ \ B_1^{(3)} \setminus (B_1^{(2)} \cup Y_3),\ \ \ldots$$

In Fig. 3 these subsets correspond to the areas (0 and 1), 2, 3, 4, 5, 6, ... of the left-hand side. Now, let $\boldsymbol{u}_1, \boldsymbol{v}_1, \ldots, \boldsymbol{u}_k, \boldsymbol{v}_k$ be a re-ordering of the tuple $\boldsymbol{w}$ such that the following holds:

1. The tuple $\boldsymbol{u}_1$ contains exactly the variables whose ν-images are in $B_1^{(0)}$.
2. For all $i, 1 \leq i \leq k$, the tuple $\boldsymbol{v}_i$ contains exactly the variables whose ν-images are in $Y_1^{(i)}$.
3. For all $i, 1 < i \leq k$, the tuple $\boldsymbol{u}_i$ contains exactly the variables whose ν-images are in $B_1^{(i-1)} \setminus (B_1^{(i-2)} \cup Y_1^{(i-1)})$.

Obviously, this implies that the variables in the tuples $\boldsymbol{v}_i$ have label 2, whereas the variables in the tuples $\boldsymbol{u}_i$ have label 1. Note that some of these tuples may be of dimension 0. This re-ordering of $\boldsymbol{w}$ determines the linear ordering we choose in Step 4. Let

$$\alpha = \forall \boldsymbol{u}_1.\exists \boldsymbol{v}_1. \cdots \forall \boldsymbol{u}_k.\exists \boldsymbol{v}_k.\,\gamma_1' \quad \text{and} \quad \beta = \exists \boldsymbol{u}_1.\forall \boldsymbol{v}_1. \cdots \exists \boldsymbol{u}_k.\forall \boldsymbol{v}_k.\,\gamma_2'$$

be the output pair that is obtained by these choices. Let $\boldsymbol{x}_i := \nu(\boldsymbol{w}_i)$ and $\boldsymbol{e}_i := \nu(\boldsymbol{v}_i)$. For $i = 1, \ldots, k$, let Z_i denote a finite set of variables in $B_1^{(i-1)} \cap (V \cup Y_1)$ that generates all elements in $\boldsymbol{e}_i$. We claim that the sequence $\boldsymbol{x}_1, \boldsymbol{e}_1, \ldots, \boldsymbol{x}_k, \boldsymbol{e}_k$ and the sets $Z_1, \ldots, Z_k$ satisfy Condition 2 of Lemma 4 for $\varphi = \gamma_1'$ and the structure $\mathcal{B}_1^\infty = T(\Sigma_1, V \cup Y_1)/{=}E_1$.

[12] Here and in the sequel, $\mathcal{B}_1^\infty$ is sometimes treated as a $(\Sigma_1 \cup \Sigma_2)$-algebra, using the signature expansion described in the construction of $\mathcal{B}_1 \oplus \mathcal{B}_2$.

Part (a) of this condition is satisfied since $\mathcal{B}_1^\infty \models \gamma_1'(\nu(\boldsymbol{w}))$, and thus

$$\mathcal{B}_1^\infty \models \gamma_1'(\boldsymbol{x}_1, \boldsymbol{e}_1, \ldots, \boldsymbol{x}_k, \boldsymbol{e}_k).$$

Part (b) of the condition is satisfied since the ν-images of all variables in $\boldsymbol{w}$ are distinct according to our choice in the variable identification step. Part (c) is satisfied due to our choice of the sets Z_j. Part (d) is satisfied since the components of $\boldsymbol{x}_j$ belong to $\boldsymbol{Y}_1^{(j)}$ and $Y_1^{(j)} \cap (Z_1 \cup \ldots \cup Z_{j-1}) = \emptyset$, the last equality following from the fact that $Y_1^{(j)}$ and $\bigcup_{i=0}^{j-1} Y_1^{(i)}$ are disjoint by definition.

Thus, we can apply Lemma 4, which yields $\mathcal{B}_1^\infty \models \alpha$. Since $\mathcal{B}_1^\infty$ and $\mathcal{B}_1$ are Σ_1-isomorphic we have $\mathcal{B}_1 \models \alpha$.

Using the fact the $h_{1-2} : \mathcal{B}_1^\infty \to \mathcal{B}_2^\infty$ is a $(\Sigma_1 \cup \Sigma_2)$-isomorphism, $\mathcal{B}_2 \models \beta$. can be shown similarly. □

6 Generalizations

The combination method for equational unification algorithms that we have described in the previous two sections can be generalized along several orthogonal dimensions. Three such extensions will be described in this section. The first generalization concerns the syntactic form of input problems: we study the effect of adding negation to the mixed input sentences. Afterwards we introduce a class of structures that properly extends the class of free algebras, and show how to lift our combination results to this more general class of structures. In the third subsection we sketch a variant of the amalgamation construction introduced in Subsection 5.2, which leads to a different combined solution structure and a combination algorithm with less nondeterminism.

6.1 Adding negation to unification problems

Compared to the Nelson-Oppen approach, the major limitation of the combination results presented in the previous two sections is that they are restricted to *positive* sentences, i.e., negated equations, so-called *disequations*, are not allowed. We shall now consider the combination of unification constraints with negation. Since the constraint solvers of constraint programming languages often have to check entailment of constraints, and since entailment uses an implicit form of negation, this extension is of great practical relevance.

As before, let E_1 and E_2 denote equational theories over disjoint signatures Σ_1 and Σ_2. When treating sentences with negation, we must first decide which form of semantics we want to use: the equivalence between validity in all models of $E_1 \cup E_2$ on the one hand, and validity in the $(E_1 \cup E_2)$-free algebra over a countably infinite set of generators V on the other hand does no longer hold if we do not restrict ourselves to positive sentences. We shall look at two alternative semantics usually considered in the literature. First, we consider validity of existential $(\Sigma_1 \cup \Sigma_2)$-sentences in the free algebra $T(\Sigma_1 \cup \Sigma_2, V)/{=_{E_1 \cup E_2}}$. Later, we consider validity in the initial algebra $T(\Sigma_1 \cup \Sigma_2, \emptyset)/{=_{E_1 \cup E_2}}$. In the

first case, we talk about *solvability* and in the second about *ground solvability* of the constraints.

As long as we want to decide validity of positive existential sentences, both semantics lead to the same result as long as we assume that the joint signature contains at least one constant. This follows directly from the fact that validity of positive existential sentences is preserved under homomorphisms. For constraints with negation, the two semantics definitely lead to distinct notions of validity. The latter semantics is often preferred in the literature on constraints with negation (see, e.g., [23]), but the first semantics can also be found [17].

Since a sentence holds in a given algebra $\mathcal{A}$ if, and only if, its negation does not hold in $\mathcal{A}$, a decision procedure for validity of existential sentences in $\mathcal{A}$ immediately gives a decision procedure for validity of universal sentences in $\mathcal{A}$ and vice versa. Hence the results of this subsection concern the universal fragments of the given algebras as well.

Disunification over the free algebra In order to describe the following results, some terminology is needed. Given an equational theory E with signature Σ, an *elementary E-disunification problem* is a finite system Γ of equations $s =^? t$ and disequations $s \neq^? t$ between Σ-terms. A substitution σ solves Γ iff $\sigma(s) =_E \sigma(t)$ for each equation $s =^? t$ in Γ and $\sigma(s) \neq_E \sigma(t)$ for each disequations $s \neq^? t$ in Γ. E-disunification problems with linear constant restrictions, and solutions for E-disunification problems with linear constant restrictions are defined as in the case of E-unification problems.

Theorem 12. *Let $E_1, \ldots, E_n$ be equational theories over pairwise disjoint signatures, let $E := E_1 \cup \ldots \cup E_n$ denote their union. Then solvability of E-disunification problems is decidable provided that solvability of E_i-disunification problems with linear constant restrictions is decidable for $i = 1, \ldots n$.*

Since existential quantification distributes over disjunction, the theorem shows that validity of existential sentences in $T(\Sigma_1 \cup \ldots \cup \Sigma_n, V)/=_E$ is decidable if solvability of E_i-disunification problems with linear constant restrictions is decidable for $i = 1, \ldots n$. Unfortunately, we do not have a logical characterization of E_i-*dis*unification problems with linear constant restrictions.

A proof of this theorem can be found in [5]. It is based on a combination algorithm that is a variant of the Combination Algorithm for combined E-unification problems. In principle, the only difference is that, for each pair (x, y) of variables in the input problem that is not identified at the variable identification step, we add a disequation $x \neq y$ to both output systems. For details we refer to [5].

Disunification over the initial algebra A solution σ of the E-disunification problem Γ is a *ground solution* iff $\sigma(x)$ is a ground term (i.e., does not contain variables) for all variables x occurring in Γ.

In view of Theorem 12, an obvious conjecture could be that ground solvability of a disunification problem Γ in the combined theory E can be decided by

decomposing Γ into a finite set of pairs of E_i-disunification problems with linear constant restrictions, and then asking for ground solvability of the subproblems. However, in [5] an example is given that shows that this method is only sound, but not complete (see Example 4.2, p. 243). The proper adaption of Theorem 13 to the case of ground solvability needs another notation: a solution σ of an E-disunification problem with linear constant restrictions, $(\Gamma, X, C, <)$, is called *restrictive* if, under σ, all variables $x \in X$ are mapped to terms $\sigma(x)$ that are not E-equivalent to a variable.

Theorem 13. *Let $E_1, \ldots, E_n$ be equational theories over pairwise disjoint signatures $\Sigma_1, \ldots, \Sigma_n$, and let $E := E_1 \cup \ldots \cup E_n$ denote the combined theory. Assume that the initial algebras $T(\Sigma_i, \emptyset)/=_{E_i}$ are infinite for $i = 1, \ldots, n$. Then ground solvability of E-disunification problems is decidable provided that restrictive solvability of E_i-disunification problems with linear constant restrictions is decidable for $i = 1, \ldots n$.*

A proof of this theorem, as well as of some variants that relax the condition that all the initial algebras must be infinite, can be found in [5]. These techniques yield the following result.

Corollary 1. *Solvability of disunification problems is decidable for every equational theory that is a disjoint combination of finitely many theories E_f expressing associativity, associativity-commutativity, or associativity-commutativity-idempotence of some binary function symbol, together with a free theory F. If the free theory F contains at least one constants and one function symbol of arity $n \geq 1$, then ground solvability of disunification problems over the combined theory is decidable as well.*

6.2 More general solution structures

Except for the initial description of the Nelson-Oppen procedure, our discussion has been restricted to constraints that are composed of equations and disequations, and the only solution domains that we considered so far were free algebras. Obviously, a much broader variety of constraints and solution domains are relevant for the general field of constraint programming. In this subsection we first introduce a class of structures that properly extends the class of free algebras. The class contains many non-free algebras and relational structures that are of interest for constraint solving. Then we discuss the problem of combining solution domains within the given class. Finally, we show how the combination results that we obtained for free algebras can be lifted to this more general situation.

Quasi-free structures The motivation for introducing the class of quasi-free structures is the observation that most of the non-numerical and non-finite solution domains that are used in different areas of constraint programming can be

treated within a common algebraic background when we generalize the concept of a free algebra appropriately.

In a first step, one can go from free algebras to free structures where, in addition to function symbols, the signature may also contain predicate symbols. Free structures are defined Mal'cev [47] analogously to free algebras: a Σ-structure $\mathcal{A}$ is called free over X in the class of Σ-structures $\mathcal{K}$ if $\mathcal{A} \in \mathcal{K}$ is generated by X, and if every mapping from X into the domain of a structure $\mathcal{B} \in \mathcal{K}$ can be extended to a Σ-homomorphism of $\mathcal{A}$ into $\mathcal{B}$. Mal'cev shows that free structures have properties that are very similar to the properties of free algebras. This fact was used in [4] to extend the combination results for unification constraints to more general constraints over free solution structures.

The following lemma (see [7], Theorem 3.4) yields an internal characterization of structures that are free in some class of over a countably infinite set of generators. It will be the basis for our generalization from free structures to quasi-free structures.

Lemma 6. *A Σ-structure $\mathcal{A}$ is free (in some class of Σ-structures) over $X \subseteq A$ iff*

1. *$\mathcal{A}$ is generated by X,*
2. *for every finite subset X_0 of X, every mapping from X_0 to A can be extended to a surjective endomorphism of $\mathcal{A}$.*

We will now generalize the first condition in order to arrive at the concept of a quasi-free structure. Since some of the following notions are quite abstract, the algebra of rational trees will be used to exemplify definitions. In the sequel, we consider a fixed Σ-structure $\mathcal{A}$ with domain A. With $End_{\mathcal{A}}^{\Sigma}$ we denote the monoid of Σ-endomorphisms of $\mathcal{A}$. It should be stressed that in the sequel the signature Σ is arbitrary in the sense that it may contain predicate symbols as well as function symbols.

Definition 3. *Let A_0, A_1 be subsets of the Σ-structure $\mathcal{A}$. Then* A_0 stabilizes A_1 *iff all elements m_1 and m_2 of $End_{\mathcal{A}}^{\Sigma}$ that coincide on A_0 also coincide on A_1. For $A_0 \subseteq A$ the* stable hull *of A_0 is the set*

$$SH^{\mathcal{A}}(A_0) := \{a \in A \mid A_0 \text{ stabilizes } \{a\}\}.$$

The stable hull of a set A_0 has properties that are similar to those of the subalgebra generated by A_0: $SH^{\mathcal{A}}(A_0)$ is always a Σ-substructure of $\mathcal{A}$, and $A_0 \subseteq SH^{\mathcal{A}}(A_0)$. In general, however, the stable hull can be larger than the generated substructure. For example, if $\mathcal{A} := \mathcal{R}(\Sigma, X)$ denotes the algebra of rational trees over signature Σ and with variables in X, and if $Y \subseteq X$ is a subset of the set of variables X, then $SH^{\mathcal{A}}(Y)$ consists of all *rational* trees with variables in Y, while Y generates all *finite* trees with variables in Y only.

Definition 4. *The set $X \subseteq A$ is an atom set for $\mathcal{A}$ if every mapping $X \to A$ can be extended to an endomorphism of $\mathcal{A}$.*

For example, if $\mathcal{A} := \mathcal{R}(\Sigma, X)$ is the algebra of rational trees with variables in X, then X is an atom set for $\mathcal{A}$.

Definition 5. *A countably infinite Σ-structure $\mathcal{A}$ is* quasi-free *iff $\mathcal{A}$ has an infinite atom set X where every $a \in A$ is stabilized by a finite subset of X.*

The definition generalizes the characterization of free algebras given in Lemma 6. The countably infinite set of generators is replaced by the atom set, but we retain some properties of generators. In the free case, every element of the algebra is generated by a finite set of generators, whereas in the quasi-free case it is stabilized by a finite set of atoms. It can be shown easily that the second condition of Lemma 6 holds in the quasi-free case as well.

Examples 14 Each free algebra and each free structure is quasi-free. Examples of non-free quasi-free structures are rational tree algebras; nested, hereditarily finite non-wellfounded sets, multisets, and lists; as well as various types of feature structures. In each case we have to assume the presence of a countably infinite set of atoms (variables, urelements, etc.). For the exact definitions of these example structures we refer to [7].

Free amalgamation of quasi-free structures When combining constraint systems for quasi-free structures, the question arises how to define the combined solution structure. It turns out that the amalgamation construction that we have described in Subsection 5.2 can be generalized from free algebras to quasi-free structures. The result of this construction is a quasi-free structure over the combined signature. In the modified construction, the atom sets of the two quasi-free component structures play the rôle of the variable sets. The intermediate substructures that occur during the fibering process are now defined as the stable hulls of the atom sets considered at the steps of the construction.

In the case of free algebras, the use of the amalgamation construction was justified by the fact that it yielded exactly the combined algebra we were looking for, i.e., the free algebra for the combined theory. In the case of quasi-free structures, we do not have an equational theory defining the component structures. Thus, the question arises whether the amalgamation construction really yields a "sensible" combined solution structure. This question has been answered affirmatively in [7].

In fact, the resulting combined structure has a unique and privileged status. In [7] we have introduced the notion of an admissible combination of two structures. The *free amalgamated product* $\mathcal{A}_1 \oplus \mathcal{A}_2$ of two structures is the most general admissible combination of $\mathcal{A}_1$ and $\mathcal{A}_2$ in the sense that every other admissible combination $\mathcal{C}$ is a homomorphic image of $\mathcal{A}_1 \oplus \mathcal{A}_2$ (see [7] for an exact definition). It can be shown that the free amalgamated product of two quasi-free structures over disjoint signatures always exists since it coincides with the structure produced by our amalgamation construction (i.e., the extension to quasi-free structures of the construction presented above for the case of free algebras).

Solving mixed constraints in the free amalgamated product When using the free amalgamated product of two given quasi-free structures $\mathcal{A}_1$ and $\mathcal{A}_2$ as the solution domain for mixed constraints, a simple adaption of the logical reformulation of the Combination Algorithm can be used to reduce solvability of positive existential formulae in $\mathcal{A}_1 \oplus \mathcal{A}_2$ to solvability of positive sentences in the component structures $\mathcal{A}_1$ and $\mathcal{A}_2$. The only difference comes from the fact that we now have a new type of atomic formulae in the input problems, namely, atomic formulae that are built with predicate symbols in the signature. It is, however, straightforward to show that, given an existential sentence $\exists \boldsymbol{u}.\, \gamma$ over the mixed signature $\Sigma_1 \cup \Sigma_2$, it is possible to compute an equivalent existential sentence of the form $\exists \boldsymbol{v}.\, (\gamma_1 \wedge \gamma_2)$ where the conjunction of atomic formulae γ_i is built using symbols from Σ_i only; in fact, the variable abstraction step introduced in Section 3 also treats non-equational atoms. The remaining steps of the logical version of the Combination Algorithm can be used without any changes.

The correctness of the modified Combination Algorithm, which is proved in [7], yields the following result.

Theorem 15. *Let $\mathcal{A}_1, \ldots, \mathcal{A}_n$ be quasi-free structures over disjoint signatures $\Sigma_1, \ldots, \Sigma_n$, and let Σ denote the union of these signatures. Then validity of positive existential Σ-sentences in the free amalgamated product $\mathcal{A}_1 \oplus \cdots \oplus \mathcal{A}_n$ is decidable provided that validity of positive Σ_i-sentences in the component structures is decidable.*

As in the case of free algebras, it is possible to lift this result to general positive input sentences.

Theorem 16. *Let $\mathcal{A}_1, \ldots, \mathcal{A}_n$ be quasi-free structures over disjoint signatures $\Sigma_1, \ldots, \Sigma_n$, and let Σ denote the union of these signatures. Then validity of positive Σ-sentences in the free amalgamated product $\mathcal{A}_1 \oplus \cdots \oplus \mathcal{A}_n$ is decidable provided that validity of positive Σ_i-sentences in the component structures is decidable.*

For the following quasi-free structures, the positive theories turn out to be decidable (cf. Ex. 14): non-ground rational feature structures with arity; finite or rational tree algebras; nested, hereditarily finite wellfounded or non-wellfounded sets; and nested, hereditarily finite wellfounded or non-wellfounded lists. Hence, provided that the signatures are disjoint, the free amalgamated product of any finite number of these structures has a decidable positive theory.

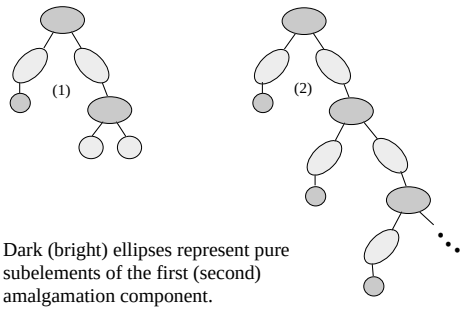
It is also possible to extend the results combination results for disunification to the case of quasi-free structures. This yields decidability results for the existential (or universal) theory of the free amalgamated product of a great variety of structures, such as feature structures, nested lists, sets and multisets, rational tree algebras and others. We refer to Kepser [36, 35] for details.

6.3 Other amalgamation techniques

In Schulz and Kepser [39] a second systematic way of combining constraint systems over quasi-free structures, called *rational amalgamation*, has been intro-

duced. Like the free amalgamated product, rational amalgamation yields a combined structure with "mixed" elements that inter-weave a finite number of "pure" elements of the two components in a particular way. The difference between both constructions becomes transparent when we ignore the interior structure of these pure subelements and consider them as construction units with a fixed arity, similar to "complex function symbols." Under this perspective, and ignoring details that concern the ordering of the children of a node, mixed elements of the free amalgamated product can be considered as finite trees, whereas mixed elements of the rational amalgam are like rational trees.[13]

Mixed element of free amalgam (1) and of rational amalgam (2).



Dark (bright) ellipses represent pure subelements of the first (second) amalgamation component.

With this background, it should not be surprising that in praxis rational amalgamation appears to be the preferred combination principle in the literature in situations where the two solution structures to be combined are themselves "rational" or "cyclic" domains: for example, it represents the way how rational trees and rational lists are interwoven in the solution domain of Prolog III [22], and a variant of rational amalgamation has been used to combine feature structures with non-wellfounded sets in a system introduced by Rounds [59].

Rational amalgamation can be used to combine so-called non-collapsing quasi-free structures over disjoint signatures.

Definition 6. *An quasi-free structure $\mathcal{A}$ with atom set X is* non-collapsing *if every endomorphism of $\mathcal{A}$ maps non-atoms to non-atoms (i.e., $m(a) \in A \setminus X$ for all $a \in A \setminus X$ and all endomorphisms m of $\mathcal{A}$).*

For example, quotient term algebras for collapse-free equational theories, rational tree algebras, feature structures, feature structures with arity, the domains with nested, finite or rational lists, and the domains with nested, finite or rational multi-sets are always non-collapsing.

The amalgamation construction for rational amalgamation is rather technical and thus beyond the scope of this paper; we refer to [39] for details. Just as in the case of free amalgamation, constraint solvers for two component structures can be combined to a constraint solver for their rational amalgam. To be

[13] A (possibly infinite) tree is *rational* if it is finitely branching and has only a finite number of distinct subtrees; see [21, 45, 24].

more precise, validity of positive existential sentences in the rational amalgam can be reduced to solvability of conjunctions of atomic constraints with so-called atom/non-atom declarations in the component structures (see [39] for a formal definition of this notion). From the algorithmic point of view, rational amalgamation appears to be interesting since the combination technique for rational amalgamation avoids one source of nondeterminism that is needed in the corresponding scheme for free amalgamation: the choice of a linear ordering, which is indispensable for free amalgamation, must be omitted in the case of rational amalgamation.

One interesting connection between free and rational amalgamation is the observation that the free amalgamated product is always a substructure of the rational amalgamated product (see [35]).

7 Optimization and complexity issues

Until now, we have described the combination method for unification algorithms from a theoretical point of view, that is, our main emphasis was on clearness of presentation and on ease of proving correctness. It should be clear, however, that a naïve implementation of the highly nondeterministic Combination Algorithm cannot be used in practice. It is easy to see that the nondeterminism of the procedure indeed represents a serious problem: the number of possible partitions of a set of n variables is known as the n-th Bell number, which grows faster than 2^n. The choice of a labeling function and a linear ordering leads to another exponential number of subcases that must be investigated. Hence, significant optimizations are necessary before one can hope for a combined unification algorithm that can be used in a realistic application.

In the following, we show how the algorithm that combines decision procedures can be optimized. (An optimized version of the combination method for algorithms that compute complete sets of unifiers can be found in [15].) In general, however, there is an inherent nondeterminism in the problem of combining unification algorithms, which cannot be avoided. We will come back to this point at the end of this section.

Some simple optimizations of the Combination Algorithm are quite straightforward. It is possible to restrict all nondeterministic choices to "shared" variables, that is, variables that occur in at least two subproblems of the decomposed problem. Another simple optimization relies on the observation that different linear orders need not lead to different constant restrictions. For example, assume that x, y are variables and c, d are (variables treated as) constants. Then the ordering $x < c < d < y$ leads to the same restrictions on solutions of a unification problem as the ordering $x < d < c < y$ (both just say that x must not be replaced by a term containing c or d). This observation can easily be used to prune the number of different linear orderings that must be considered.

On a more sophisticated level, Kepser and Richts [37] have described two powerful orthogonal optimization methods. We describe the first method, called

"deductive method," in more detail, and then briefly sketch the second one, called "iterative method," and the integration of both approaches.

The deductive method tries to reduce the amount of nondeterminism by avoiding certain branches in the search tree for which one can "easily detect" that they cannot lead to solutions. Before going into more detail, we consider an example that illustrates the basic idea.

Example 7. Assume that the component theory E_i is collapse-free and the decomposed input problem contains an equation $x =^? f(\ldots)$ where $f \in \Sigma_i$. Then x must receive label i since $x \neq_{E_i} \sigma(f(\ldots))$ for all substitutions σ, i.e., if x is treated as a constant in the ith subproblem, then this problem does not have a solution. Consequently, labeling functions Lab with $Lab(x) \neq i$ need not be considered.

If E_i is regular, the decomposed input problem contains an equation $x =^? t$, and $y \in Var(t)$ for a variable y with $Lab(x) \neq Lab(y)$, then there cannot be a solution σ (of the subproblem in which x is instantiated) in which y does not occur in $\sigma(x)$. Hence, we can deterministically choose the order $y < x$ between x and y, i.e., the other alternative need not be considered.

In order to formalize this idea, we introduce a constraint language that allows us to represent such mandatory choices on the way to a fully specified output pair of the Combination Algorithm. A complete set of guesses of the algorithm—with the trivial optimizations mentioned above included now—can be described in the form $(\Pi, Lab, <)$, where

- Π is a partition of the set X of shared variables of the decomposed problem $\Gamma_1 \uplus \ldots \uplus \Gamma_n$ reached after the first step,
- $Lab : X \to \{1, \ldots, n\}$ is a labeling function that respects equivalence classes of Π, i.e., if x and y belong to the same class, then $Lab(x) = Lab(y)$, and
- $<$ is a strict linear ordering on the equivalence classes. We write $x < y$ if the equivalence classes $[x]$ and $[y]$ of x and y are in the relation $[x] < [y]$.

In the sequel, output problems will be described as quadruples of the form $(\Gamma_i, \Pi, Lab, <)$. The corresponding E_i-unification with linear constant restrictions, $(\Gamma_i', X_i, C_i, <)$, can be obtained from this quadruple as described in the Combination Algorithm, i.e., Γ_i' is obtained from Γ_i by replacing all shared variables by the representatives of their equivalence classes w.r.t. Π, X_i is the union of the set of shared variables with label i and the set of non-shared variables in Γ_i, and C_i is the set of shared variables with a label different from i. The quadruple $(\Gamma_i, \Pi, Lab, <)$ is said to be *solvable* iff the corresponding E_i-unification with linear constant restrictions is solvable.

Constraints are of the form $x = y$, $\neg(x = y)$, $x < y$, $\neg(x < y)$, $x : i$, or $\neg(x : i)$, with the obvious meaning that $x = y$ ($\neg(x = y)$) excludes partitions in which x and y belong to different classes (the same class), $x < y$ ($\neg(x < y)$) excludes orderings and partitions in which $y \leq x$ ($x < y$), and $x : i$ ($\neg(x : i)$) excludes labelling functions Lab such that $Lab(x) \neq i$ ($Lab(x) = i$). On the one hand, a set of constraints excludes certain triples $(\Pi, Lab, <)$. On the other hand, it can

also be seen as a partial description of a triple that satisfies these constraints (i.e., is not excluded by them). A set of constraints $\mathcal{C}$ is called *complete* iff there is exactly one triple $(\Pi, Lab, <)$ that satisfies $\mathcal{C}$, and it is called *inconsistent* iff no triple satisfies $\mathcal{C}$ (i.e., it contains two contradictory constraints).

The deductive method assumes that each theory E_i is equipped with a *component algorithm* that, given a pure E_i-unification problem Γ_i together with a set of constraints $\mathcal{C}$, deduces a (possibly empty) set of additional constraints $\mathcal{D}$. This algorithm is required to be correct in the following sense: if $(\Pi, Lab, <)$ is a triple that satisfies $\mathcal{C}$ and for which $(\Gamma_i, \Pi, Lab, <)$ is solvable, then $(\Pi, Lab, <)$ also satisfies $\mathcal{D}$.

Given a system $\Gamma_1 \uplus \cdots \uplus \Gamma_n$ in decomposed form, the search for an appropriate triple $(\Pi, Lab, <)$ is now performed by the nondeterministic algorithm of Fig. 4.

```
Initialize C := ∅;
Repeat
   Repeat
      For each system i
         (* Deduce new constraints *)
         call the component algorithm of theory E_i to calculate
         new consequences D of Γ_i and C;
         set the current set of constraints to C := C ∪ D
   Until C is inconsistent or no more new constraints are computed;
   If C is consistent and not complete
      (* Select next choice *)
      Select a constraint c such that {c, ¬c} ∩ C = ∅;
      Non-deterministically choose either
      C := C ∪ {c} or
      C := C ∪ {¬c}
Until C is inconsistent or complete;
Return C
```

Fig. 4. The deductive method.

Proposition 5. *Let $\Gamma := \Gamma_1 \uplus \cdots \uplus \Gamma_n$ be an (elementary) $(E_1 \cup \cdots \cup E_n)$-unification problem in decomposed form where the equational theories E_i have pairwise disjoint signatures. Then the following statements are equivalent:*

1. *Γ is solvable, i.e., there exists an $(E_1 \cup \cdots \cup E_n)$-unifier of Γ.*
2. *One of the complete constraint sets generated by the nondeterministic algorithm of Fig. 4 describes a triple $(\Pi, \mathrm{Lab}, <)$ such that, for all $i = 1, \ldots, n$, $(\Gamma_i, \Pi, Lab, <)$ is solvable.*

One should note that the trivial component algorithm that always returns the empty set of constraints is correct. If all component algorithms are trivial, then the algorithm of Fig. 4 simply generates all possible triples $(\Pi, Lab, <)$.

We have already illustrated by an example that the fact that a theory is regular and/or collapse-free can be used to derive new constraints. For a free theory E_i, the most general unifier of Γ_i (which can be computed in linear time) can be used to read off new constraints. The following example shows how information provided by one component algorithm can help another component algorithm in deriving additional constraints. This explains why the step of deducing new constraints must be iterated.

Example 8. Assume that we are given the mixed input problem $\{f(g(x_4), x_2) \stackrel{?}{=} f(g(y), x_4),\ x_4 \stackrel{?}{=} f(a, a)\}$, where f, a belong to the regular, collapse-free theory E_1 (e.g., AC_f) and g belongs to the free theory E_2. By decomposition, the E_1-subsystem $\{f(x_1, x_2) \stackrel{?}{=} f(x_3, x_4),\ x_4 \stackrel{?}{=} f(a, a)\}$ and the E_2-subsystem $\{x_1 \stackrel{?}{=} g(x_4),\ x_3 \stackrel{?}{=} g(y)\}$ are created. Since E_1 is collapse-free, the equation $x_4 \stackrel{?}{=} f(a, a)$ can be used by the first component algorithm to deduce the constraint $x_4 : 1$. From the most general unifier $\{x_1 \mapsto g(x_4),\ x_3 \mapsto g(y)\}$ of the E_2-subsystem, the second component algorithm can derive the constraints $x_1 : 2, x_3 : 2$ and $x_4 < x_1$. Given the regularity of E_1, the first component algorithm can now derive $x_1 = x_3$. In fact, x_1 (which must be treated as a constant in the E_1-subsystem) occurs on the left-hand side of $f(x_1, x_2) \stackrel{?}{=} f(x_3, x_4)$, and thus must occur on the (instantiated) right-hand side $f(\sigma(x_3), \sigma(x_4))$ for any solution σ of the E_1-subsystem. Since we already have the constraints $x_4 : 1$, $x_4 < x_1$, and $x_3 : 2$, we know that $\sigma(x_3) = x_3$ and x_1 cannot occur in $\sigma(x_4)$. Consequently, x_1 can only occur in $f(\sigma(x_3), \sigma(x_4)) = f(x_3, \sigma(x_4))$ if x_1 and x_3 are identified.

Obviously, the quality of the component algorithms used in the deductive method decides the amount of optimization achieved. The goal is to deduce as much information as is possible with a reasonable effort. Detailed descriptions of component algorithms for free theories, the theory AC of an associative-commutative function symbol, and the theory ACI of an associative-commutative-idempotent function symbol can be found in [56].

While the deductive method helps to reach certain decisions deterministically, the "iterative method" introduced in [37, 35]—which is relevant if $n \geq 3$ theories are combined—determines in which order the nondeterministic decisions should best be made. Basically, the output systems are solved iteratively, one system at a time. All decisions in nondeterministic steps are made locally, for the current system i only. This means, for example, that we only consider variables occurring in the system Γ_i, and for such a variable we just decide if it receives label i or not. In the latter case, the exact label $j \neq i$ is not specified. Once all decisions relevant to system Γ_i have been made, it is immediately tested for solvability. If Γ_i turns out to be unsolvable, we thus have avoided finding this out as many times as there are possible choices for the constraints not relevant to the subsystem Γ_i.

An integration of the deductive and the iterative method is achieved by plugging the iterative selection strategy into the deductive algorithm. To be more precise, whenever the deductive algorithm (see Fig. 4) needs to make a nondeterministic choice (since no more constraints can be deduced), the selection strategy of the iterative method decides for which constraint this choice is made. This synthesis of both optimization techniques has been implemented, and run

time tests show that the optimized combination method obtained this way leads to combined decision procedures that have a quite reasonable practical time complexity [37, 38].

Fundamental limitations for optimization Complexity theoretical considerations in [62] show that, in many cases, there are clear limitations for optimizing the Combination Algorithm. We close this section with some results concerning the situation where an equational theory E is combined with a free theory in order to obtain an algorithm for general E-unification.

Definition 7. *A* polynomial-time optimization of the Combination Algorithm for general E-unification *is an algorithm that accepts as input an arbitrary general E-unification problem Γ and computes in polynomial time a finite set M of output pairs $((\Gamma_1, \Pi, X_1, X_2, <), (\Gamma_2, \Pi, X_2, X_1, <))$ of an E-unification problems with linear constant restrictions and a free unification problems with linear constant restrictions such that*

- *each output pair in M is also a possible output pair of the original Combination Algorithm, and*
- *Γ is solvable iff, for some output pair in M, both components are solvable.*

On the one hand, Schulz [62] characterizes a large class of equational theories E where a polynomial optimization of the Combination Algorithm for general E-unification is impossible unless $P = NP$. In order to formulate one result that follows from this characterization, we need the following notation: a binary function symbol "f" is called a commutative (resp. associative) function symbol of the equational theory E if f belongs to the signature of E and $f(x, y) =_E f(y, x)$ (resp. $f(x, f(y, z)) =_E f(f(x, y), z)$).

Theorem 17. *Let E be an equational theory that contains an associative or commutative function symbol. If E is regular, then there exists no polynomial-time optimization of the Combination Algorithm for general E-unification, unless $P = NP$.*

In [63] it is shown that such impossibility results for polynomial optimization of combination algorithms are by no means specific to the problem of combining E-unification algorithms. The paper presents a general framework that characterizes situations in which combination algorithms for decision procedures cannot be polynomially optimized. In particular, various combinations of first-order theories are characterized where the non-deterministic variant of the Nelson-Oppen procedure does not have a polynomial optimization.

On the other hand, Schulz [62] also introduces a class of equational theories for which a polynomial-time optimization of the Combination Algorithm is always possible. Basically, these are regular and collapse-free theories of unification type unitary (i.e., all solvable unification problems have a most general unifier) such that "enough" information about the most general unifier can be computed in polynomial time.

8 Open problems

The results described in this paper show that the problem of combining constraint solvers over disjoint signatures is well-investigated, at least if one considers as constraint solvers procedures that decide satisfiability of constraints.

As mentioned in Section 2.1, it is often desirable to have constraint solvers that are able to compute solved forms in an incremental way. To the best of our knowledge, there are no *general* results on how to combine such incremental constraint solvers. A general solution to this problem depends on a general and abstract definition of the concept of a solved form that covers most of the relevant instances.

Another challenging field for future research is the problem of combining constraint solvers over non-disjoint signatures. Since non-disjoint combinations may lead to undecidability, the main point is to find appropriate restrictions on the constraint languages to be combined. For the kind of combination problems considered by Nelson-Oppen, first combination results for the non-disjoint case have been obtained by Ch. Ringeissen and C. Tinelli [58, 71, 73]. Similarly, the known combination methods for solving the word problem in the union of equational theories have been lifted to the case of non-disjoint signatures in [26, 10–12]. Concerning the combination of unification algorithms for equational theories over non-disjoint signatures, first results have been presented in [26]. Using the more abstract algebraic concepts that have been developed during the last years it should be possible to simplify and then generalize this work, which only addresses the combination of algorithms for computing complete sets of unifiers.

Acknowledgements The authors should like to thank the anonymous referee for his comments, which helped to improve the presentation of this paper. This work was partially supported by the ESPRIT Working Group CCL and by the DFG SPP "Deduktion".

References

1. F. Baader. On the complexity of Boolean unification. *Information Processing Letters*, 67(4):215–220, 1998.
2. F. Baader and K.U. Schulz. Unification in the union of disjoint equational theories: Combining decision procedures. In D. Kapur, editor, *Proceedings of the 11th International Conference on Automated Deduction*, volume 607 of *Lecture Notes in Artificial Intelligence*, pages 50–65, Saratoga Springs, NY, USA, 1992. Springer-Verlag.
3. F. Baader and K.U. Schulz. General A- and AX-unification via optimized combination procedures. In *Proceedings of the Second International Workshop on Word Equations and Related Topics*, volume 677 of *Lecture Notes in Computer Science*, pages 23–42, Rouen, France, 1993. Springer-Verlag.
4. F. Baader and K.U. Schulz. Combination of constraint solving techniques: An algebraic point of view. In *Proceedings of the 6th International Conference on Rewriting Techniques and Applications*, volume 914 of *Lecture Notes in Artificial Intelligence*, pages 352–366, Kaiserslautern, Germany, 1995. Springer-Verlag.

5. F. Baader and K.U. Schulz. Combination techniques and decision problems for disunification. *Theoretical Computer Science B*, 142:229–255, 1995.
6. F. Baader and K.U. Schulz. Unification in the union of disjoint equational theories: Combining decision procedures. *J. Symbolic Computation*, 21:211–243, 1996.
7. F. Baader and K.U. Schulz. Combination of constraint solvers for free and quasi-free structures. *Theoretical Computer Science*, 192:107–161, 1998.
8. 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*, pages 41–125. Oxford University Press, Oxford, UK, 1994.
9. F. Baader and W. Snyder. Unification theory. In J.A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*. Elsevier Science Publishers, 2000. To appear.
10. F. Baader and C. Tinelli. A new approach for combining decision procedures for the word problem, and its connection to the Nelson-Oppen combination method. In W. McCune, editor, *Proceedings of the 14th International Conference on Automated Deduction (Townsville, Australia)*, volume 1249 of *Lecture Notes in Artificial Intelligence*, pages 19–33. Springer-Verlag, 1997.
11. F. Baader and C. Tinelli. Deciding the word problem in the union of equational theories sharing constructors. In P. Narendran and M. Rusinowitch, editors, *Proceedings of the 10th International Conference on Rewriting Techniques and Applications (RTA-99)*, volume 1631 of *Lecture Notes in Computer Science*, pages 175–189. Springer-Verlag, 1999.
12. F. Baader and C. Tinelli. Combining equational theories sharing non-collapse-free constructors. In H. Kirchner and Ch. Ringeissen, editors, *Proceedings of the 3rd International Workshop on Frontiers of Combining Systems (FroCoS 2000)*, volume 1794 of *Lecture Notes in Artificial Intelligence*, pages 257–271, Nancy, France, 2000. Springer-Verlag.
13. L. Bachmair. *Canonical Equational Proofs*. Birkhäuser, Boston, Basel, Berlin, 1991.
14. A. Bockmayr. Algebraic and logical aspects of unification. In K.U. Schulz, editor, *Proceedings of the 1st International Workshop on Word Equations and Related Topics (IWWERT '90)*, volume 572 of *Lecture Notes in Computer Science*, pages 171–180, Tübingen, Germany, October 1992. Springer-Verlag.
15. A. Boudet. Combining unification algorithms. *Journal of Symbolic Computation*, 8:449–477, 1993.
16. A. Boudet, J.-P. Jouannaud, and M. Schmidt-Schauß. Unification in Boolean rings and Abelian groups. *J. Symbolic Computation*, 8:449–477, 1989.
17. W.L. Buntine and H.-J. Bürckert. On solving equations and disequations. *Journal of the ACM*, 41(4):591–629, 1994.
18. H.-J. Bürckert. *A Resolution Principle for a Logic with Restricted Quantifiers*, volume 568 of *Lecture Notes in Artificial Intelligence*. Springer-Verlag, 1991.
19. J. Calmet and K. Homann. Classification of communication and cooperation mechanisms for logical and symbolic computation systems. In F. Baader and K.U. Schulz, editors, *Frontiers of Combining Systems: Proceedings of the 1st International Workshop, Munich (Germany)*, Applied Logic, pages 221–234. Kluwer Academic Publishers, March 1996.
20. C.C. Chang and H.J. Keisler. *Model Theory*, volume 73 of *Studies in Logic and the Foundations of Mathematics*. Elsevier, Amsterdam, 3rd edition, 1990. (1st ed., 1973; 2nd ed., 1977).

21. A. Colmerauer. Equations and inequations on finite and infinite trees. In *Proceedings of the International Conference on Fifth Generation Computer Systems*, pages 85–99, Tokyo, Japan, 1984. North Holland.
22. A. Colmerauer. An introduction to Prolog III. *Communications of the ACM*, 33(7):69–90, 1990.
23. H. Comon. Disunification: A survey. In J.-L. Lassez and G. Plotkin, editors, *Computational Logic: Essays in Honor of Alan Robinson*, pages 322–359. MIT Press, Cambridge, MA, 1991.
24. B. Courcelle. Fundamental properties of infinite trees. *Theoretical Computer Science*, 25(2):95–169, 1983.
25. D. Cyrluk, P. Lincoln, and N. Shankar. On Shostak's decision procedure for combinations of theories. In M.A. McRobbie and J.K. Slaney, editors, *Proceedings of the 13th International Conference on Automated Deduction, (New Brunswick, NJ)*, volume 1104 of *Lecture Notes in Artificial Intelligence*, pages 463–477. Springer-Verlag, 1996.
26. E. Domenjoud, F. Klay, and C. Ringeissen. Combination techniques for non-disjoint equational theories. In A. Bundy, editor, *Proceedings of the 12th International Conference on Automated Deduction*, volume 814 of *Lecture Notes in Artificial Intelligence*, pages 267–281, Nancy, France, 1994. Springer-Verlag.
27. F. Fages. Associative-commutative unification. In R. E. Shostak, editor, *Proceedings of the 7th International Conference on Automated Deduction*, volume 170 of *Lecture Notes in Computer Science*, pages 194–208, Napa, USA, 1984. Springer-Verlag.
28. E. Hemaspaandra. Complexity transfer for modal logic. In *Proceedings of the Ninth Annual IEEE Symposium on Logic in Computer Science (LICS '94)*, pages 164–175, Paris, France, 1994. IEEE Computer Society Press.
29. A. Herold. Combination of unification algorithms. In J.H. Siekmann, editor, *Proceedings of the 8th International Conference on Automated Deduction*, volume 230 of *Lecture Notes in Computer Science*, pages 450–469, Oxford, UK, 1986. Springer-Verlag.
30. A. Herold and J.H. Siekmann. Unification in Abelian semigroups. *J. Automated Reasoning*, 3:247–283, 1987.
31. J. Jaffar and J.-L. Lassez. Constraint logic programming. In *Proceedings of the 14th ACM Symposium on Principles of Programming Languages*, pages 111–119, Munich, Germany, 1987.
32. J. Jaffar, J.-L. Lassez, and M. Maher. A theory of complete logic programs with equality. *J. Logic Programming*, 1, 1984.
33. 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 A. Robinson*. MIT Press, Cambridge, MA, 1991.
34. J.-P. Jouannaud and H. Kirchner. Completion of a set of rules modulo a set of equations. *SIAM J. Computing*, 15(4):1155–1194, 1986.
35. S. Kepser. *Combination of Constraint Systems*. Phd thesis, CIS–Universität München, München, Germany, 1998. Also available as CIS-Report 98-118.
36. S. Kepser. Negation in combining constraint systems. In Dov Gabbay and Maarten de Rijke, editors, *Frontiers of Combining Systems 2, Papers presented at FroCoS'98*, pages 117–192, Amsterdam, 1999. Research Studies Press/Wiley.
37. S. Kepser and J. Richts. Optimisation techniques for combining constraint solvers. In Dov Gabbay and Maarten de Rijke, editors, *Frontiers of Combining Systems 2, Papers presented at FroCoS'98*, pages 193–210, Amsterdam, 1999. Research Studies Press/Wiley.

38. S. Kepser and J. Richts. Unimok – a system for combining equational unification algorithms. In P. Narendran and M. Rusinowitch, editors, *Proceedings of the 10th International Conference on Rewriting Techniques and Applications*, volume 1631 of *Lecture Notes in Computer Science*, pages 248–251, Trento, Italy, 1999. Springer-Verlag.
39. S. Kepser and K. U. Schulz. Combination of constraint systems II: Rational amalgamation. In E.C. Freuder, editor, *Principles and Practice of Constraint Programming - CP96*, volume 1118 of *LNCS*, pages 282–296. Springer, 1996. Long version to appear in Theoretical Computer Science.
40. C. Kirchner. *Méthodes et Outils de Conception Systématique d'Algorithmes d'Unification dans les Théories Equationelles.* Thèse d'État, Université de Nancy I, France, 1985.
41. C. Kirchner and H. Kirchner. Constrained equational reasoning. In *Proceedings of the ACM-SIGSAM 1989 International Symposium on Symbolic and Algebraic Computation*, Portland, Oregon, 1989. ACM Press.
42. H. Kirchner and Ch. Ringeissen. Combining symbolic constraint solvers on algebraic domains. *Journal of Symbolic Computation*, 18(2):113–155, 1994.
43. M. Kracht and F. Wolter. Properties of independently axiomatizable bimodal logics. *The Journal of Symbolic Logic*, 56(4):1469–1485, December 1991.
44. C. Landauer and K.L. Bellman. Integration systems and interaction spaces. In F. Baader and K.U. Schulz, editors, *Frontiers of Combining Systems: Proceedings of the 1st International Workshop, Munich (Germany)*, Applied Logic, pages 249–266. Kluwer Academic Publishers, March 1996.
45. M.J. Maher. Complete axiomatizations of the algebras of finite, rational and infinite trees. In *Proceedings of the Third Annual IEEE Symposium on Logic in Computer Science*, pages 348–357, Edinburgh, Scotland, 1988. IEEE Computer Society.
46. G.S. Makanin. The problem of solvability of equations in a free semigroup. *Math. Sbornik*, 103:147–236, 1977. English translation in Math. USSR Sbornik 32, 1977.
47. A.I. Mal'cev. *The Metamathematics of Algebraic Systems*, volume 66 of *Studies in Logic and the Foundation of Mathematics.* North Holland, Amsterdam, 1971.
48. A.I. Mal'cev. *Algebraic Systems*, volume 192 of *Die Grundlehren der mathematischen Wissenschaften in Einzeldarstellungen.* Springer-Verlag, Berlin, 1973.
49. G. Nelson and D.C. Oppen. Simplification by cooperating decision procedures. *ACM Trans. on Programming Languages and Systems*, 1(2):245–257, October 1979.
50. G. Nelson and D.C. Oppen. Fast decision procedures based on congruence closure. *J. ACM*, 27(2):356–364, 1980.
51. R. Nieuwenhuis and A. Rubio. AC-superposition with constraints: No AC-unifiers needed. In A. Bundy, editor, *Proceedings of the 12th International Conference on Automated Deduction*, volume 814 of *Lecture Notes in Artificial Intelligence*, pages 545–559, Nancy, France, 1994. Springer-Verlag.
52. E. Ohlebusch. On the modularity of termination of term rewriting systems. *Theoretical Computer Science*, 136:333–360, 1994.
53. E. Ohlebusch. Modular properties of composable term rewriting systems. *Journal of Symbolic Computation*, 20(1):1–41, 1995.
54. D.C. Oppen. Complexity, convexity and combinations of theories. *Theoretical Computer Science*, 12:291–302, 1980.
55. G. Plotkin. Building in equational theories. *Machine Intelligence*, 7:73–90, 1972.
56. J. Richts. *Effiziente Entscheidungsverfahren zur E-Unifikation.* Dissertation, RWTH Aachen, Germany, 1999. Published by Shaker Verlag Aachen, *Berichte aus der Informatik*, 2000.

57. C. Ringeissen. Unification in a combination of equational theories with shared constants and its application to primal algebras. In A. Voronkov, editor, *Proceedings of the Conference on Logic Programming and Automated Reasoning*, Lecture Notes in Artificial Intelligence, pages 261–272, St. Petersburg, Russia, 1992. Springer-Verlag.
58. Ch. Ringeissen. Cooperation of decision procedures for the satisfiability problem. In F. Baader and K.U. Schulz, editors, *Frontiers of Combining Systems: Proceedings of the 1st International Workshop, Munich (Germany)*, Applied Logic, pages 121–140. Kluwer, March 1996.
59. W.C. Rounds. Set values for unification-based grammar formalisms and logic programming. Research Report CSLI-88-129, CSLI, Stanford, 1988.
60. M. Schmidt-Schauß. Unification in a combination of arbitrary disjoint equational theories. *J. Symbolic Computation*, 8(1,2):51–99, 1989.
61. K.U. Schulz. Makanin's algorithm for word equations: Two improvements and a generalization. In K.U. Schulz, editor, *Proceedings of the 1st International Workshop on Word Equations and Related Topics (IWWERT '90)*, volume 572 of *Lecture Notes in Computer Science*, pages 85–150, Berlin, Germany, October 1992. Springer-Verlag.
62. K.U. Schulz. Tractable and intractable instances of combination problems for unification and disunification. *J. Logic and Computation*, 10(1):105–135, 2000.
63. K.U. Schulz. Why combined decision procedures are often intractable. In H. Kirchner and Ch. Ringeissen, editors, *Proceedings of the 3rd International Workshop on Frontiers of Combining Systems (FroCoS 2000)*, volume 1794 of *Lecture Notes in Artificial Intelligence*, pages 217–244, Nancy, France, 2000. Springer-Verlag.
64. R.E. Shostak. Deciding combinations of theories. *Journal of the ACM*, 31:1–12, 1984.
65. J.H. Siekmann and P. Szabó. The undecidability of the D_A-unification problem. *J. Symbolic Computation*, 54(2):402–414, 1989.
66. M. E. Stickel. A complete unification algorithm for associative-commutative functions. In *Proceedings of the 4th International Joint Conference on Artificial Intelligence*, pages 71–82, Tblisi, USSR, 1975.
67. M.E. Stickel. A unification algorithm for associative commutative functions. *J. of the ACM*, 28(3):423–434, 1981.
68. M.E. Stickel. Automated deduction by theory resolution. *J. Automated Reasoning*, 1(4):333–355, 1985.
69. E. Tidén. Unification in combinations of collapse-free theories with disjoint sets of function symbols. In J.H. Siekmann, editor, *Proceedings of the 8th International Conference on Automated Deduction*, volume 230 of *Lecture Notes in Computer Science*, pages 431–449, Oxford, UK, 1986. Springer-Verlag.
70. E. Tidén and S. Arnborg. Unification problems with one-sided distributivity. *J. Symbolic Computation*, 3(1–2):183–202, 1987.
71. C. Tinelli. *Combining Satisfiability Procedures for Automated Deduction and Constraint-based Reasoning*. Phd thesis, Department of Computer Science, University of Illinois, Urbana-Champaign, Illinois, 1999.
72. C. Tinelli and M. Harandi. A new correctness proof of the Nelson-Oppen combination procedure. In F. Baader and K.U. Schulz, editors, *Frontiers of Combining Systems: Proceedings of the 1st International Workshop, Munich (Germany)*, Applied Logic, pages 103–120. Kluwer, March 1996.
73. C. Tinelli and Ch. Ringeissen. Non-disjoint unions of theories and combinations of satisfiability procedures: First results. Technical Report UIUCDCS-R-98-2044,

Department of Computer Science, University of Illinois at Urbana-Champaign, April 1998. (also available as INRIA research report no. RR-3402).
74. Y. Toyama. On the Church-Rosser property for the direct sum of term rewriting systems. *J. ACM*, 34:128–143, 1987.
75. K. Yelick. Unification in combinations of collapse-free regular theories. *J. Symbolic Computation*, 3(1,2):153–182, 1987.