Chapter 8

Unification Theory

Franz Baader

Wayne Snyder

Second readers: Paliath Narendran, Manfred Schmidt-Schauss, and Klaus Schulz.

Contents

HANDBOOK OF AUTOMATED REASONING
Edited by Alan Robinson and Andrei Voronkov

1. Introduction

Unification is a fundamental process upon which many methods for automated deduction are based. Unification theory abstracts from the specific applications of this process: it provides formal definitions for important notions like instantiation, most general unifier, etc., investigates properties of these notions, and provides and analyzes unification algorithms that can be used in different contexts. In this introductory section, we will first present the concept of unification in an informal way, then make some historical remarks on where unification was originally introduced, and finally explain our approach to writing this chapter.

1.1. What is unification?

Very generally speaking, unification tries to identify two symbolic expressions by replacing certain sub-expressions (variables) by other expressions. To be more concrete, one usually considers *terms* that are built from function symbols (say f, a, and b, where f is binary and a, b are nullary) and variable symbols (say x and y). The *unification problem* for the terms $s = f(a, x)$ and $t = f(y, b)$ is concerned with the following question: is it possible to replace the variables x, y in s and t by terms such that the two terms obtained this way are (syntactically) equal. In this example, if we substitute b for x and a for y, we obtain the *unified term* $f(a, b)$. This substitution is denoted as $\sigma := \{x \mapsto b,\, y \mapsto a\}$, and its application to terms is written suffix, i.e., $s\sigma = f(a, b) = t\sigma$. Note that different occurrences of the same variable in a unification problem must always be replaced by the same term. For this reason, the terms $s' = f(a, x)$ and $t' = f(x, b)$ cannot be unified since this would require the occurrence of x in s' to be replaced by b, and the occurrence of x in t' to be replaced by the different constant a.

In most applications of unification, one is not just interested in the *decision problem* for unification, which simply asks for a "yes" or "no" answer to the question of whether two terms s and t are unifiable. If they are unifiable, one would like to construct a solution, i.e., a substitution that identifies s and t. Such a substitution is called a *unifier* of s and t. In general, a unification problem may have infinitely many solutions; e.g., $f(x, y)$ and $f(y, x)$ can be unified by replacing x and y by the same term s (and there are infinitely many terms available). Fortunately, the applications of unification in automated deduction do not require the computation of all unifiers. It is sufficient to consider the so-called *most general unifier*, i.e., a unifier such that every other unifier can be obtained by *instantiation*. In the above example, $\sigma := \{x \mapsto y\}$ is such a most general unifier since for all terms s we have $\{x \mapsto s,\, y \mapsto s\} = \sigma\{y \mapsto s\}$. A unification algorithm should thus not only decide solvability of a given unification problem: if the problem is solvable, it should also compute a most general unifier. As we will show, there exist very efficient algorithms for this purpose.

Unification as described until now is called *syntactic* unification of *first-order* terms. "Syntactic" means that the terms must be made syntactically equal, whereas "first-order" expresses the fact that we do not allow for higher-order variables, i.e., variables for functions. For example, the terms $f(x,a)$ and $g(a,x)$ obviously cannot be made syntactically equal by first-order unification. However, $f(x,a)$ and $G(a,x)$ can be made equal by *higher-order unification* if G is a (higher-order) variable, which may be replaced by f. We will not consider higher-order unification in more detail since it is treated in [Dowek 2001] (Chapter 16 of this Handbook). However, *equational unification*—as opposed to syntactic unification—of first-order terms will be one of the most important topics of this chapter. Instead of requiring that the terms are made syntactically equal, equational unification is concerned with making terms equivalent with respect to a congruence induced by certain equational axioms E. In this case, one talks about E-unification or unification modulo E. For example, if $E = \{f(a,a) \approx g(a,a)\}$, then the terms $f(x,a)$ and $g(a,x)$, which are not (syntactically) unifiable, are E-unifiable: for the substitution $\sigma := \{x \mapsto a\}$, we have $f(x,a)\sigma = f(a,a) =_E g(a,a) = g(a,x)\sigma$, where $=_E$ denotes the equational theory induced by E. For equational unification, things are not as nice as for syntactic unification. In fact, depending on the theory E in question, E-unifiability may be undecidable, and even if it is decidable, solvable unification problems need not have a most general E-unifier. Research on equational unification is, on the one hand, interested in classifying equational theories of interest according to their behavior in this respect. On the other hand, it develops general approaches and algorithms that apply to whole classes of theories.

1.2. History and applications

The name "unification" and the first formal investigation of this notion is due to J.A. Robinson [1965], who introduced unification as the basic operation of his resolution principle, showed that unifiable terms have a most general unifier, and described an algorithm for computing this unifier. In the propositional case, the resolution principle can be described as follows, see also [Bachmair and Ganzinger 2001] (Chapter 2 of this Handbook). Assume that clauses $C \vee p$ and $C' \vee \neg p$ have already been derived (where C, C' are sub-clauses and p is a propositional variable). Then one can also deduce $C \vee C'$. In the first-order case, the clauses one starts with may contain variables. Herbrand's famous theorem implies that finitely many ground instances (i.e., instances obtained by substituting all variables by terms without variables) are sufficient to show unsatisfiability of a given unsatisfiable set of clauses by propositional reasoning (e.g., propositional resolution). The problem is, however, to find the appropriate instantiations. Early theorem provers approached this problem by a breadth-first enumeration of all possible ground instantiations, which led to an immediate combinatorial explosion [Robinson 1963]. Theorem provers based on the resolution principle need not search blindly for the right instantiations: they can compute them via syntactic unification. For example, assume the clauses $C \vee P(s)$ and $C' \vee \neg P(t)$ are given. Obviously, the resolution rule applies to ground instances of these clauses iff in these instances the predicate P contains the same term, i.e., iff the substitution used in the instantiation process is a (syntactic) unifier of s and t.

Instead of using all ground unifiers for instantiation, Robinson proposed to lift the resolution principle to terms with variables, and apply only the most general unifier σ of s and t. In the example, this yields the resolvent $(C \vee C')\sigma$. The completeness proof for propositional resolution can be lifted to non-ground resolution by using the fact that every ground unifier of s, t is an instance of the most general unifier. In fact, the notion "most general unifier" was defined in this way just to make this lifting possible.

Similar ideas for determining appropriate instantiations have been proposed prior to Robinson by Post, Herbrand [1930*a*, 1930*b*, 1967, 1971] (in the investigation of his property A), Prawitz [1960], and Guard [1964, 1969]. However, in this previous work, the notions "unification" and "most general unifier" are not singled out as interesting concepts of their own (they don't even receive a name). Prawitz only considers the function-free case (in which unification is rather trivial), and Herbrand also first presents his approach for this restricted case. The description by Herbrand of the unification algorithm for the general case (which appears to be the first published account of such an algorithm, and which is similar to the transformation-based algorithm by Martelli and Montanari [1982]) is rather informal, and there is no proof of correctness.[1]

The notions "unification" and "most general unifier" were independently re-invented by Knuth and Bendix [1970] as a tool for testing term rewriting systems for local confluence by computing critical pairs. Again, the definition of the most general unifier makes sure that every critical situation is an instance of a critical pair, and thus it is sufficient to test the critical pairs for confluence, see [Dershowitz and Plaisted 2001] (Chapter 9 of this Handbook). *Equational unification* was introduced both in resolution-based theorem proving and in term rewriting as a means for treating certain troublesome equational axioms (like associativity and commutativity) in a special manner. In automated theorem proving, it quickly became apparent that the equality relation requires a special treatment (see [Degtyarev and Voronkov 2001*a*, Nieuwenhuis and Rubio 2001], Chapters 10 and 7 of this Handbook), since a simple integration of axioms that describe the properties of equality (in principle, being a congruence relation) often leads to an unacceptable increase in the search space. Whereas the first approaches providing such a special treatment of equality replaced only the axiomatization of equality by special inference rules, Plotkin [1972] proposed to go one step further. In his approach, also certain axioms that use equality (like $f(x, y) \approx f(y, x)$ and $f(f(x, y), z) \approx f(x, f(y, z))$) can be built into the inference rule (namely resolution). This is achieved by replacing the use of syntactic unification in the resolution step by equational unification, i.e., unification modulo the equational theory induced by the axioms to be built in.

In term rewriting, axioms like commutativity (i.e., $f(x, y) \approx f(y, x)$) cannot be oriented into terminating rewrite rules. One way of solving this problem is to take such non-orientable identities completely out of the rewrite process, and perform

[1] Strictly speaking, Herbrand's unification algorithm is not an algorithm for simple syntactic unification, but an algorithm for unification with so-called linear constant restrictions (see section 3.3.2). This is due to the fact that he does not Skolemize his formulae, and thus he has both universal and existential quantifiers in the quantifier prefix.

rewriting with respect to the remaining (orientable) rules modulo the unoriented ones. In this setting, critical pairs must now be computed by equational unification, i.e., modulo the unoriented identities, see, e.g., [Peterson and Stickel 1981, Jouannaud and Kirchner 1986] and [Dershowitz and Plaisted 2001] (Chapter 9 of this Handbook).

1.3. Approach

This chapter is not intended to give a complete coverage of all the results obtained in unification theory (see the overview articles [Jouannaud and Kirchner 1991, Baader and Siekmann 1994] for this purpose). Instead we try to cover a number of significant topics in more detail. This should give a feeling for unification research and its methodology, provide the most important references, and enable the reader to study recent research papers on the topic.

Notational and typographic conventions
We will try to keep as close as possible to the typographic conventions introduced by Dershowitz and Jouannaud [1991], which they also used in their survey article on rewrite systems [Dershowitz and Jouannaud 1990]. In particular, substitutions are written in suffix notation (i.e., $s\sigma$ instead of $\sigma(s)$), and consequently composition of substitution should be read from left to right (i.e., $\sigma\tau$ means: first apply σ and then τ).

Equational axioms (written $s \approx t$) that define equational theories will be called "identities," whereas unification problems consist of "equations" (written $s =^? t$ for syntactic unification and $s =^?_E t$ for unification modulo E). Thus, identities must hold, whereas equations must be solved.

2. Syntactic unification

As mentioned earlier, syntactic unification of first-order terms was introduced by Post and Herbrand in the early part of this century. Various researchers have studied the problem further [Champeaux 1986, Corbin and Bidoit 1983, Huet 1976, Martelli and Montanari 1982, Paterson and Wegman 1978, Robinson 1971, Venturini-Zilli 1975] and, among other results, it was shown that linear time algorithms for unification exist [Martelli and Montanari 1976, Paterson and Wegman 1978]. The corresponding lower complexity bound was shown by Dwork, Kanellakis and Mitchell [1984]: the unification problem is log-space complete for P, the class of polynomial-time solvable problems. In particular, this implies that it is very unlikely that an efficient parallel unification algorithm exists.

In this section we review the major approaches to syntactic unification.

2.1. Definitions

A *signature* is a (finite or countably infinite) set of function symbols $\mathcal{F}$. We assume the reader is familiar with the term algebra $\mathcal{T}(\mathcal{F}, \mathcal{V})$ generated by a signature function symbols $\mathcal{F}$ and a (countably) infinite set of variables $\mathcal{V}$; we shall call these *$\mathcal{F}$-terms*, or simply *terms* when $\mathcal{F}$ is unimportant, and denote them by the letters l, r, s, t, u, and v. Variables will be denoted by w, x, y, and z. The set of variables occurring in a term t will be denoted by $\mathcal{V}ars(t)$, and this will be extended to sets of variables, equations, and sets of equations.

A substitution is a mapping from variables to terms which is almost everywhere equal to the identity, and will generally be represented by σ, θ, η, or ρ. The identity substitution is represented by Id. The application of a substitution σ to a term t, denoted $t\sigma$, is defined by induction on the structure of terms:

$$t\sigma \;:=\; \begin{cases} x\sigma & \text{if } t = x, \\ f(t_1\sigma, \ldots, t_n\sigma) & \text{if } t = f(t_1, \ldots, t_n). \end{cases}$$

In the second case of this definition, $n = 0$ is allowed: in this case, f is a constant symbol and $f\sigma = f$. Substitutions can also be applied to sets of terms, equations, and sets of equations, in the obvious fashion.

For a substitution σ, the *domain* is the set of variables

$$\mathcal{D}om(\sigma) \;:=\; \{\, x \mid x\sigma \neq x \,\},$$

the *range* is the set of terms

$$\mathcal{R}an(\sigma) \;:=\; \bigcup_{x \in \mathcal{D}om(\sigma)} \{x\sigma\},$$

and the set of variables occurring in the range is $\mathcal{VR}an(\sigma) \;:=\; \mathcal{V}ars(\mathcal{R}an(\sigma))$. A substitution can be represented explicitly as a function by a set of bindings of variables in its domain, e.g.,

$$\{\, x_1 \mapsto s_1, \ldots, x_n \mapsto s_n \,\}.$$

The *restriction* of a substitution θ to a set of variables X, denoted by $\theta|_X$, is the substitution which is equal to the identity everywhere except over $X \cap \mathcal{D}om(\sigma)$, where it is equal to σ. *Composition* of two substitutions is written $\sigma\theta$, and is defined by

$$t\sigma\theta \;=\; (t\sigma)\theta.$$

An algorithm for constructing the composition $\sigma\theta$ of two substitutions represented as sets of bindings is as follows:

1. Apply θ to every term in $\mathcal{R}an(\sigma)$ to obtain σ_1;
2. Remove from θ any binding $x \mapsto t$, where $x \in \mathcal{D}om(\sigma)$, to obtain θ_1;
3. Remove from σ_1 any trivial binding $x \mapsto x$, to obtain σ_2; and

4. Take the union of the two sets of bindings σ_2 and θ_1.

It is also useful to be able to represent substitutions in their *triangular form* as a sequential list of bindings, e.g.,

$$[\, x_1 \mapsto t_1;\, x_2 \mapsto t_2; \ldots;\, x_n \mapsto t_n \,],$$

which represents the composition of n substitutions each consisting of a single binding:

$$\{\, x_1 \mapsto t_1 \,\}\{\, x_2 \mapsto t_2 \,\} \ldots \{\, x_n \mapsto t_n \,\}.$$

A substitution is *idempotent* if $\sigma\sigma = \sigma$; it is easy to show that this is true iff $\mathcal{D}om(\sigma) \cap \mathcal{VR}an(\sigma) = \emptyset$.

A *variable renaming* substitution is defined as a substitution σ such that $\mathcal{D}om(\sigma) = \mathcal{R}an(\sigma)$. (For example, $\{x \mapsto y, y \mapsto z, z \mapsto x\}$ is a variable renaming, whereas $\{x \mapsto y\}$ and $\{y \mapsto z, x \mapsto z\}$ are not.) For any such variable renaming $\rho = \{x_1 \mapsto y_1, \ldots, x_n \mapsto y_n\}$, we denote its inverse $\{y_1 \mapsto x_1, \ldots, y_n \mapsto x_n\}$ by ρ^{-1}.

Two substitutions are equal, denoted $\sigma = \theta$, if $x\sigma = x\theta$ for every variable x. We say that σ is *more general than* θ, denoted $\sigma \mathrel{\dot{\leq}} \theta$, if there exists an η such that $\theta = \sigma\eta$. The relation $\dot{\leq}$ is called the *instantiation quasi-ordering*. The corresponding equivalence relation (i.e., $\dot{\leq} \cap \dot{\geq}$) is denoted by $\dot{=}$; it can be shown [Lassez, Maher and Mariott 1987] that $\sigma \mathrel{\dot{=}} \theta$ iff there exists some variable renaming ρ such that $\sigma = \theta\rho$.

2.1. DEFINITION. A substitution σ is a *unifier* of two terms s and t if $s\sigma = t\sigma$; it is a *most general unifier* (or *mgu* for short), if for every unifier θ of s and t, $\sigma \mathrel{\dot{\leq}} \theta$. A *unification problem* for two terms s and t is represented by $s =^? t$.

A *multiset* is an unordered collection with possible duplicate elements. We denote the number of occurrences of an object x in a multiset M by $M(x)$, and define the multiset union $M \cup N$ as the multiset Q such that $Q(x) = M(x) + N(x)$ for every x.

2.2. Unification of terms

In this section and the next, we present a series of algorithms for unification, each of which returns an *mgu* for two unifiable terms. Our approach will be two-sided: on the one hand we will present a series of practical algorithms, from the "naive" to the more sophisticated (and faster), in pseudo-code suitable for implementing in a programming language; and on the other we will present a "rule-based" approach which serves to clarify the essential properties of the process and also to prove the correctness of some of the practical algorithms.

2.2.1. A naive algorithm

The simplest algorithm for unification is perhaps one that is taught in many introductory courses in AI:

Write down two terms and set markers (e.g., two index fingers) at the beginning of the terms. Then:

1. Move the markers together, one symbol at a time, until both move off the end of the term (**success**!), or until they point to two different symbols;
2. If the two symbols are both non-variables, then **fail**; otherwise, one is a variable (call it x) and the other is the first symbol of a subterm (call it t):
   (a) If x occurs in t, then **fail**;
   (b) Otherwise, write down "$x \mapsto t$" as part of the solution, replace x everywhere by t (including in the solution), and return to (1).

This simple algorithm methodically finds disagreements in the two terms to be unified, and attempts to repair them by binding variables to terms: it fails when function symbols clash, or when an attempt is made to unify a variable with a term containing that variable (which is impossible). Already present in this simple algorithm are several interesting issues:

Implementation: What data structures should be used for terms and substitutions? How should application of a substitution be implemented? What order should the operations be performed in?

Correctness: Does the algorithm always terminate? Does it always produce an *mgu* for two unifiable terms, and fail for non-unifiable terms? Do these answers depend on the order of operations?

Complexity: How much space does this take, and how much time?

In the remainder of this section we will consider these issues in detail while developing our sequence of algorithms.

2.2.2. Unification by recursive descent

If we take our naive algorithm and implement it as simply as possible in a programming language, then we would represent terms using either explicit pointer structures (as in C or Pascal) or built-in recursive data types (as in ML and Lisp), and represent substitutions as lists of pairs of terms. Application of a substitution would involve constructing a new term or replacing a variable with a new term. The left-to-right search for disagreements would then be implemented by recursive descent through the terms as shown in Figure 1.

(In an actual implementation, the case "Unify(t, s)" could be moved up before the first "**else if**" and simply swap s and t if the former is not a variable.) The only detail that might cause some confusion is the exact method for implementing the composition in the last line. This was described in section 2.1; however, in our naive unification algorithm, we omitted the second and third steps from the informal algorithm for composition, and this may be done as well here, due to a simple but important fact about these algorithms: when a binding $x \mapsto t$ is created and applied, x will never appear in another term considered by the algorithm—x has been "eliminated" and occurs only once, in the solution.

This algorithm is essentially the one first described by Robinson [1965], and has been almost universally used in symbolic computation systems.

```
global σ : substitution;   { Initialized to Id }

Unify( s : term; t : term )
  begin
    if s is a variable then   { Instantiate variables }
      s := sσ;
    if t is a variable then
      t := tσ;
    if s is a variable and s = t then
      { Do nothing }
    else if s = f(s_1, ..., s_n) and t = g(t_1, ..., t_m) for n, m ≥ 0 then begin
      if f = g then
        for i := 1 to n do
          Unify( s_i, t_i );
      else Exit with failure   { Symbol clash }
    end
    else if s is not a variable then
      Unify( t, s );
    else if s occurs in t then
      Exit with failure;   { Occurs check }
    else σ := σ{s ↦ t};
  end;
```

Figure 1: Unification by recursive descent

2.2.3. A rule-based approach $\mathcal{U}$

In order to explore some of the logical properties of this algorithm, we now present a simple inference system for deriving solutions for unification problems.

An idempotent substitution $\{x_1 \mapsto t_1, \ldots, x_n \mapsto t_n\}$ may be represented by a set of equations $\{x_1 \approx t_1, \ldots, x_n \approx t_n\}$ in *solved form*, i.e., where each x_i has a single occurrence in the set. For any idempotent substitution σ, the corresponding solved form set will be denoted by $[\sigma]$, and for any set of equations S in solved form, the corresponding substitution will be denoted by σ_S.

A *system* is either the symbol $\perp$ (representing failure) or a pair consisting of a multiset P of unification problems and a set S of equations in solved form. We will use Γ to denote an arbitrary system. A substitution is said to be a unifier (or solution) of a system $P; S$ if it unifies each of the equations in P and S; the system $\perp$ has no unifiers.

The inference system $\mathcal{U}$ consists of the following transformations on systems:[2]

[2]The symbol $\cup$ below when applied to P is *multiset union*.

Trivial:

$$\{s \stackrel{?}{=} s\} \cup P'; S \implies P'; S$$

Decomposition:

$$\{f(s_1, \ldots, s_n) \stackrel{?}{=} f(t_1, \ldots, t_n)\} \cup P'; S \implies \{s_1 \stackrel{?}{=} t_1, \ldots, s_n \stackrel{?}{=} t_n\} \cup P'; S$$

(Note that possibly $n = 0$.)

Symbol Clash:

$$\{f(s_1, \ldots, s_n) \stackrel{?}{=} g(t_1, \ldots, t_m)\} \cup P'; S \implies \bot$$

if $f \neq g$.

Orient:

$$\{t \stackrel{?}{=} x\} \cup P'; S \implies \{x \stackrel{?}{=} t\} \cup P'; S$$

if t is not a variable.

Occurs Check:

$$\{x \stackrel{?}{=} t\} \cup P'; S \implies \bot$$

if $x \in \mathcal{V}ars(t)$ but $x \neq t$.

Variable Elimination:

$$\{x \stackrel{?}{=} t\} \cup P'; S \implies P'\{x \mapsto t\}; S\{x \mapsto t\} \cup \{x \approx t\}$$

if $x \notin \mathcal{V}ars(t)$.

In order to unify s and t, we create an initial system $\{s =^? t\}; \emptyset$ and apply successively rules from $\mathcal{U}$; we show below that such a process must terminate, producing a terminal system (i.e., to which no rule applies) in the form of $\bot$ or $\emptyset; S$, where S is a solved form system representing the *mgu* of s and t.

The inference system $\mathcal{U}$ is in essence the same algorithm for unification presented by Herbrand (see Appendix 3 in [Herbrand 1971]); more recently, this formulation of the unification process was introduced by Martelli and Montanari [1982] and has gained wide currency as a formalism for representing unification algorithms (see, for example, [Jouannaud and Kirchner 1991, Snyder 1991]).

Before considering $\mathcal{U}$ *per se*, let us consider how this set of transformations might simulate the actions of the recursive descent algorithm. Suppose we were to print out a trace of the terms s and t, and the global substitution σ, just before the third *if*-statement in *Unify*, e.g.,

$$\begin{array}{lll} s_1 & t_1 & Id \\ s_2 & t_2 & \sigma_2 \\ s_3 & t_3 & \sigma_3 \\ \ldots & & \end{array}$$

This sequence can be simulated by a sequence of transformations

$$\begin{array}{ll} & \{s_1 =^? t_1\}; \emptyset \\ \Longrightarrow & \{s_2 =^? t_2\} \cup P_2; S_2 \\ \Longrightarrow & \{s_3 =^? t_3\} \cup P_3; S_3 \\ \Longrightarrow & \ldots \end{array}$$

where each $s_i =^? t_i$ is the equation acted on by the rule, and each σ_i is identical to σ_{S_i}. Furthermore, if the call to Unify ends in failure, then the transformation sequence ends in $\perp$; and if the call to Unify terminates with success, with a global substitution σ_n, then the transformation sequence ends in a system $\emptyset; S$ where $\sigma_S = \sigma_n$. This simulation can be achieved by treating the multiset P as a stack, always applying a rule to the top equation, and only using Trivial when s is a variable; there is only one possible rule to apply at each step under this control strategy.

Therefore, $\mathcal{U}$ can be viewed as an abstract version of the recursive descent algorithm, and can be used to prove its correctness. In fact, $\mathcal{U}$ has many interesting features in its own right, as we now proceed to show.

2.2.4. Technical results about $\mathcal{U}$

In this section we present a number of results about $\mathcal{U}$, adapted from Martelli and Montanari [1982]. Perhaps the simplest property to show is termination.

2.2. LEMMA. *For any finite multiset of equations P, every sequence of transformations in $\mathcal{U}$*

$$P; \emptyset \implies P_1; S_1 \implies P_2; S_2 \implies \ldots$$

terminates either with $\perp$ or with $\emptyset; S$, with S in solved form.

PROOF. Define a complexity measure $\langle n_1, n_2, n_3 \rangle$ on multisets of equations, ordered by the (well-founded) lexicographic ordering on triples of natural numbers, where

$n_1 =$ The number of distinct variables in P;
$n_2 =$ The number of symbols in P; and
$n_3 =$ The number of equations in P of the form $t =^? x$, with t not a variable.

Each rule in $\mathcal{U}$ reduces the complexity of the problem P. Furthermore, any equation must fit into one of the cases mentioned on the left-hand sides of the rules, so that a rule can always be applied to a system with non-empty P. Thus, a system to which no rule applies must be in the form $\perp$ or $\emptyset; S$. Since whenever an equation is added to S, the variable on the left-side is eliminated from the rest of the system, each of the systems $S_1, S_2, \ldots, S$ must be in solved form. $\square$

Another interesting fact is that a solution σ produced by $\mathcal{U}$ is always idempotent.

2.3. Corollary. *If* $P; \emptyset \implies^{+} \emptyset; S$*, then* σ_S *is idempotent.*

One of the most interesting features of $\mathcal{U}$ is that its rules do not change the set of unifiers of a system. The main correctness results about $\mathcal{U}$ are essentially corollaries of this fact.

2.4. Lemma. *For any transformation* $P; S \implies \Gamma$*, a substitution* θ *unifies* $P; S$ *iff it unifies* Γ*.*

Proof. The only non-trivial cases concern Occurs Check and Variable Elimination. If x occurs in, but is not equal to, t, then clearly x contains fewer symbols than t; but then $x\theta$ must also contain fewer symbols than $t\theta$, so that x and t can have no unifier.

Regarding Variable Elimination, we know that $x\theta = t\theta$, from which (by structural induction) we can show that

$$u\theta \;=\; (u\{x \mapsto t\})\theta$$

for any term u, or indeed for any equation or multiset of equations. But then

$$P'\theta \;=\; P'\{x \mapsto t\}\theta \quad \text{and} \quad S\theta \;=\; S\{x \mapsto t\}\theta$$

from which the result follows. □

The first of our major results about $\mathcal{U}$ shows that it does indeed produce a unifier.

2.5. Theorem. *(Soundness) If* $P; \emptyset \implies^{+} \emptyset; S$*, then* σ_S *unifies every equation in* P*.*

Proof. Note that σ_S unifies S, because it is idempotent; a simple induction with lemma 2.4 shows that σ_S must unify the equations in P. □

Our second major result shows that $\mathcal{U}$ is able to calculate an *mgu* for two unifiable terms.

2.6. Theorem. *(Completeness) If* θ *unifies every equation in* P*, then* any *maximal sequence of transformations*

$$P; \emptyset \implies \ldots$$

must end in some system $\emptyset; S$ *such that* $\sigma_S \lessdot \theta$*.*

Proof. Lemmas 2.2 and 2.4 show that such a sequence must end in some terminal system $\emptyset; S$ where θ unifies S. Now for every binding $x \mapsto t$ in σ_S,

$$x\sigma_S\theta \;=\; t\theta \;=\; x\theta,$$

and for every $x \notin \mathcal{D}om(\sigma_S)$, $x\sigma_S\theta \;=\; x\theta$, so that $\theta \;=\; \sigma_S\theta$. □

An immediate consequence of these two results is the following.

2.7. COROLLARY. *If P has no unifier, then any maximal transformation sequence from $P;\emptyset$ must have the form*

$$P;\emptyset \Longrightarrow \ldots \Longrightarrow \bot.$$

The most interesting feature of this proof (and the reason for the emphasis on the word "any") is that the choice of a rule to apply at any stage of the computation is *don't care non-deterministic*, which implies that any control strategy will result in an *mgu* for two unifiable terms, and failure for two non-unifiable terms. Thus, any practical unification algorithm which proceeds by performing the atomic actions of $\mathcal{U}$, in any order, is sound and complete, and in particular it generates idempotent *mgu*s for unifiable terms. However, some sequences of these basic operations may be longer than others, or create larger terms, and not all sequences end in the same exact *mgu*. Before considering the issue of complexity in detail, we digress for a moment to consider this last point.

2.2.5. Some properties of MGU's

Theorem 2.6 shows that any substitution produced by $\mathcal{U}$ (or any algorithm that $\mathcal{U}$ can simulate) is a compact representation of the (infinite) set of all unifiers, which could be generated by composing all possible substitutions with the *mgu*. This means that no information is lost in symbolic computation systems (such as first-order theorem provers and logic-programming interpreters) in solving a unification subproblem and applying the solution to the rest of the computation (this is what happens, in fact, during the unification process itself).

The inference system $\mathcal{U}$, starting from a single pair of terms s and t, could produce (finitely) many different terminal forms, corresponding to distinct *mgu*s for s and t. What is the relationship of these distinct *mgu*s? Are there other *mgu*s than these? Is there an infinite number? The key to answering these questions lies in the concept of a variable renaming, defined in section 2.1: if σ and θ are both *mgu*s of s and t, then $\sigma \doteq \theta$, i.e., they are instances of each other, and hence $\sigma = \theta\rho$ for some variable renaming ρ (for a proof, see [Lassez et al. 1987].)

This means that the set of *mgu*s of two terms can be generated from a single *mgu*, by composition with variable renamings (which is a special case of the fact that the set of all unifiers can be generated by composition with arbitrary substitutions). By such an operation, it is possible to create an infinite number of idempotent *mgu*s and an infinite number of non-idempotent *mgu*s; the finite search tree generated by $\mathcal{U}$ is not able to construct any arbitrary *mgu*, nor even every idempotent *mgu*.

An oft-repeated phrase in the literature states that "*mgu*s are unique up to renaming"; the reader should now understand that this vague statement should more properly be: "*mgu*s are unique up to composition with a variable renaming."

This brief exposition of some of the important properties of *mgu*s should convince the reader that the collection of all unifiers of two terms has non-trivial properties; later on in this chapter we shall examine the even more complex case of sets of unifiers for E-unification problems. For further characterizations of the set of *mgu*s produced by $\mathcal{U}$, and on unifiers in general, see [Lassez et al. 1987, Eder 1985].

2.2.6. Complexity of recursive descent
This section will begin to consider the complexity of the unification process, a question which will motivate the consideration of further, more sophisticated algorithms for unification.

The approaches to unification so far considered, unfortunately, can take exponential time and space.

2.8. EXAMPLE.

$$h(x_1, x_2, \ldots, x_n, f(y_0, y_0), \ldots, f(y_{n-1}, y_{n-1}), y_n)$$

and

$$h(f(x_0, x_0), f(x_1, x_1), \ldots, f(x_{n-1}, x_{n-1}), y_1, \ldots, y_n, x_n)$$

Unifying these two terms will create an *mgu* where each x_i and each y_i is bound to a term with $2^{i+1} - 1$ symbols. Clearly the problem is that the substitution contains many duplicate copies of the same subterms. One idea that might help here would be to represent substitutions as "triangular forms." Thus,

$$[\, y_0 \mapsto x_0; y_n \mapsto f(y_{n-1}, y_{n-1});\, y_{n-1} \mapsto f(y_{n-2}, y_{n-2}); \ldots]$$

would be a triangular form unifier of the two terms. Building up such a substitution during unification consists of simply collecting a list of bindings; no duplicate terms are created, and hence triangular form unifiers can be no larger than the original problem.

Unfortunately, this good idea is not sufficient to rescue the algorithm, as it appears that substitution, and hence the duplication of subterms, is necessary in the terms themselves: in the example, the call to *Unify* on the last arguments, x_n and y_n, which by then are bound to terms with $2^{n+1} - 1$ symbols, will lead to an exponential number of recursive calls. The solution to this problem is to develop a more subtle data structure for terms, and a different method for applying substitutions.

2.3. Unification of term dag*s*

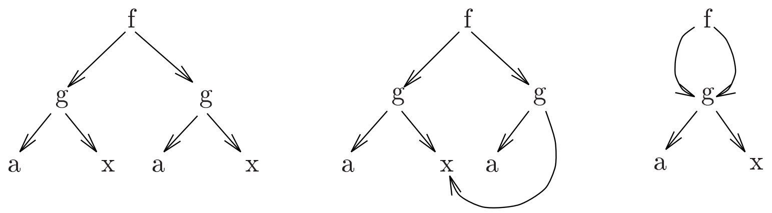
In this section, we consider two approaches to speeding up the unification process. The first approach, which we adapt from Corbin and Bidoit [1983], fixes the problem of duplication of subterms created by substitution by using a graph representation of terms which can share structure; this results in a quadratic algorithm. To develop an asymptotically faster algorithm, however, it is necessary to abandon the recursive descent approach, and recast the problem of unification as the construction of a certain kind of equivalence relation on graphs. This second approach is due to Huet [1976].

2.3.1. Term dag*s and substitution*
Concerning example 2.8, it should be remarked that the explosion in the size of the terms occurred precisely because there were duplicate occurrences of the same

variables, which cause a duplication of ever larger and larger terms. In order to fix this problem, it is necessary to consider in detail how to represent terms as explicit graphs which share subterms.

2.9. DEFINITION. A *term dag* is a directed, acyclic graph whose nodes are labeled with function symbols, constants, or variables, whose outgoing edges from any node are ordered, and where the outdegree of any node labelled with a symbol f is equal to the arity of f (variables have outdegree 0).

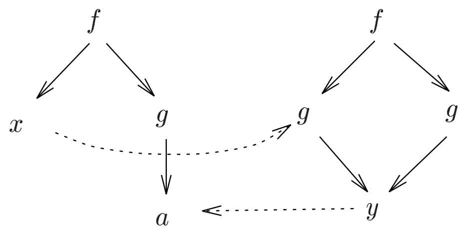
In such a graph, each node has a natural interpretation as a term, and we shall speak of nodes and terms as if they were one and the same (e.g., a "node" $f(a, x)$ is one labeled with f and having arcs to nodes a and x). The only difference between various *dag*s representing a particular term is the amount of *structure sharing* among the subterms. For example, we could represent the term $f(g(a, x), g(a, x))$ by any of the following *dag*s:



Assuming that names of symbols are strings of characters, it is possible to create a *dag* with unique, shared occurrences of variables in $O(n)$, where n is the number of all characters in the string representation of a unification problem. For example, one can use a *trie* to store the variable names when parsing the terms, so that duplicate occurrences of variables can be pointed to a unique, shared representation of the variable. In the normal case, names have a constant size, and n just represents the number of symbols in the term; we make this assumption in what follows.

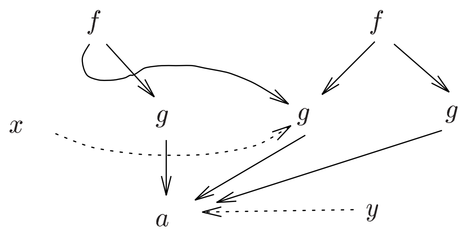
Therefore, we assume that the input to our algorithm is a term *dag* representing the two terms to be unified, with unique, shared occurrences of all variables. We also assume that each node t has an attribute *parents*(t) which is a list of all parents of t in the graph (i.e., all nodes p which point to t), but do not show these in the diagrams below for simplicity. Parent pointers are necessary when sharing nodes in the dag.

A substitution involving only the subterms of a term *dag* can be represented directly by a relation on the nodes of the *dag*, either stored explicitly as a list of pairs of pointers to nodes, or by storing a link (we will call these *substitution arcs*) in the graph itself, and maintaining a list of variables (nodes) bound by the substitution. Application of such a substitution can be implicit or explicit, the latter involving actual moving of subterm links. For example, two terms $f(x, g(a))$ and $f(g(y), g(y))$, and their *mgu* $\{ x \mapsto g(a), y \mapsto a \}$ can be represented by the *dag*:



The *implicit* application of a substitution identifies two nodes connected with a substitution arc, without actually moving any of the subterm links; the binding for a variable can be determined by traversing the graph depth first, left to right. This essentially represents the triangular form (e.g., $[\,x \mapsto g(y);\, y \mapsto a\,]$) in the *dag*. We use this form of substitution in the algorithm of section 2.3.3.

The *explicit* application of a substitution expresses the substitution of binding for variable by moving any arc (subterm or substitution) pointing to a variable to point to the binding. For example,



This represents the "functional" form $\{\,x \mapsto g(a),\, y \mapsto a\,\}$ of the substitution in a direct way. We shall use this explicit form of application in the next algorithm.

2.3.2. Recursive descent on term dag*s*

In this section we present the first algorithm which uses term *dag*s. If we think about tracing the operation of the recursive descent algorithm on this new data structure, it might appear that the source of exponential blowup has been removed, since substitution does not duplicate terms. However, it still may be possible to have duplicate *calls* to the same term; in example 2.8, for instance, the terms bound to x_n and y_n (see fig. 2) will be unified when x_0 is bound to y_0; however, the recursive descent algorithm will then blithely explore every other path through the pair of terms, resulting in an exponential number of recursive calls.

Clearly, we need to keep from revisiting already-solved problems in the graph. The best solution is simply to do structure sharing "on the fly" by merging unified terms (which are, after all, now identical), and then checking for identity of nodes in the first step. Merging two nodes s and t in a graph Δ can be implemented by moving arcs. Let $parents(s) = \{p_1, \ldots, p_n\}$; then

1. For each p_i, replace the subterm arc $p_i \longrightarrow s$ by $p_i \longrightarrow t$;
2. Let $parents(t) := parents(s) \cup parents(t)$; and
3. Let $parents(s) := \emptyset$.

This shares the structure of t and isolates the node s. In the algorithm below, we will denote by Replace(Δ, s, t) the new graph created from a graph Δ by merging

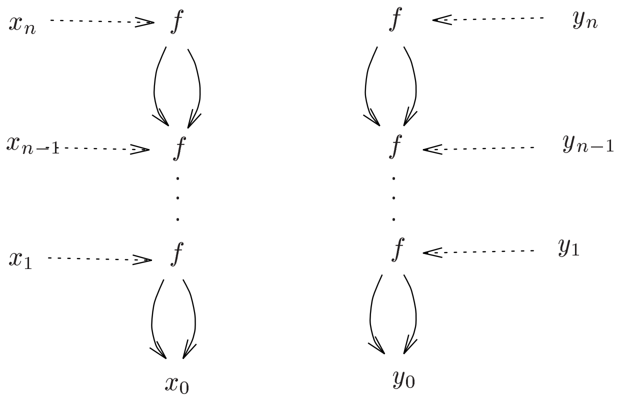


Figure 2: A dag representation of the terms bound to x_n and y_n in example 2.8.

s and t in this fashion.

The algorithm takes as input a term *dag* in which all occurrences of variables are shared (i.e., each variable occurs exactly once). Even with these additions, our recursive descent algorithm is mostly unchanged:

```
global Δ : termDag;   { Term dag for s and t with shared variables }
global σ : list of pairs of nodes;   { Initialized to empty }

UnifyDag( s : node; t : node )
    begin
        if s and t are the same node then
            { Do nothing }
        else if s = f(s_1, ..., s_n) and t = g(t_1, ..., t_m) then begin
            if f = g then
                for i := 1 to n do
                    UnifyDag( s_i, t_i );
            else Exit with failure   { Symbol clash }
        end
        else if s is not a variable then
            Unify( t, s );
        else if s occurs in t then
            Exit with failure;   { Occurs check }
        else
            Add (s,t) to the end of the list σ;
        Δ := Replace(Δ, s, t);   { Since they are now unified }
    end;
```

The occurs check is implemented as a standard graph traversal to search for the given node s below t by following subterm arcs.

The correctness of the data structure for this algorithm is dependent on the following result from Corbin and Bidoit [1983], which can be proved by induction on the *dag*.

2.10. LEMMA. *Let* Δ *be a term* dag *with nodes* x *and* t *such that there is no path from* t *to* x*.*

- *Replace(*Δ*,* x*,* t*) is an acyclic graph containing the same nodes (with the same labels) as* Δ*.*
- *Consider a distinguished node in* Δ *corresponding to the term* s*, and let* s' *be the term corresponding to the same node in Replace(*Δ*,* x*,* t*); then:*
  - *if* $s = x$*, then* $s' = x$*;*
  - *otherwise,* $s' = s\{x \mapsto t\}$*.*

In order to prove soundness and completeness, we may again show that $\mathcal{U}$ can "trace" the terms in each call to UnifyDags, the only difference being that when Trivial is used, s may not necessarily be a variable (i.e., when UnifyDag is called on two terms previously unified, and hence shared as one node). From a logical point of view (thinking in term of the symbolic expressions being manipulated), nothing has changed—only the underlying data structure for terms and substitutions.

Thus, the only thing that remains to be considered is the complexity of UnifyDag. Since each call to this function isolates a node, there can not be more than n calls *in toto* (where n is the number of symbols occurring in the original terms). Each call does a constant amount of work except for the occurs check (which traverses no more than n nodes) and the moving of no more than n pointers. Maintaining the lists of parents costs $O(n)$ at each call. The original construction of the *dag* takes $O(n)$. This results in a time complexity of $O(n^2)$; clearly the space used is $O(n)$.

2.3.3. An almost-linear algorithm

It would be possible to speed up this algorithm by making changes to the way substitutions are represented (see [Baader and Siekmann 1994]), however, we will now consider an alternate approach which gives more insight into the nature of unification. This approach makes the following fundamental changes to the approach considered so far:

- instead of recursive calls to pairs of subterms which must be unified, we will recast the problem as that of constructing an equivalence relation whose classes are terms that must be unified;
- substitution will (in some sense) be replaced by the union of equivalence classes; and
- the repeated calls to the occurs check will be replaced by a single pass through the graph to check for acyclicity.

The term *dag* data structure will be used for these algorithms as well, however, we will not move pointers as in the last section. Instead, we consider the unification problem as one involving the following relation on terms.

2.11. DEFINITION. A *term relation* is an equivalence relation on terms, and is *homogeneous* if no equivalence class contains $f(\ldots)$ and $g(\ldots)$ with $f \neq g$; it is *acyclic* if no term is equivalent to a proper subterm of itself.

A *unification relation* is a homogeneous, acyclic term relation satisfying the *unification axiom*: For any f and terms s_i and t_i,

$$f(s_1, \ldots, s_n) \cong f(t_1, \ldots, t_n) \longrightarrow s_1 \cong t_1 \wedge \ldots \wedge s_n \cong t_n.$$

The *unification closure of s and t*, when it exists, is the least unification relation which makes s and t equivalent.

The algorithm presented in this section takes its starting point from the following fact.

2.12. LEMMA. *If s and t are unifiable, then there exists a unification closure for s and t.*

PROOF. For any unifier θ of s and t, define the relation

$$u \cong_\theta v \text{ iff } u\theta = v\theta.$$

Clearly this is a unification relation. Since the intersection of two unification relations relating s and t is again a unification relation relating s and t, whenever s and t are unifiable there is a *least* such relation $\cong$ which joins classes only when forced to apply the unification axiom to subterms of s and t. □

The unification-closure approach to unification, first presented in [Huet 1976], attempts to construct this relation on two terms, which, as we shall show, corresponds to finding an *mgu*. However, before presenting the algorithm, we need a number of ancillary notions.

2.13. DEFINITION. For any term relation $\cong$, let a *schema function* be a function ς from equivalence classes to terms such that for any class C,

1. $\varsigma(C) \in C$; and
2. $\varsigma(C)$ is a variable only if C consists entirely of variables.

The term $\varsigma(C)$ will be called the *schema term* for C.

The point here is that the schema term is a functional form whenever such exists, and will be used to propagate information downward using the unification axiom; it is also used to define substitutions. Note that schema functions are not unique, but there always exists at least one for any term relation; we assume in the following that such a function has been chosen for any given unification closure.

Note that for any acyclic term relation there exists a partial ordering $\succ$ such that for any term $f(\ldots s \ldots)$, we have $[f(\ldots s \ldots)] \succ [s]$.

2.14. DEFINITION. For any unification closure $\cong$, define $\sigma_\cong$ by:

$$x\sigma_\cong = \begin{cases} y & \text{if } \varsigma([x]) = y \\ f(s_1\sigma_\cong, \ldots, s_n\sigma_\cong) & \text{if } \varsigma([x]) = f(s_1, \ldots, s_n) \end{cases}$$

(This notion is well-defined by recursion on the partial order $\succ$; $\mathcal{D}om(\sigma_{\cong})$ is finite because $\cong$ has only a finite number of non-trivial equivalence classes.)

2.15. Theorem. *Terms s and t are unifiable iff there is a unification closure for s and t. In the affirmative case, $\sigma_{\cong}$ is an* mgu *for s and t.*

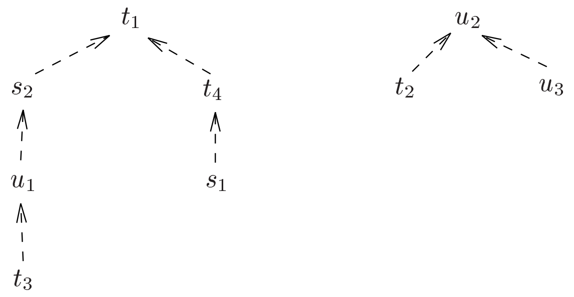
Proof. The *only if* direction has been proved in our previous lemma. For the other direction, let $\cong$ be a unification closure for s and t. We claim that for every term u, $u\sigma_{\cong} = \varsigma([u])\sigma_{\cong}$ (thus, $\sigma_{\cong}$ unifies each pair of equivalent terms, in particular s and t), and proceed by induction on the size of u. For the base case, if u is a constant or variable, then the result is trivial by the definition of $\sigma_{\cong}$. Now suppose that $u = f(s_1, \ldots, s_n)$ and $\varsigma([u]) = f(t_1, \ldots, t_n)$; since $\cong$ is closed under the unification axiom, then for each i, $s_i \cong t_i$, and thus by the induction hypothesis, $s_i\sigma_{\cong} = t_i\sigma_{\cong}$.

To prove that $\sigma_{\cong}$ is an *mgu* in the affirmative case, we show that for any unifier θ, we have $u\sigma_{\cong}\theta = u\theta$ for any term u, and proceed by induction on $\succ$. Assume that $\cong_\theta$ is as defined in the previous lemma. (In the following, ς refers to some fixed schema function for $\sigma_{\cong}$.) First, note that if $u \cong v$, then $u\theta = v\theta$, since $\cong$ is contained in $\cong_\theta$. Now, for the base case, if $[u]$ contains only constants and variables, then $u\sigma_{\cong} = \varsigma([u]) \cong u$, from which it follows that $u\sigma_{\cong}\theta = u\theta$. For the induction step, it must be the case that $\varsigma([u])$ equals some $f(s_1, \ldots, s_n)$, and u is either a term of the form $f(t_1, \ldots, t_n)$, or is a variable x. In the first case, $u\sigma_{\cong}\theta = u\theta$ by a direct use of the induction hypothesis. In the second case, $x\sigma_{\cong} = f(s_1\sigma_{\cong}, \ldots, s_n\sigma_{\cong})$, and $x\theta = f(s_1, \ldots, s_n)\theta$ (since $\cong$ is contained in $\cong_\theta$), so that

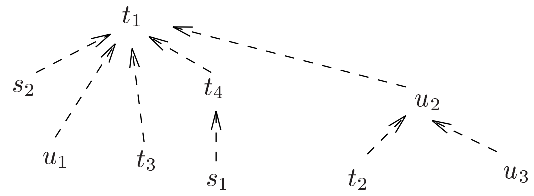
$$x\theta = f(s_1\theta, \ldots, s_n\theta) = f(s_1\sigma_{\cong}\theta, \ldots, s_n\sigma_{\cong}\theta) = f(s_1\sigma_{\cong}, \ldots, s_n\sigma_{\cong})\theta = x\sigma_{\cong}\theta,$$

the second step involving the induction hypothesis. □

This result motivates the design of an efficient unification algorithm which attempts to build a unification closure for two terms, and then extracts the *mgu*. To do this, it is necessary to have some means for maintaining equivalence classes and for applying the unification axiom to classes; the most efficient data structure represents classes as trees of *class pointers* (which we represent by dashed lines) with a class *representative* at the root:



To determine whether two terms are equivalent, it is only necessary to find the roots of the trees and check for identity; and to join two classes, one class is made a subtree of the other's root. To reduce the height of the trees as much as possible, two subtle refinements are made: (i) maintain a count of the size of each class in the representative, and when joining classes, make the smaller one a subtree of the larger; and (ii) when following paths to the root to determine equivalence, compress the paths by pointing all nodes encountered directly at the root. For example, if we wished to take the union of the two classes $[t_3]$ and $[u_3]$, we would find the representatives for the two classes, compressing the path from t_3, and then add a class link from the representative of the smaller class to the larger:



Such a data structure can process a series of $O(n)$ Unions and Finds in $O(n\,\alpha(n))$, where α is the functional inverse of Ackermann's function, and which, for all *practical* purposes, may be considered as a small constant factor.

The term *dag* for this approach needs no parent pointers, as in the previous algorithm, but does need

- *class* pointers;
- a counter of the *size* of the class stored in the representative;
- a pointer from each representative to the *schema* term for the class;
- boolean flags *visited* and *acyclic* in each node used in cycle checking (both initialized to **false**);
- a pointer *vars* from each representative to a list of all variables in the class (used when generating solutions).

Note that maintaining lists of parents of each node is not necessary in this algorithm. A representative is simply a node whose class pointer points to itself. The algorithm based on this approach may now be given. It is shown in Figures 3 and 4. The term *dag* Δ for s and t is initialized to the identity relation, where each class contains a single term; thus for each node the *class* and *schema* pointers are initialized to point to the same node, and the *size* is initialized to 1. The vars list is initialized to empty for non-variable nodes, and to a singleton list for variable nodes.

If Unify$(s,\ t)$ does not fail, then σ contains a triangular form solution. Find-Solution attempts to find such a solution, and fails iff there exists a cycle in the graph. (We are essentially traversing the common term $s\sigma$ by replacing s by its schema term in the first line.) The fields *visited* and *acyclic* are both necessary, the first to find a cycle in the current exploration path, and the second to keep from reexamining nodes which have already been excluded from any possible cycles.

The correctness of this method depends on verifying that it implements correctly the construction of an acyclic unification closure. The essential points are that

- the equivalence is clearly homogeneous;
- equivalence classes are joined iff required by the unification axiom, hence the relation is *least*;

```
global Δ : termDag;   { Term dag for s and t with shared variables }
global σ : list of bindings := nil;   { Triangular form solution }

Unify( s : node; t : node )
    begin
        UnifClosure(s, t);
        FindSolution(s);
    end;

UnifClosure( s : node; t : node )
    begin
        s := Find(s);   { Find representatives }
        t := Find(t);
        if s and t are the same node then
            { Do nothing }
        else begin
            if ς([s]) = f(s_1, ..., s_n) and ς([t]) = g(t_1, ..., t_m) for n, m ≥ 0
            then begin
                if f = g then begin
                    Union(s, t);
                    for i := 1 to n do
                        UnifClosure( s_i, t_i );
                end
                else Exit with failure   { Symbol clash }
            end
            else Union(s, t);
        end;
    end;

Union( s : node; t : node )  { s and t are representatives }
    begin
        if size(s) ≥ size(t) then begin
            size(s) := size(s) + size(t);
            vars(s) := concatenation of lists vars(s) and vars(t);
            if ς([s]) is a variable then
                ς([s]) := ς([t]);
            class(t) := s;
        end
        else begin
            size(t) := size(t) + size(s);
            vars(t) := concatenation of lists vars(t) and vars(s);
                if ς([t]) is a variable then
                    ς([t]) := ς([s]);
                class(s) := t;
        end;
    end;
```

Figure 3: Unification algorithm

```
Find( s : node )   { Returns representative for [s] and compresses paths }
    t : node;
    begin
        if class(s) = s   { s is a representative } then
            Return s;
        else begin
            t := Find(class(s));
            class(s) := t;
            return t;
        end;
    end;

FindSolution(s : node);    { Fails if exists a cycle below s }
    begin;
        s := ς(Find(s));
        if acyclic(s) then
            Return;   { s is not part of a cycle }
        if visited(s) then
            Fail;   { Exists a cycle }
        if s = f(s_1, ..., s_n) for some n > 0 then begin
            visited(s) := true;
            for i := 1 to n do
                FindSolution(s_i);
            visited(s) := false;
        end;
        acyclic(s) := true;
        foreach x ∈ vars(Find(s)) do
            if x ≠ s then
                Add [x ↦ s] to front of σ;
    end;
```

Figure 4: Unification algorithm, continued

- FindSolution fails iff there is a cycle in the graph; and
- whenever a binding $[x \mapsto s]$ is added to σ, all relevant bindings for variables in s already occur in σ.

The complexity of the algorithm is $O(n\,\alpha(n))$, as, with the exception of **Find**, each function can be called at most n times for terms with n symbols, and each call performs a constant amount of work (note that the work of concatenating the *vars* lists can be accomplished in $O(n)$ if pointers to the last cell in the list are kept, and concatenation is performed by moving pointers rather than by the standard append operation). The dominating cost is therefore the calls to **Find**, which, as mentioned above, can cost $O(n\,\alpha(n))$.

Linear-time algorithms for unification have been presented by Paterson and Wegman [1978] (cf. [Champeaux 1986]) and Martelli and Montanari [1982], to which we refer the reader for further study.

3. Equational unification

Like syntactic unification, equational unification is concerned with the problem of making terms equal by applying a suitable substitution. The only difference is that syntactic equality is replaced by equality modulo an equational theory E. At first sight, one might think that this is minor change, and that the notions and approaches from syntactic unification can easily be adapted to this new situation. It turns out, however, that equational unification requires some non-trivial adjustments of the basic notation. In particular, the notion of a most general unifier is no longer sufficient for the purpose of representing all unifiers since there may exist E-unifiable terms that do not have a most general E-unifier. In the first subsection, we introduce the basic notions as they are currently used in unification theory, and in the subsequent subsection, we point out some differences to the case of syntactic unification, and explain the reason for introducing the notions in this modified way. The third subsection introduces order-theoretic, logical, algebraic, and category-theoretic reformulations of some of these notions. We conclude the section with a short survey of results in unification theory. Some of these results will be treated in more detail in subsequent sections.

3.1. Basic notions

An *equational theory* is defined by a *set of identities* E, i.e., a subset of $\mathcal{T}(\mathcal{F},\mathcal{V}) \times \mathcal{T}(\mathcal{F},\mathcal{V})$ for a signature $\mathcal{F}$ and a (countably infinite) set of variables $\mathcal{V}$. It is the least congruence relation on the term algebra $\mathcal{T}(\mathcal{F},\mathcal{V})$ that is closed under substitution and contains E, and it will be denoted by $=_E$ (see [Dershowitz and Plaisted 2001, page 575] (Chapter 9 of this Handbook) for a more detailed definition of the relation $=_E$). Identities are written in the form $s \approx t$. If $s =_E t$, then we say that the term s is *equal modulo* E to the term t. For example, let f be a binary function symbol. The identity $C := \{f(x,y) \approx f(y,x)\}$ says that f is commutative, and the identity $A := \{f(f(x,y),z) \approx f(x,f(y,z))\}$ expresses associativity of f. We have $f(f(a,b),c) =_C f(c,f(b,a))$, and $f(a,f(x,b)) =_A f(f(a,x),b)$. In the following, we will often slightly abuse the notion of an equational theory by also calling a set of defining identities E an equational theory. For a given set of identities E, we denote by $\mathcal{S}ig(E)$ the set of all function symbols occurring in E.

3.1. DEFINITION. Let E be an equational theory and $\mathcal{F}$ a signature containing $\mathcal{S}ig(E)$. An *E-unification problem* over $\mathcal{F}$ is a finite set of equations

$$\Gamma = \{s_1 \overset{?}{\underset{E}{=}} t_1, \ldots, s_n \overset{?}{\underset{E}{=}} t_n\}$$

between $\mathcal{F}$-terms with variables in a (countably infinite) set of variables $\mathcal{V}$. An *E-unifier* of Γ is a substitution σ such that $s_1\sigma =_E t_1\sigma, \ldots, s_n\sigma =_E t_n\sigma$. The set of all E-unifiers of Γ is denoted by $\mathcal{U}_E(\Gamma)$, and Γ is *E-unifiable* iff $\mathcal{U}_E(\Gamma) \neq \emptyset$.

Obviously, syntactic unification is the special case of this definition where $E = \emptyset$. Any syntactic unifier of an E-unification problem Γ is also an E-unifier, but for $E \neq \emptyset$, the set $\mathcal{U}_E(\Gamma)$ may have additional elements. For example, if a and b are distinct constant symbols, then the C-unification problem $\{f(a,x) =^?_C f(b,y)\}$ has $\{x \mapsto b,\, y \mapsto a\}$ as C-unifier, whereas the terms $f(a,x)$ and $f(b,y)$ do not have a syntactic unifier. For the A-unification problem $\Gamma := \{f(a,x) =^?_A f(y,b)\}$, the set $\mathcal{U}_A(\Gamma)$ contains the syntactic unifier $\{x \mapsto b,\, y \mapsto a\}$ of $f(a,x)$ and $f(y,b)$, but also additional A-unifiers such as $\{x \mapsto f(z,b),\, y \mapsto f(a,z)\}$.

The instantiation quasi-ordering $\lessdot$ on substitutions is adapted to the case of equational unification as follows:

3.2. DEFINITION. Let E be an equational theory and $\mathcal{X}$ a set of variables. The substitution σ is *more general modulo E on $\mathcal{X}$* than the substitution θ iff there exists a substitution λ such that $x\theta =_E x\sigma\lambda$ for all $x \in \mathcal{X}$. In this case we write $\sigma \lessdot^{\mathcal{X}}_E \theta$ and say that θ is an E-instance of σ on $\mathcal{X}$.

It is easy to see that $\lessdot^{\mathcal{X}}_E$ is a quasi-ordering, i.e., a reflexive and transitive binary relation. The associated equivalence is denoted by $\doteq^{\mathcal{X}}_E$, i.e., $\sigma \doteq^{\mathcal{X}}_E \theta$ iff $\sigma \lessdot^{\mathcal{X}}_E \theta$ and $\theta \lessdot^{\mathcal{X}}_E \sigma$.

When comparing E-unifiers of a problem Γ, the set $\mathcal{X}$ is the set of all variables occurring in Γ. Unlike the case of syntactic unification, unifiable E-unification problems need not have a most general E-unifier. For example, the C-unification problem $\{f(x,y) =^?_C f(a,b)\}$ has the two C-unifiers $\sigma_1 := \{x \mapsto a,\, y \mapsto b\}$ and $\sigma_2 := \{x \mapsto b,\, y \mapsto a\}$. On $\mathcal{V}ar(\Gamma) = \{x,y\}$, any C-unifier of Γ is equal to either σ_1 or σ_2, and σ_1 and σ_2 are not comparable w.r.t the instantiation quasi-ordering $\lessdot^{\{x,y\}}_C$. Consequently, there cannot be a most general C-unifier of Γ. Thus, the rôle of the most general unifier must in general be taken on by a complete set of unifiers.

3.3. DEFINITION. Let Γ be an E-unification problem over $\mathcal{F}$ and let $\mathcal{X} := \mathcal{V}ar(\Gamma)$ be the set of all variables occurring in Γ. A *complete set of E-unifiers* of Γ is a set $\mathcal{C}$ of substitutions such that

1. $\mathcal{C} \subseteq \mathcal{U}_E(\Gamma)$, i.e., each element of $\mathcal{C}$ is an E-unifier of Γ,
2. for each $\theta \in \mathcal{U}_E(\Gamma)$ there exists $\sigma \in \mathcal{C}$ such that $\sigma \lessdot^{\mathcal{X}}_E \theta$.

The set $\mathcal{C}$ is a *minimal complete set of E-unifiers* of Γ iff it is a complete set that satisfies

3. two distinct elements of $\mathcal{C}$ are incomparable w.r.t. $\lessdot^{\mathcal{X}}_E$, i.e., for all $\sigma, \sigma' \in \mathcal{C}$, $\sigma \lessdot^{\mathcal{X}}_E \sigma'$ implies $\sigma = \sigma'$.

The substitution σ is a *most general E-unifier* of Γ iff $\{\sigma\}$ is a (minimal) complete set of E-unifiers of Γ.

If the unification problem Γ is not E-unifiable, then the empty set is a minimal complete set of E-unifiers of Γ. Depending on the equational theory E, minimal complete sets of E-unifiers need not always exist, and even if they do, they may be infinite (see below). It is, however, easy to show that they are unique up to instantiation equivalence $\stackrel{\bullet}{=}^{\mathcal{X}}_{E}$ (see subsection 3.3.1). This makes sure that the following definition of the unification type of an E-unification problem and of an equational theory E is unambiguous.

3.4. DEFINITION. Let E be an equational theory, and let Γ be an E-unification problem over $\mathcal{F}$. The problem Γ has type *unitary* (*finitary*, *infinitary*) iff it has a minimal complete set of E-unifiers of cardinality 1 (finite cardinality, infinite cardinality). If Γ does not have a minimal complete set of E-unifiers, then it is of type *zero*. We abbreviate type unitary by 1, type finitary by ω, type infinitary by ∞, and type zero by 0, and order these types as follows: $1 < \omega < \infty < 0$.

The *unification type of* E w.r.t. the signature $\mathcal{F}$ is the maximal type of an E-unification problem over $\mathcal{F}$.

According to this definition, an equational theory that is unitary is *not* finitary, and a theory of type zero is *not* infinitary. In the literature, these notion have sometimes been defined such that unitary implies finitary (i.e., unitary theories are a special case of finitary theories) and type zero implies infinitary. We prefer a stricter separation between the types. In order to express that a theory is unitary or finitary (in the sense of definition 3.4) we say that it is *at most finitary*. Analogously, to express that a theory is infinitary or of type zero we say that it is *at least infinitary*.

It should also be noted that the unification type of an equational theory depends not only on E, but also on the set of function symbols $\mathcal{F}$ that are allowed to occur in the unification problems (see subsection 3.2.2 for more details). We provide an example for each of the four types.

3.5. EXAMPLE *(unitary).* Since any unifiable unification problem has a most general syntactic unifier, the empty theory $\emptyset$ (which obviously defines syntactic equality) has unification type unitary w.r.t. any signature $\mathcal{F}$.

3.6. EXAMPLE *(finitary).* Above, we have seen that commutativity C is not unitary since the C-unification problem $\{f(x,y) =^?_C f(a,b)\}$ does not have a most general C-unifier. It is not hard to show that C is finitary w.r.t. any signature $\mathcal{F}$. In fact, the C-equivalence class $[t]_C := \{t' \mid t =_C t'\}$ of a given term t is easily shown to be finite. For a given C-unification problem $\Gamma = \{s_1 =^?_C t_1, \ldots, s_n =^?_C t_n\}$, we consider all possible syntactic unification problems of the form $\Gamma' = \{s'_1 =^? t'_1, \ldots, s'_n =^? t'_n\}$ where $s_i =_C s'_i$ and $t_i =_C t'_i$ for all $i, 1 \leq i \leq n$. There are only finitely many such sets Γ', and it can be shown that the collection of all the syntactic most general unifiers of these sets is a complete set of C-unifiers of Γ [Siekmann 1979]. In most cases, this set is not minimal, but obviously a minimal complete set can be obtained by eliminating redundant elements, i.e., elements that are C-instances of other elements of the set.

3.7. EXAMPLE *(infinitary).* Even though associativity A is similar to C in that A-equivalence classes are finite, the unification method outlined for C does not work for A. It is easy to see that the A-unification problem $\{f(a,x) =^?_A f(x,a)\}$ has an infinite minimal complete set of A-unifiers, namely $\{\sigma_n \mid n \geq 1\}$, where for each n the substitution $\sigma_n := \{x \mapsto f(a, f(a, \cdots, f(a,a)\cdots))\}$ replaces x by a term containing n occurrences of the constant a. Consequently, A cannot be unitary or finitary. Plotkin [1972] describes a procedure that generates a minimal complete set of A-unifiers of a given A-unification problem over an arbitrary set of function symbols $\mathcal{F}$, which shows that A is in fact infinitary and not of type zero.

3.8. EXAMPLE *(zero).* The first example of an equational theory of unification type zero was described by Fages and Huet [1983] and [1986]. In [Baader 1986] it is shown that the theory of idempotent semigroups, i.e., $AI := A \cup \{f(x,x) \approx x\}$ is of unification type zero since the AI-unification problem $\{f(x, f(y,x)) =^?_{AI} f(x, f(z,x))\}$ does not have a minimal complete set of AI-unifiers. This result was also shown by Schmidt-Schauß [1986*b*], but his example problem $\{f(z, f(a, f(x, f(a,z)))) =^?_{AI} f(z, f(a,z))\}$ contains an additional constant a.

For syntactic unification, a "unification algorithm" is an algorithm that computes a most general (syntactic) unifier of a given unification problem if it exists, and determines non-unifiability otherwise. For equational unification, this notion must be adapted. More precisely, we are interested in different types of algorithms, depending on what the equational theory allows and what is needed in applications.

An *E-unification algorithm* (w.r.t. $\mathcal{F}$) is an algorithm that computes a *finite* complete set of E-unifiers for all E-unification problems over $\mathcal{F}$. Ideally, the computed sets should also be minimal. There are, however, theories for which it is easier to compute a not necessarily minimal set (commutativity C is an example). We call an E-unification algorithm *minimal* iff it computes a finite minimal complete set of E-unifiers. As mentioned in example 3.6, an E-unification algorithm can always be turned into a minimal one by eliminating redundant unifiers, provided that the E-instantiation quasi-ordering is decidable.

In applications such as constraint-based approaches to automated deduction and rewriting (see [Bürckert 1991, Nieuwenhuis and Rubio 1994, Kirchner and Kirchner 1989] and [Nieuwenhuis and Rubio 2001], Chapter 7 of this Handbook), it is not necessary to compute complete sets of unifiers. Instead, it is sufficient to test unification problems for unifiability. An algorithm that is able to decide unifiability of E-unification problems (over $\mathcal{F}$) is called a *decision procedure* for E-unification (w.r.t. $\mathcal{F}$). Obviously, any E-unification algorithm yields a decision procedure for E-unification since a given E-unification problem Γ is unifiable iff the computed finite complete set is nonempty.

For theories that are not unitary or finitary, the notion of an E-unification algorithm, as introduced above, is not appropriate. A *(minimal) E-unification procedure* is a procedure that enumerates a possibly infinite (minimal) complete set of E-unifiers. The procedure by Plotkin [1972] mentioned in example 3.7 is a minimal A-unification procedure. An E-unification procedure need not yield a decision procedure for E-unification since it need not terminate even if the input problem does not have E-unifiers. This is, e.g., the case for Plotkin's procedure. A-

unification (more precisely, the question whether there exists an A-unifier for a given A-unification problem) is nevertheless decidable, but this is a lot harder to show [Makanin 1977] than designing a minimal A-unification procedure.

3.2. *New issues*

The notions introduced above deviate in several respects from the notions introduced for syntactic unification. In this subsection, we point out the reasons why this was necessary.

3.2.1. *The instantiation quasi-ordering*

For syntactic unification, the instantiation quasi-ordering $\stackrel{\bullet}{\leq}$ was defined by $\sigma \stackrel{\bullet}{\leq} \theta$ iff there exists λ such that $\theta = \sigma\lambda$. In the definition of the instantiation quasi-ordering for E-unification, syntactic equality is (quite naturally) replaced by equality modulo E. What may seem less clear is why we have restricted this equality (modulo E) to the variables occurring in the unification problem. Obviously, the ordering obtained this way is stronger than the one that requires equality on all variables (i.e., more substitutions are comparable). In applications in automated deduction, where substitutions generally have meaning only in the context of the expressions (i.e., unification problems) that produced them, it is admissible to use an ordering that compares alternate solutions only with respect to this small set of variables. It is also advisable, as this stronger ordering allows for smaller minimal complete sets. For example, the theory $ACU := AC \cup \{f(x,e) = x\}$ is known to be unitary w.r.t. $\mathcal{F} := \{f, e\}$. If the weaker instantiation quasi-ordering (i.e., the one comparing substitutions on all variables) were used, this would no longer be true [Baader 1991].

Another difference between the equational case and the syntactic case concerns the characterization of the instantiation equivalence $\stackrel{\bullet}{=}$. For $E = \emptyset$, two substitutions are instantiation equivalent iff they are equal up to composition with a variable renaming. It should be noted that this need no longer be the case for $E \neq \emptyset$, even if one replaces "equal up to composition with a variable renaming" by "equal modulo E up to composition with a variable renaming." For example, consider the equational theory $I := \{f(x,x) \approx x\}$, and the substitutions $\sigma := \{x \mapsto y\}$ and $\theta := \{x \mapsto f(y,z)\}$. Using the substitutions $\lambda_1 := \{y \mapsto f(y,z)\}$ and $\lambda_2 := \{y \mapsto y,\, z \mapsto y\}$, it is easy to show that $\sigma \stackrel{\bullet}{=}_E^{\{x\}} \theta$. However, a variable renaming cannot identify y and z, and thus $f(y,z)\rho \neq_I y$ for every such renaming ρ.

3.2.2. *The signature matters*

In the definitions of E-unification problems, unification type, etc., we have always explicitly stated which function symbols may occur in the unification problems. The reason is that the unification properties of an equational theory (like decidability, unification type, etc.) may depend on this set of function symbols. In most cases, however, a less fine-grained distinction is sufficient. Recall that $\mathcal{S}ig(E)$ denotes the set of all function symbols occurring in the equational theory E.

3.9. DEFINITION. Let E be an equational theory and Γ an E-unification problem over $\mathcal{F}$.

- Γ is an *elementary* E-unification problem iff $\mathcal{F} = \mathcal{S}ig(E)$.
- Γ is an E-unification problem *with constants* iff $\mathcal{F} \setminus \mathcal{S}ig(E)$ is a set of constant symbols.
- In a *general* E-unification problem, $\mathcal{F} \setminus \mathcal{S}ig(E)$ may contain arbitrary function symbols.

Following this distinction, we can introduce three different unification types for an equational theory. The *unification type of E w.r.t. elementary unification* is the maximal unification type of E w.r.t. all sets of function symbols $\mathcal{F}$ satisfying $\mathcal{F} = \mathcal{S}ig(E)$. Accordingly, the *unification type of E w.r.t. unification with constants* is the maximal unification type of E w.r.t. all sets of function symbols $\mathcal{F}$ such that $\mathcal{F} \setminus \mathcal{S}ig(E)$ is a set of constant symbols, and the *unification type of E w.r.t. general unification*[3] is the maximal unification type of E w.r.t. all signatures $\mathcal{F}$. Obviously, the same distinction can be made for decidability of E-unification, and for other interesting properties of E-unification problems. Constant (function) symbols that do not occur in E are called *free* constant (function) symbols w.r.t. E.

The theory ACU introduced above is an example of a theory that is unitary for elementary unification, but only finitary for unification with constants (see, e.g., [Herold and Siekmann 1987]). Bürckert [1989] has shown that there exists an equational theory for which elementary unification is decidable, but unification with constants is undecidable.

Applications of equational unification in automated deduction usually yield general unification problems. For example, in resolution-based theorem proving, the additional free function symbols are often generated by Skolemization.

From a strictly formal point of view, the definition of an E-unifier (see definition 3.1) is ambiguous since it does not specify over which signature the terms that are substituted for the variables may be built. By default, we have assumed that this set is the set $\mathcal{F}$, which contains all function symbols occurring in E or Γ. One might ask whether there would be a significant difference if we allowed the substitutions to introduce additional free function symbols. It is easy to show, however, that there is no such difference since any E-unifier of Γ that introduces additional free function symbols is an instance of an E-unifier that uses only symbols from $\mathcal{F}$: this more general unifier can, in principle, be obtained by replacing subterms starting with such additional function symbols by new variables, while taking care that $=_E$-equal subterms are replaced by the same variable. Thus, if we restrict the set of E-unifiers to substitutions over $\mathcal{F}$, we obtain a complete set of E-unifiers even w.r.t. substitutions over larger signatures. This justifies the (formally somewhat sloppy) definition of the set of E-unifiers given above.

[3]It should be noted that this use of the term "general unification" is distinct from the one in [Gallier and Snyder 1989, Snyder 1991], where it refers to methods that provide unification procedures for arbitrary equational theories (see section 4.1).

3.2.3. Single equations versus systems of equations
For syntactic unification, solving a system of term equations can be reduced to solving a single equation $s =^? t$. For this reason, syntactic unification is sometimes only considered for single equations. For equational unification, the same holds if one considers general unification. In fact, if $f \in \mathcal{F}$ is an n-ary function symbol not contained in $\mathcal{S}ig(E)$, then the E-unification problem $\{s_1 =^?_E t_1, \ldots, s_n =^?_E t_n\}$ over $\mathcal{F}$ has the same set of unifiers as $\{f(s_1, \ldots, s_n) =^?_E f(t_1, \ldots, t_n)\}$.

For elementary unification and for unification with constants, however, there may be significant differences. For example, there exists an equational theory E such that all elementary E-unification problems of cardinality 1 (i.e., single equations) have minimal complete sets of E-unifiers, but E is of type zero w.r.t. elementary unification since there exists an elementary E-unification problem of cardinality 2 that does not have a minimal complete set of E-unifiers [Bürckert, Herold and Schmidt-Schauß 1989]. Narendran and Otto [1990] give an example of a theory such that unifiability of elementary unification problems of cardinality 1 is decidable, but unifiability is undecidable for elementary unification problems of larger cardinality.

3.3. Reformulations

In this subsection, we consider reformulations of (some of) the notions introduced above from an order-theoretic, logical, algebraic, and category-theoretic point of view. This will shed a new light on the notions, and it allows us to utilize approaches and results from the respective areas in unification theory.

3.3.1. The order-theoretic point of view
Let E be an equational theory and Γ an E-unification problem with variables $\mathcal{X} := \mathcal{V}ar(\Gamma)$. We know that the relation $\lessdot^{\mathcal{X}}_E$ is a quasi-ordering on $\mathcal{U}_E(\Gamma)$ with associated equivalence relation $\doteq^{\mathcal{X}}_E$. Thus, $\lessdot^{\mathcal{X}}_E$ induces a partial ordering $\leq$ on the set $U := \{[\sigma] \mid \sigma \in \mathcal{U}_E(\Gamma)\}$ of all $\doteq^{\mathcal{X}}_E$-classes $[\sigma] := \{\tau \mid \sigma \doteq^{\mathcal{X}}_E \tau\}$:

$$[\sigma] \leq [\theta] \quad \text{iff} \quad \sigma \lessdot^{\mathcal{X}}_E \theta.$$

This allows us to investigate notions like complete and minimal complete sets of E-unifiers on the abstract order-theoretic level.

Thus, let $(U, \leq)$ be an arbitrary partially ordered set. A subset C of U is called *complete* iff for all $u \in U$ there exists $c \in C$ such that such that $c \leq u$. A complete set C is called *minimal* iff it is minimal with respect to set inclusion.

3.10. LEMMA. *The complete set $C \subseteq U$ is minimal iff for all $x, y \in C$, $x \leq y$ implies $x = y$.*

PROOF. If the elements x, y of the complete set C satisfy $x < y$, then $C \setminus \{y\}$ is also complete, which shows that C is not minimal. Conversely, if C_1, C_2 are complete sets such that $C_1 \subset C_2$, then there exists $y \in C_2 \setminus C_1$. Since C_1 is complete, there exists $x \in C_1$ such that $x \leq y$, and since $y \notin C_1$, we have $x \neq y$. $\square$

The following lemma describes the connection between minimal complete sets and minimal elements in partially ordered sets.

3.11. LEMMA. *Let M be the set of $\leq$-minimal elements of U.*
1. If $C \subseteq U$ is a minimal complete set, then $C = M$.
2. If M is complete, then it is minimal complete.

PROOF. The second statement is obvious, since different $\leq$-minimal elements of U cannot be comparable w.r.t. $\leq$. To show the first statement, let $C \subseteq U$ be a minimal complete set. Obviously, $M \subseteq C$ since any $\leq$-minimal element must be contained in a complete set. To see the other inclusion, assume that $y \in C$ is not minimal. Thus, there exists an element $y' \in U$ such that $y' < y$. Since C is complete, there exists $x \in C$ such that $x \leq y'$. Thus, we have $x, y \in C$ with $x < y$, which shows that C cannot be minimal. □

Figure 5 shows (the Hasse diagrams of) two partially ordered sets. The left one consists of an infinitely descending chain $x_1 > x_2 > x_3 > \cdots$. Consequently, the set of $\leq$-minimal elements is empty, and thus not complete. The right one also contains an infinitely descending chain (consisting of the elements $y_1, y_2, \ldots$), but the set of $\leq$-minimal elements (the elements $z_1, z_2, \ldots$) is obviously complete. If

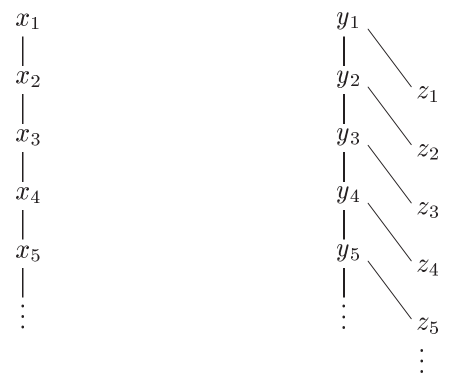


Figure 5: Two partially ordered sets.

$U = \{[\sigma] \mid \sigma \in \mathcal{U}_E(\Gamma)\}$ is the set of $\stackrel{\bullet}{=}{}^{\mathcal{X}}_E$-classes of E-unifiers of Γ, and $\leq$ is the partial ordering on U induced by $\stackrel{\bullet}{\leq}{}^{\mathcal{X}}_E$, then lemma 3.11 yields a nice characterization of all minimal complete sets of E-unifiers. If M is a subset of U, then a set of representatives of M is any subset of $\mathcal{U}_E(\Gamma)$ that contains for each class $m \in M$ exactly one representative, i.e., a unifier σ_m such that $[\sigma_m] = m$.

3.12. THEOREM. *Let M be the set of all $\leq$-minimal elements of U. If $\mathcal{C}$ is a minimal complete set of E-unifiers of Γ, then $M = \{[\sigma] \mid \sigma \in \mathcal{C}\}$. Conversely, if M is complete, then any set of representatives of M is a minimal complete set of E-unifiers of Γ.*

As an immediate consequence of this theorem we can deduce

3.13. Corollary. *Let M be the set of all $\leq$-minimal elements of U.*

1. *A minimal complete set of E-unifiers of Γ exists iff M is complete.*
2. *If a minimal complete set of E-unifiers of Γ exists, then it is unique up to the equivalence $\stackrel{\bullet}{=}^{\mathcal{X}}_E$.*

In [Baader 1989*a*], this order-theoretic point of view was used to derive different characterizations of unification type zero.

3.3.2. The algebraic and logical point of view

It is well known that the decision problems for elementary unification and for unification with constants correspond to natural classes of logical decision problems [Bockmayr 1992], and it turns out that the same is true for general unification.

Before stating these logical characterizations of E-unification, we recall some results from universal algebra about equationally defined classes (see, e.g., [Cohn 1965, Mal'cev 1971, Grätzer 1979] for more details). An equational theory E defines a *variety* (or equational class) $V(E)$, i.e., the class of all models of E. The theory E is called *non-trivial* if $V(E)$ contains algebras of cardinality > 1, and *trivial* otherwise. Obviously, E is trivial iff $x =_E y$ for distinct variables x, y. If E is a non-trivial equational theory, then $V(E)$ contains free algebras over any set of generators. In fact, let $\mathcal{F}_0 := \mathcal{S}ig(E)$, and let $\mathcal{X}$ be a set of variables of cardinality α. Then the quotient term algebra $\mathcal{T}(\mathcal{F}_0, \mathcal{X})/_{=_E}$ is a free algebra in $V(E)$. Its set of generators consists of the $=_E$-classes of the variables, and this set has cardinality α since E was assumed to be non-trivial. We call this algebra the *E-free algebra with generators $\mathcal{X}$*.[4] The fact that it is free in $V(E)$ means that any mapping from $\mathcal{X}$ into an algebra $\mathcal{A} \in V(E)$ can uniquely be extended to a homomorphism of $\mathcal{T}(\mathcal{F}_0, \mathcal{X})/_{=_E}$ into $\mathcal{A}$.

Now, we introduce the classes of formulae that correspond to equational unification problems. Let E be an equational theory, and $\mathcal{F}_0 := \mathcal{S}ig(E)$ be the set of function symbols occurring in E. An *atomic $\mathcal{F}_0$-formula* is an equation $s = t$. A *positive $\mathcal{F}_0$-matrix* is built from atomic $\mathcal{F}_0$-formulae using conjunction and disjunction. A *positive $\mathcal{F}_0$-sentence* is a quantifier-prefix followed by a positive $\mathcal{F}_0$-matrix that contains only variables introduced in the prefix. Without loss of generality we assume that the variables occurring in the prefix are all distinct. A *positive existential* $\mathcal{F}_0$-sentence is a positive $\mathcal{F}_0$-sentence whose prefix contains only existential quantifiers, and a positive AE $\mathcal{F}_0$-sentence has a prefix consisting of a block of universal quantifiers, followed by a block of existential quantifiers. The positive (positive existential, positive AE) fragment of the equational theory E consists of the set of all positive (positive existential, positive AE) $\mathcal{F}_0$-sentences that are valid in E, i.e., true in all models of E. Accordingly, for an $\mathcal{F}_0$-algebra $\mathcal{A}$, the positive

[4]Strictly speaking, the generators are the $=_E$-classes of the elements of $\mathcal{X}$, but since different variables belong to different classes, we slightly abuse the notation by identifying a variable $x \in \mathcal{X}$ with its $=_E$-class.

(positive existential, positive AE) theory of $\mathcal{A}$ is the set of all positive (positive existential, positive AE) $\mathcal{F}_0$-sentences that are true in $\mathcal{A}$.

3.14. THEOREM. *Let E be a non-trivial equational theory, $\mathcal{F}_0 := \mathcal{S}ig(E)$, and $\mathcal{V}$ a countably infinite set of variables.*

1. *Elementary E-unification is decidable iff the positive existential fragment of E is decidable iff the positive existential theory of $\mathcal{T}(\mathcal{F}_0, \mathcal{V})/_{=_E}$ is decidable.*
2. *E-unification with constants is decidable iff the positive AE fragment of E is decidable iff the positive AE theory of $\mathcal{T}(\mathcal{F}_0, \mathcal{V})/_{=_E}$ is decidable.*

PROOF. (1.1) Let $\Gamma := \{s_1 =^?_E t_1, \ldots, s_n =^?_E t_n\}$ be an elementary E-unification problem, and let $\mathcal{V}ar(\Gamma) = \{x_1, \ldots, x_k\}$. The terms $s_1, t_1, \ldots, s_n, t_n$ are $\mathcal{F}_0$-terms with variables in $\mathcal{V}ar(\Gamma)$, which implies that

$$\phi_\Gamma := \exists x_1. \cdots \exists x_k.\ s_1 = t_1 \wedge \ldots \wedge s_n = t_n$$

is a positive existential $\mathcal{F}_0$-sentence. We claim that Γ is E-unifiable iff ϕ_Γ holds in $\mathcal{T}(\mathcal{F}_0, \mathcal{V})/_{=_E}$ iff ϕ_Γ is valid in E.

Assume that σ is an E-unifier of Γ, i.e., $s_1\sigma =_E t_1\sigma, \ldots, s_n\sigma =_E t_n\sigma$. Without loss of generality we may assume that σ introduces only variables from $\mathcal{V}$. Thus, the substitution σ may also be considered as a valuation of the variables $\{x_1, \ldots, x_k\}$ by elements of $\mathcal{T}(\mathcal{F}_0, \mathcal{V})/_{=_E}$. Conversely, any such valuation can be seen as a substitution. This shows that Γ is E-unifiable iff ϕ_Γ holds in $\mathcal{T}(\mathcal{F}_0, \mathcal{V})/_{=_E}$.

If ϕ_Γ is valid in all models of E, it obviously holds in $\mathcal{T}(\mathcal{F}_0, \mathcal{V})/_{=_E} \in V(E)$. Conversely, assume that ϕ_Γ holds in $\mathcal{T}(\mathcal{F}_0, \mathcal{V})/_{=_E}$. If ϕ_Γ is not valid in E, then there exists an algebra $\mathcal{A} \in V(E)$ in which ϕ_Γ does not hold. By the Löwenheim-Skolem theorem, we may without loss of generality assume that $\mathcal{A}$ is countable. Thus, there exists a surjective homomorphism from $\mathcal{T}(\mathcal{F}_0, \mathcal{V})/_{=_E}$ onto $\mathcal{A}$ (extending an arbitrary surjection of $\mathcal{X}$ onto the carrier of $\mathcal{A}$). Since validity of positive sentences is invariant under surjective homomorphisms,[5] validity of ϕ_Γ in $\mathcal{T}(\mathcal{F}_0, \mathcal{V})/_{=_E} \in V(E)$ implies validity of ϕ_Γ in $\mathcal{A}$, which is a contradiction.

(1.2) Let $\phi = \exists x_1. \cdots \exists x_n.\ \psi$ be a positive existential $\mathcal{F}_0$-sentence. Without loss of generality we may assume that its matrix ψ is in disjunctive normal form, i.e., $\psi = \psi_1 \vee \ldots \vee \psi_n$ where the formulae ψ_i are conjunctions of equations. Since existential quantifier distribute over disjunction, ϕ is valid in E (in $\mathcal{T}(\mathcal{F}_0, \mathcal{V})/_{=_E}$) iff one of the formulae $\exists x_1. \cdots \exists x_n.\ \psi_i$ is valid in E (in $\mathcal{T}(\mathcal{F}_0, \mathcal{V})/_{=_E}$). Obviously, the formulae ψ_i can be translated into unification problems Γ_i, and as in part (1.1) of the proof we can show that Γ_i is unifiable iff $\exists x_1. \cdots \exists x_n.\ \psi_i$ is valid in E (in $\mathcal{T}(\mathcal{F}_0, \mathcal{V})/_{=_E}$).

(2) The second equivalence can be shown as in part (1.1) of the proof (since there we have only used the fact that ϕ_Γ is a positive $\mathcal{F}_0$-sentence).

To see the first equivalence, assume that ϕ is a positive AE sentence. Skolemizing the universally quantified variables[6] yields a positive existential $(\mathcal{F}_0 \cup \mathcal{F}_1)$-sentence

[5] See [Mal'cev 1973], pp. 143, 144 for a proof.

[6] We must Skolemize the universally quantified variables since we are interested in validity instead of satisfiability.

ϕ' such that $\mathcal{F}_1$ is a set of constants (not contained in $\mathcal{S}ig(E)$) and ϕ is valid in E iff ϕ' is valid in E. As in (1.2) of the proof, ϕ' can be translated into E-unification problems $\Gamma_{\psi'_i}$ such that ϕ' is valid in E iff one of these unification problems is unifiable. Obviously, the problems $\Gamma_{\psi'_i}$ are E-unification problem with constants since they contains the additional Skolem constants $\mathcal{F}_1$. Conversely, any E-unification problem with constants can be turned into a positive AE sentence by replacing its free constants by universally quantified variables. $\square$

The reduction described in part (1.2) of the proof is exponential in the worst case since the disjunctive normal form of the matrix ψ can be exponential in the size of ψ. For syntactic equality (i.e., $E = \emptyset$), it can be shown that the problem of deciding validity of positive existential sentences is NP-complete, whereas the corresponding unification problem is linear [Kozen 1981].

Before we state the analogous correspondence between general E-unification and the (full) positive fragment of E, we introduce another class of unification problems, which turns out to be equivalent to general E-unification.

3.15. Definition. An E-unification problem *with linear constant restrictions (lcr)* consists of an E-unification problem with constants, Γ, and a linear ordering $<$ on the variables and free constants occurring in Γ. A substitution σ is an *E-unifier* of $(\Gamma, <)$ iff it is an E-unifier of Γ that satisfies

$$x < c \text{ implies } \quad c \text{ does not occur in } x\sigma$$

for all variables x and free constants c in Γ.

For example, the (syntactic) unification problem $\{f(x) =^? f(c)\}$ has $\{x \mapsto c\}$ as most general unifier. Under the restriction $x < c$, this unifier is not admissible.

3.16. Theorem. *Let E be a non-trivial equational theory, $\mathcal{F}_0 := \mathcal{S}ig(E)$, and $\mathcal{V}$ a countably infinite set of variables. Then the following statements are equivalent:*

1. *The positive theory of E is decidable.*
2. *The positive theory of $\mathcal{T}(\mathcal{F}_0, \mathcal{V})/_{=_E}$ is decidable.*
3. *General E-unification is decidable.*
4. *E-unification with linear constant restrictions is decidable.*

Proof. We only give a sketch of the proof (see [Baader and Schulz 1996] for details).

In order to show (1) $\Leftrightarrow$ (2), it is sufficient to show that a positive $\mathcal{F}_0$-sentence ϕ is valid in E iff it is true in $\mathcal{T}(\mathcal{F}_0, \mathcal{V})/_{=_E}$. This can be shown as in part (1.1) of the proof of theorem 3.14.

A given positive sentence ϕ can be turned into a positive existential sentence ϕ' by Skolemization. As in part (2) of the proof of theorem 3.14, validity of ϕ' can be reduced to validity of several E-unification problems, which are general since they may contain Skolem functions of arbitrary arity. This shows (3) $\Rightarrow$ (1).

A given E-unification problem with linear constant restrictions $(\Gamma, <)$ can be transformed into a positive $\mathcal{F}_0$-sentence $\phi_\Gamma^<$ as follows: the matrix of $\phi_\Gamma^<$ is simply the conjunction of all equations in Γ. However, the constants in Γ are considered as variables in this matrix. The quantifier-prefix contains a universal quantifier for every free constant in Γ, and an existential quantifier for every variable in Γ. The order of the quantifiers is determined by the linear ordering $<$. It can be shown that $(\Gamma, <)$ is unifiable iff $\phi_\Gamma^<$ is valid in E. This proves (1) $\Rightarrow$ (4).

Finally, (4) $\Rightarrow$ (3) follows from the combination result in [Baader and Schulz 1996] (see section 6). $\square$

The following example, in which we assume $E = \{f(x) \approx f(x)\}$, illustrates the transformation of an E-unification problem with linear constant restrictions into a positive sentences, and of this positive sentence into a general E-unification problem (by Skolemization).

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

The problem $\{x =^?_E f(c)\}$ is not unifiable under the restriction $x < c$, since any unifier must replace x by $f(c)$, which contains the forbidden constant c. The corresponding positive sentence $\exists x. \forall y.\ x = f(y)$ is not valid since it says that f is a constant function, which is not true in all models of E. Finally, the general E-unification problem $\{x =^?_E f(h(x))\}$, which contains the Skolem function h, is not unifiable since one obtains an occurs check failure. Changing the linear ordering to $c < x$ leads to a unifiable unification problem with lcr, and the corresponding positive sentence is trivially valid.

3.3.3. The category-theoretic point of view

Let $\Gamma := \{s_i =^?_E t_i \mid i = 1, \ldots, n\}$ be an E-unification problem over $\mathcal{F}$, and $\mathcal{X} := \mathcal{V}ar(\Gamma)$ be the finite set of variables occurring in Γ. Since all our calculations are done modulo E, we may consider the terms s_i and t_i as elements of $\mathcal{T}(\mathcal{F}, \mathcal{X})/_{=_E}$, the E-free algebra with generators $\mathcal{X}$. For example, let $\mathcal{F}$ consist of a binary function symbol f, and let A axiomatize associativity of f, i.e., $A := \{f(x, f(y, z)) \approx f(f(x, y), z)\}$. The E-free algebra with generators $\mathcal{X}$ is the free semigroup $\mathcal{X}^+$, whose elements are the nonempty words over the alphabet $\mathcal{X}$. Instead of writing terms like $f(x, f(y, f(x, x)))$ in A-unification problems, we can omit the parentheses and all occurrences of the letter f, and simply write words like $xyxx$.

Also, since the instantiation quasi-ordering compares substitutions only on $\mathcal{X}$ and modulo E, each substitution can be seen as a homomorphism from $\mathcal{T}(\mathcal{F}, \mathcal{X})/_{=_E}$ into an E-free algebra $\mathcal{T}(\mathcal{F}, \mathcal{Y})/_{=_E}$, where $\mathcal{Y}$ is a suitable finite set (of variables or generators). For example, modulo A, the substitution $\sigma := \{x \mapsto f(x, f(y, f(x, x))),\ y \mapsto f(y, z)\}$ can be viewed as a homomorphism $\sigma\colon \{x, y\}^+ \to \{x, y, z\}^+$ that maps x to the word $xyxx$ and y to the word yz.

The E-unification problem Γ itself can be represented as a pair of homomorphisms between finitely generated E-free algebras. Indeed, let $\mathcal{I} := \{x_1, \ldots, x_n\}$ be a set of cardinality n. If we define σ, τ: $\mathcal{T}(\mathcal{F}, \mathcal{I})/_{=_E} \to \mathcal{T}(\mathcal{F}, \mathcal{X})/_{=_E}$ by

$$x_i\sigma := s_i \quad \text{and} \quad x_i\tau := t_i \quad (i = 1, \ldots, n),$$

then δ: $\mathcal{T}(\mathcal{F}, \mathcal{X})/_{=_E} \to \mathcal{T}(\mathcal{F}, \mathcal{Y})/_{=_E}$ is an E-unifier of Γ iff $x_i\sigma\delta = s_i\delta = t_i\delta = x_i\tau\delta$,[7] that is, iff $\sigma\delta = \tau\delta$. Consequently, any E-unification problem over $\mathcal{F}$ can be represented as a parallel pair of morphisms in the following category:[8]

3.17. DEFINITION. Let E be an equational theory and $\mathcal{F}$ be a signature such that $\mathcal{S}ig(E) \subseteq \mathcal{F}$. The category $C_{\mathcal{F}}(E)$ is defined as follows:

1. The objects of $C_{\mathcal{F}}(E)$ are the finitely generated E-free algebras $\mathcal{T}(\mathcal{F}, \mathcal{X})/_{=_E}$.
2. The morphisms of $C_{\mathcal{F}}(E)$ are the homomorphisms between these algebras. For a morphism δ: $\mathcal{T}(\mathcal{F}, \mathcal{X})/_{=_E} \to \mathcal{T}(\mathcal{F}, \mathcal{Y})/_{=_E}$, the algebra $\mathcal{T}(\mathcal{F}, \mathcal{X})/_{=_E}$ is called its domain, and the algebra $\mathcal{T}(\mathcal{F}, \mathcal{Y})/_{=_E}$ its codomain.
3. Composition $\sigma\delta$ of morphisms is the usual composition of mappings, which is only defined if the codomain of σ coincides with the domain of δ.

A *unification problem* in $C_{\mathcal{F}}(E)$ is a pair $\langle\sigma, \tau\rangle$ of morphisms σ, τ: $\mathcal{T}(\mathcal{F}, \mathcal{I})/_{=_E} \to \mathcal{T}(\mathcal{F}, \mathcal{X})/_{=_E}$ having the same domain and the same codomain. A *unifier* of $\langle\sigma, \tau\rangle$ in $C_{\mathcal{F}}(E)$ is a morphism δ with domain $\mathcal{T}(\mathcal{F}, \mathcal{X})/_{=_E}$ such that $\sigma\delta = \tau\delta$.

The instantiation quasi-order, and the notions complete and minimal complete set of unifiers as well as most general unifier can be adapted in an obvious way to this view of E-unification as a problem in $C_{\mathcal{F}}(E)$. For example, the morphism δ is a *most general* unifier of $\langle\sigma, \tau\rangle$ iff it is a unifier of $\langle\sigma, \tau\rangle$ such that, for all unifiers θ of $\langle\sigma, \tau\rangle$, there exists a morphism λ satisfying $\theta = \delta\lambda$.

Readers familiar with basic notions from category theory may have noticed that this definition of a most general unifier of $\langle\sigma, \tau\rangle$ strongly resembles the definition of a *coequalizer* of a parallel pair of morphisms (i.e., a pair with the same domain and the same codomain). The only difference is that for a most general unifier of $\langle\sigma, \tau\rangle$ to be a coequalizer, the morphism λ such that $\theta = \delta\lambda$ must always be *unique*.

It is easy to see that a most general unifier of $\langle\sigma, \tau\rangle$ need not be a coequalizer of this parallel pair. For example, the most general (syntactic) unifier $\delta := \{y \mapsto x\}$ of the equation $f(x, y) =^? f(y, x)$ can be viewed as a morphism $\delta_{\mathcal{Y}}$: $\mathcal{T}(\{f\}, \{x, y\}) \to \mathcal{T}(\{f\}, \mathcal{Y})$ for any finite set of variables $\mathcal{Y}$ containing x. All these morphisms are most general unifiers of the parallel pair corresponding to the unification problem $f(x, y) =^? f(y, x)$, but only $\delta_{\{x\}}$ is a coequalizer. More generally, a most general unifier in $C_{\mathcal{F}}(\emptyset)$ need not be a coequalizer, but it can always be transformed into one by appropriately restricting the set of generators in its codomain.

For nonempty theories, such a transformation need not be possible, however. As shown in [Baader 1991], there exists an equational theory, namely the theory ACU

[7] Since terms are now viewed as elements of E-free algebras (i.e., $=_E$-equivalence classes), we may write equality ($=$) in place of equality modulo E ($=_E$).

[8] See [Pierce 1991] for basic definitions and results of category theory.

that axiomatizes an associative-commutative binary symbol f with a unit e, such that all solvable unification problems in $C_{\{f,e\}}(ACU)$ have a most general unifier, but not all solvable unification problems in this category have a coequalizer. In the applications of E-unification in automated deduction, the additional uniqueness requirement in the definition of a coequalizer is not relevant. Thus, one should stick with the definition of a most general unifier as introduced above, and not replace it by the one of a coequalizer.

As such, the simple observation that E-unification has a category-theoretic interpretation does not solve any problems: it just transforms them into a different representation. This new representation is only of interest if techniques and results from category theory can be used to solve new and interesting problems in unification theory. Rydeheard and Burstall [1985] use the category-theoretic representation of syntactic unification to derive a unification algorithm based on colimit constructions in $C_{\mathcal{F}}(\emptyset)$. In [Baader 1989*b*], results from category theory on so-called semi-additive categories are used to obtain results on unification modulo so-called commutative theories (see subsection 5.2 below).

Even though the construction of the category $C_{\mathcal{F}}(E)$ is quite natural, there are also other ways of representing unification problems in category-theoretic terms. Whereas Goguen [1989] just introduces the dual category of $C_{\mathcal{F}}(E)$ (where morphisms are inverse homomorphisms), Ghilardi [1997] takes a quite different approach: he considers the category of all algebras in $V(E)$ (not only the finitely generated free ones), and represents unification problems as finitely presented algebras in this category. In this setting, the proof that unification in Boolean algebras and in primal algebras is unitary [Nipkow 1990] becomes trivial.

3.4. Survey of results for specific theories

Research in unification theory has produced results on unification properties of a great variety of equational theories. In this section, we will briefly review some of these results, with an emphasis on the more recent ones that are not yet covered by previous surveys of the area [Siekmann 1989, Jouannaud and Kirchner 1991, Kapur and Narendran 1992*a*, Baader and Siekmann 1994]. For each theory, we are interested in the decision problem and its complexity as well as its unification type and the existence of unification algorithms and procedures. Depending on which kind of unification problems (elementary, with constants, or general) is considered, there may exist different results for a given theory.

Associativity
The theory $A_f := \{f(f(x,y),z) \approx f(x,f(y,z))\}$ axiomatizes associativity of the binary function symbol f.

Decision problem: This problem, which is very hard and had been open for a long time, was finally solved by Makanin [1977], who proves decidability of A_f-unification with constants (see also [Pécuchet 1981, Jaffar 1990, Abdulrab and Pécuchet 1989, Schulz 1993]). Using general combination techniques and an extension of Makanin's algorithm [Schulz 1992], decidability of general A_f-unification was shown in [Baader and Schulz 1992, Baader and Schulz 1996]. The decision problem for A_f-unification is NP-hard [Benanav, Kapur and Narendran 1985]. The known upper bound is still higher, even though there has recently been considerable progress in lowering the bound: the 3-NEXPTIME result by Koscielski and Pacholski [1990] was first improved to EXPSPACE by Gutiérrez [1998], then to NEXPTIME by Plandowski [1999*a*], and finally to PSPACE [Plandowski 1999*b*]. Interestingly, the last two results no longer need Makanin's algorithm, i.e., they yield a new decision procedure that is independent of Makanin's result.

Unification type: infinitary for all three kinds of unification problems [Plotkin 1972] (see also example 3.7).

Unification procedures: Plotkin [1972] describes a minimal unification procedure for general A_f-unification, which can even deal with several associative function symbols. In general, this procedure does not yield a decision procedure since it need not terminate even for non-solvable problems or problems having a finite minimal complete set of A_f-unifiers. For certain restricted types of A_f-unification problems, modifications of Plotkin's procedure can be turned into decision procedures that are simpler than Makanin's general procedure [Auffray and Enjalbert 1992, Schmidt 1998].

Commutativity

The theory $C_f := \{f(x,y) \approx f(y,x)\}$, which axiomatizes commutativity of the binary function symbol f, has already been considered in example 3.6.

Decision problem: NP-complete for C_f-unification with constants and general C_f-unification. The hardness result for unification with constants is mentioned in [Garey and Johnson 1979], where it is attributed to Sethi (private communication, 1977). A simple NP-hardness proof due to Narendran (private communication, 1993) is sketched in [Baader and Siekmann 1994]. It is easy to see that this proof can also be used to show NP-hardness of elementary C_f-unification (private communication by Narendran, 1997).[9] NP-decision procedures for general C_f-unification can easily be obtained from the simple unification algorithm sketched in example 3.6: instead of testing all possible sets Γ', the non-deterministic decision procedure first guesses such a set Γ', and then tests whether this set has a syntactic unifier.

Unification type: finitary for all three kinds of unification problems [Siekmann 1979].

Unification algorithms: In addition to Siekmann's simple (non-minimal) unification algorithm for general C_f-unification [Siekmann 1979], various other methods have been proposed [Fages 1983, Kirchner 1985, Herold 1987]. However, none of them directly produces a *minimal* complete set of C_f-unifiers.

[9] In this proof, simply replace the constants a, b by the terms $t_a := f(x, f(x,x)$ and $t_b := f(x,x)$ and add for each propositional variable q an equation $f(x_q, y_q) =^?_{C_f} f(t_a, t_b)$, which makes sure that x_q is instantiated either by t_a or by t_b.

Distributivity

The theories $D^l_{f,g} := \{f(x, g(y,z)) \approx g(f(x,y), f(x,z))\}$ and $D^r_{f,g} := \{f(g(y,z), x) \approx g(f(y,x), f(z,x))\}$ axiomatize left-distributivity and right-distributivity of f over g, and their union $D_{f,g} := D^l_{f,g} \cup D^r_{f,g}$ axiomatizes (both-sided) distributivity of f over g. In addition, we consider combinations of these theories with A_g and $U_f := \{f(x,e) \approx x,\ f(e,x) \approx x\}$.

Decision problem: $D^l_{f,g}$-unification (and, by symmetry, $D^r_{f,g}$-unification) with constants is decidable in polynomial time [Tidén and Arnborg 1987].

If one adds a unit for f, i.e., considers $D^l_{f,g} \cup U_f$ (or $D^r_{f,g} \cup U_f$), then the problem becomes much harder since A_f-unification can be reduced to $(D^l_{f,g} \cup U_f)$-unification. Decidability of $(D^l_{f,g} \cup U_f)$-unification with constants was shown in [Schmidt-Schauß 1996*b*]. Since this decision procedure can be extended to cope with linear constant restrictions, general results on the combination of decision procedures [Baader and Schulz 1996] imply that general $(D^l_{f,g} \cup U_f)$-unification is decidable.

For unification modulo both-sided distributivity, the decision problem was open for quite a while. After some preliminary decidability results for restricted classes of $D_{f,g}$-unification problems [Contejean 1993, Schmidt-Schauß 1992], decidability of $D_{f,g}$-unification with constants was finally shown by Schmidt-Schauß [1996*a*]. His non-deterministic algorithm reduces solvability of $D_{f,g}$-unification problems with constants to A_f-unification with constants and ACU-unification with linear constant restrictions. Thus, the algorithm is of quite high complexity, compared to the best known lower bound, which is NP-hard [Tidén and Arnborg 1987].

Undecidability of $(D_{f,g} \cup A_g)$-unification with constants was proved in [Szabó 1982, Siekmann and Szabó 1989]. This negative result has been strengthened in [Tidén and Arnborg 1987]: every equational theory that lies above $(D_{f,g} \cup A_g)$ or $(D^l_{f,g} \cup U_f \cup A_g)$ and is consistent with Peano arithmetic (where f stands for multiplication, g for addition, and e for 1) has an undecidable unification problem. Decidability of $(D_{f,g} \cup U_f)$-unification is still an open problem.

Unification type: infinitary for $D_{f,g}$-unification problems with constants and general $D_{f,g}$-unification problems. Szabó [1982] gives an example of a $D_{f,g}$-unification problem with constants whose minimal complete set of unifiers is infinite. The existence of minimal complete sets of $D_{f,g}$-unifiers (for all three kinds of unification problems) is a consequence of the fact that the $=_{D_{f,g}}$-class of a given term is always finite [Szabó 1982], which implies that the instantiation quasi-ordering $\lessdot^{\mathcal{X}}_{D_{f,g}}$ is Noetherian [Szabó 1982, Bürckert et al. 1989].

$D^l_{f,g}$-unification (and, by symmetry, $D^r_{f,g}$-unification) with constants is unitary, and an mgu can be computed in polynomial time [Tidén and Arnborg 1987].

Associativity-commutativity

The theories $AC_f := A_f \cup C_f$ and $ACU_f := AC_f \cup U_f$ will be considered in more detail in subsection 5.1. Examples of operations satisfying theses identities are addition and multiplication of (rational, real, etc.) numbers.

Decision problem: NP-complete for unification problems with constants and general unification problems both for AC_f and ACU_f [Kapur and Narendran 1992*a*]. Elementary ACU_f-unification problems always have a trivial solution, and solvability of elementary AC_f-unification problems is decidable in polynomial time using linear programming [Domenjoud 1991].

Unification type: ACU_f is unitary for elementary and finitary for the two other kinds of unification problems, and AC_f is finitary for all three kinds of unification problems [Livesey and Siekmann 1975, Stickel 1981, Fages 1987]. The number of unifiers in a minimal complete set of AC_f-unifiers may be doubly-exponential in the size of a given elementary AC_f-unification problem [Kapur and Narendran 1992*b*].

Unification algorithms: Because unification modulo associativity-commutativity has many applications in automated deduction, a great variety of unification algorithms has been developed for AC_f and ACU_f [Stickel 1975, Livesey and Siekmann 1975, Kirchner 1985, Fortenbacher 1985, Büttner 1986*a*, Herold 1987, Herold and Siekmann 1987, Lincoln and Christian 1989, Boudet, Contejean and Devie 1990] (see also subsection 5.1).

Associativity-commutativity-idempotency

We consider the theories $ACI_f := AC_f \cup \{f(x,x) \approx x\}$, its extension by a unit e, $ACUI_f := ACI_f \cup U_f$, and by a zero n, $ACUZI_f := ACUI \cup \{f(x,n) \approx n\}$. Examples of operations satisfying theses identities are union and intersection of sets. The theory $ACUI_f$ will be considered in more detail in subsection 5.1.

Decision problem: For all three theories, the decision problem is polynomial for elementary unification and for unification with constants, and NP-complete for general unification [Kapur and Narendran 1992*a*, Narendran 1996*b*]. Like syntactic unification, ACI_f- and $ACUI_f$-unification with constants are not only in P, but even P-complete [Hermann and Kolaitis 1997].

Unification type: $ACUI_f$ is unitary for elementary and finitary for the two other kinds of unification problems, and ACI_f is finitary for all three kinds of unification problems [Livesey and Siekmann 1975, Büttner 1986*b*, Baader and Büttner 1988, Kapur and Narendran 1992*b*]. As with AC_f, the number of ACI_f-unifiers in a minimal complete set may be doubly-exponential in the size of a given elementary ACI_f-unification problem [Kapur and Narendran 1992*b*]. Hermann and Kolaitis show that computing the cardinality of a minimal complete set of unifiers for given ACI_f- or $ACUI_f$-unification unification problems is $\#P$-hard, which implies that this function cannot be computed in polynomial time, unless P = NP [Hermann and Kolaitis 1997].

Unification algorithms: Baader and Büttner [1988] describe an algorithm for $ACUI_f$-unification problems with constants consisting of a single equation, and Kapur and Narendran [1992*b*] sketch an algorithm for general ACI_f-unification.

Abelian groups

The theory of Abelian groups is defined by the identities $AG_f := ACU_f \cup \{f(i(x), x) \approx e\}$.

Decision problem: trivial for elementary unification, polynomial for unification with constants [Baader and Siekmann 1994], and NP-complete for general unification [Schulz 1997].

Unification type: unitary for elementary unification and for unification with constants [Lankford, Butler and Brady 1984], and finitary for general unification [Schmidt-Schauß 1989*b*, Boudet, Jouannaud and Schmidt-Schauß 1989]. Computing the cardinality of a minimal complete set of unifiers for a given general AG_f-unification is again $\#P$-hard [Hermann and Kolaitis 1996].

Unification algorithms: Lankford et al. [1984] describe an algorithm for AG_f-unification with constants, and Schmidt-Schauß [1989*b*] shows that this algorithm can be combined with an algorithm for syntactic unification into an algorithm for general AG_f-unification.

Commutative and Boolean rings

Let CRU denote the well-known axioms for commutative rings with a (multiplicative) unit, and BR the theory of Boolean rings.

Decision problem: As sketched in [Baader and Siekmann 1994], undecidability of elementary CRU-unification is an easy consequence of the fact that Hilbert's 10th problem is undecidable [Matiyasevich 1971, Davis 1973].
For the theory BR, the decision problem is NP-complete for elementary unification, Π_2^p-complete for unification with constants, and PSPACE-complete for general unification [Baader 1998].

Unification type: The unification type of CRU is at least infinitary, even for elementary unification [Burris and Lawrence 1990].[10].
BR is unitary for elementary unification and for unification with constants [Büttner and Simonis 1987, Martin and Nipkow 1989*b*, Martin and Nipkow 1989*a*], and finitary for general unification [Schmidt-Schauß 1989*b*]. As with the theory of Abelian groups, the problem of computing the cardinality of a minimal complete set of unifiers is $\#P$-hard for general BR-unification [Hermann and Kolaitis 1996].

Unification algorithms: Algorithms that compute most general unifiers for elementary BR-unification and BR-unification with constants are described in [Büttner and Simonis 1987, Martin and Nipkow 1989*b*, Martin and Nipkow 1989*a*]. General combination methods can be used to obtain algorithms for general BR-unification [Schmidt-Schauß 1989*b*, Boudet et al. 1989].

Endomorphisms

The theory $End_{h,g} := \{h(g(x, y)) \approx g(h(x), h(y))\}$ states that the unary function symbol h behaves like an endomorphism for the binary function symbol g, and

[10]The closely related theory of commutative *semi*rings is known to be of unification type zero w.r.t. elementary unification [Franzen 1992]

$End_{h,e} := \{h(e) \approx e\}$ states that h behaves like an endomorphism for the constant symbol e. We consider these two theories in combination with some of the theories introduced above:

Decision problem: Solvability of $End_{h,g}$-unification problems with constants is decidable [Vogel 1978].
For the theories $End_{h,g} \cup AC_g$ and $End_{h,g} \cup End_{h,e} \cup ACU_g$, solvability of unification problems with constants is undecidable [Narendran 1996*a*].
In contrast, solvability of unification problems with constants is decidable for the theory $End_{h,g} \cup End_{h,e} \cup ACUI_g$. In [Baader and Narendran 1998] it shown that this problem is EXPTIME-complete.
A similar result holds for $End_{h,g} \cup ACUI_g$: for this theory, the decision problem is known to be co-NP-hard and in EXPTIME [Guo, Narendran and Shukla 1998].
Finally, for $End_{h,g} \cup End_{h,e} \cup AG_g$, decidability of unification with constants was shown in [Baader 1993]. Since this decidability result can be extended to unification with linear constant restrictions, general combination results yield decidability for general unification modulo this theory [Baader and Nutt 1996].

Unification type: The theory $End_{h,g}$ is unitary for unification with constants [Vogel 1978].
$End_{h,g} \cup End_{h,e} \cup ACU_g$ and $End_{h,g} \cup End_{h,e} \cup ACUI_g$ are of type zero, even for elementary unification [Baader 1993, Baader 1989*b*].
$End_{h,g} \cup End_{h,e} \cup AG_g$ is unitary for elementary unification and for unification with constants [Nutt 1990, Baader 1993], and finitary for general unification [Baader and Nutt 1996].

In addition to investigating unification properties of specific equational theories of interest, unification theory also tries to develop more general methods, and thus to obtain results for whole classes of equational theories. Since unification modulo equational theories is in general undecidable (as illustrated by some of the examples above), and also unification properties such as the unification type of a given theory are in general undecidable [Nutt 1991], approaches that apply to all equational theories are likely to yield very weak results. For example, the general E-unification procedure introduced in section 4.1, which can be used to enumerate a complete set of E-unifiers, is very inefficient, and usually does not yield a decision procedure or a (minimal) E-unification algorithm even for unitary or finitary theories whose unification problem is decidable. In order to obtain more useful results, one can try to develop methods that work for appropriately restricted classes of theories. There are basically two different ways of introducing appropriate restrictions on equational theories. *Syntactic approaches* impose restrictions on the syntactic form of the identities defining the equational theories. The unification methods produced by these approaches are usually also of a quite syntactic nature: as with the rule-based approach to syntactic unification, they transform the given unification problem into a problem in solved form (section 4). In contrast, *semantic approaches* depend on properties of the (free) algebras defined by the equational theory. Unification problems are translated into equations over certain algebraic structures, which (in some cases) can be solved using known results from mathematics (section 5).

4. Syntactic methods for E-unification

In this section we discuss two syntactic approaches to generating complete sets of E-unifiers, using inference systems extending the set $\mathcal{U}$ presented in section 2.2.3. We first consider the general problem (E-unification in arbitrary theories) and show how it can be solved by adding a single rule to introduce identities into the transformation process; this simple method is proved to be complete and some restrictions which preserve completeness are discussed. We then present the most significant special case of the general problem, when the equational theory can be presented by a convergent set of rewrite rules. This method, called *narrowing*, has been thoroughly investigated, and we will present the major results in the framework of transformation rules.

4.1. E-unification in arbitrary theories

In this section, we present a rule for introducing identities into inference steps in $\mathcal{U}$ in such a way that a complete set of E-unifiers for an arbitrary set E of equations may be generated. By specializing various aspects of the resultant calculus (and its completeness proof), we will obtain more practical methods for the special case of convergent sets of rewrite rules. The results of this section are based on [Gallier and Snyder 1989, Snyder 1991].

In this section we assume that the reader is familiar with the basic concepts of rewriting (especially equational proofs, reduction orderings, ground convergence, and critical pairs) discussed in [Dershowitz and Plaisted 2001] (Chapter 9 of this Handbook). By *rewrite proof* we refer to a sequence of rewrite steps between two terms of the form

$$s \xrightarrow{*} u \xleftarrow{*} t$$

where u is in normal form. We will use $e[u]$ in the following to represent a equation (or identity) with a distinguished occurrence of a subterm u in one of its terms; in such a context $e[r]$ will denote the result of replacing this subterm with the term r. We will use systems $P; S$, representing unification problems and sets of equations in solved form, as before.

4.1. DEFINITION. For any equational theory E, a substitution θ is an *E-solution* (or simply a *solution* when E is understood) of a system $P; S$ if it is an E-unifier of every equation in P, and a unifier of every equation in S.

4.1.1. The calculus $\mathcal{G}$

The set $\mathcal{G}$ of inference rules consists of the rules Trivial, Decomposition, Orientation, and Variable Elimination from $\mathcal{U}$, plus the following rule for introducing identities:

Lazy Paramodulation (LP):

$$\{e[u]\} \cup P; S \implies_{\mathsf{lp}} \{l \stackrel{?}{=} u, e[r]\} \cup P; S$$

for a fresh variant[11] of the identity $l \approx r$ from $E \cup E^{-1}$, and where (i) u is *not* a variable, and (ii) if l is not a variable, then the top symbols of l and u are identical, and no other inference rule may be applied to the equation $l =^? u$ before it is subjected to a Decomposition step.

Computation in $\mathcal{G}$ proceeds as in $\mathcal{U}$, starting with an initial system of the form $\{s =^? t\}; \emptyset$ and applying inference rules in an attempt to find some terminal system $\emptyset; S$ representing an E-unifier σ_S of s and t. Clearly, by the general characteristics of E-unification discussed above, such a process can not share the nice properties of $\mathcal{U}$ which we discussed in section 2.2.4. However, it is possible to say quite a lot about how to restrict the application of rules, as we shall see.

4.1.2. Completeness of $\mathcal{G}$

It can be shown easily that the calculus $\mathcal{G}$ is *sound* in the sense that a solution it produces is always an E-unifier; however this proof does not give much insight into the properties of $\mathcal{G}$ and we refer the interested reader to [Gallier and Snyder 1989]. It is more interesting to consider the issue of completeness, which is considerably more complex than in the standard case. What we want to show is that if we consider the (finitely-branching but infinite) search tree of every possible transformation sequence starting from $\{s =^? t\}; \emptyset$, then the leaves form a complete set of E-unifiers for s and t. However, it is simpler to state and prove this in the following "non-deterministic" form.

4.2. Theorem. *Let E be a non-trivial equational theory and P be a set of unification problems. If θ is an E-solution of $P; \emptyset$, then there exists a sequence*

$$P; \emptyset \stackrel{*}{\Longrightarrow} \emptyset; S$$

(with S in solved form) in the calculus $\mathcal{G}$ such that $\sigma_S \lessdot_E^{\mathcal{X}} \theta$, where $\mathcal{X} = \mathcal{V}ars(P)$.

There are three main stages to the proof. First we will prove the result given certain strong restrictions on the equational theory E. Then we construct a kind of "abstract completion" of E which has the requisite restrictions; finally, we show that any transformation sequence using this abstract completion can be converted into one using simply E.

The major difficulty in proving completeness of equational inference systems is generally in dealing with the restriction that equational steps not take place at variable positions (hence, "u is not a variable" in LP). The solution, due to Peterson [1983], is to work with a restricted form of substitution in the proof.

4.3. Definition. Given a rewrite system R, a substitution θ is *R-reduced* (or just *reduced* if R is unimportant) if for every $x \in \mathcal{D}om(\theta)$, $x\theta$ is in R-normal form.

[11] By a *fresh variant* we refer to an expression that has been renamed with fresh variables that do not occur anywhere else in the previous computation. Whenever we mention a rewrite rule or identity used in an inference step, we will assume that it has been so renamed.

Note that it is always possible for any θ and terminating set of rules R to find an R-equivalent reduced substitution θ'. This allows us to assume, when "lifting" rewrite steps at the ground level to inference steps, that the position is a non-variable.

Another essential ingredient in our proof is the notion of an "oriented ground instance" of an identity.

4.4. DEFINITION. Let E be a non-trivial equational theory and $\succ$ be a reduction ordering total on ground terms. The set of *ground instances* of E is

$$Gr(E) \ := \ \{ \, l\rho \approx r\rho \mid l\rho \text{ and } r\rho \text{ are ground and } l \approx r \in E \cup E^{-1} \, \}.$$

The set of *oriented ground instances* of E is

$$Gr^{\succ}(E) \ := \ \{ \, l\rho \longrightarrow r\rho \mid l\rho \approx r\rho \in Gr(E) \text{ and } l\rho \succ r\rho \, \}.$$

A member $l\rho \longrightarrow r\rho$ of such a set is called *reduced* if ρ is reduced with respect to the entire set.[12] For any E, the set of reduced oriented ground instances is denoted R_E.

An important fact about $Gr(E)$ is the following.

4.5. PROPOSITION. *For any two ground terms s and t, there exists an equational proof $s \overset{*}{\longleftrightarrow}_E t$ iff there exists a proof $s \overset{*}{\longleftrightarrow}_{Gr(E)} t$*

This is easily proved by showing that equational steps are closed under instantiation, and hence we can instantiate any "unbound variables" by ground terms so that only ground instances of identities from E are used.

Another kind of restriction on proofs, which will be essential in proving the "no inferences into variable positions" restriction in our completeness result, is the subject of the next definition and lemma.

4.6. DEFINITION. Let $u\theta$ be an instance of u, and R a set of rewrite rules. A rewrite step $u\theta \longrightarrow_R u'$ is *based on u* iff the redex is at a non-variable position in u (equivalently, is not wholly contained within a term introduced by θ). A rewrite sequence $s\theta \overset{*}{\longrightarrow}_R t$ is *based on s* (or simply *basic*) iff either $s\theta = t$ (reflexive case) or it starts with a rewrite step based on s, e.g.,

$$s\theta \ \longrightarrow_R (s\theta)[r\rho] = s[r]\theta\rho \overset{*}{\longrightarrow}_R t$$

and the remainder is based on $s[r]$. A rewrite proof $s\theta \overset{*}{\longrightarrow} \overset{*}{\longleftarrow} t\theta$ is *basic* if the left side is based on s and the right side is based on t.

Intuitively, this means that no rewrite step can take place at a term introduced by any substitution.

The relationship between reduced substitutions, reduced oriented ground instances, ground convergence, and basic rewrite sequences is now explored.

[12] This notion is well-defined, as it could more formally be defined by induction on a suitable ordering of rules, using the fact that l can not be a variable when E is non-trivial.

4.7. Lemma. *Let E be a non-trivial equational theory such that $Gr^{\succ}(E)$ is ground convergent, and $s\theta$ be a ground term such that θ is R_E-reduced. Then for any rewrite sequence $s\theta \xrightarrow{*} t$ using rules from $Gr^{\succ}(E)$ to reduce $s\theta$ to its normal form t, there exists a basic rewrite sequence $s\theta \xrightarrow{*} t$ using rules only from R_E.*

Proof. Since $Gr^{\succ}(E)$ is ground canonical, we may choose any fair strategy for reduction; in particular, we may specify that at each step, among all the possible rules that could be used for reduction, we choose one that is minimal in the lexicographic extension of $\succ$ to pairs of terms. But then for any $l\rho \longrightarrow r\rho$ used in the sequence, ρ must be reduced, or else the rule would not be minimal. Thus, there exists a rewrite sequence from $s\theta$ to t using rules only from R_E; clearly, since all substitutions involved are reduced, this is also a basic sequence. □

For our purposes we may summarize these results as follows.

4.8. Corollary. *Let E be an equational theory such that $Gr^{\succ}(E)$ is ground convergent. For any ground terms $s\theta$ and $t\theta$, where θ is reduced with respect to $Gr^{\succ}(E)$, the following are equivalent:*

1. *$s\theta$ and $t\theta$ are E-equivalent.*
2. *There exists a basic rewrite proof for $s\theta$ and $t\theta$ using rules from $Gr^{\succ}(E)$.*

We now prove our completeness result in the special case we have been discussing.

4.9. Lemma. *Let E be a non-trivial equational theory such that $Gr^{\succ}(E)$ is ground convergent, and P be a set of unification problems. If θ is a $Gr^{\succ}(E)$-reduced solution of $P;\emptyset$, then there exists a sequence*

$$P;\emptyset \stackrel{*}{\Longrightarrow} \emptyset;S$$

(with S in solved form) in the calculus $\mathcal{G}$ such that $\sigma_S \trianglelefteq^{\mathcal{X}} \theta$ for $\mathcal{X} = \mathcal{V}ars(P)$.

Proof. We proceed by induction, using the following measure. The complexity of a system $P;S$ and its solution θ is a four-tuple $\langle m, n_1, n_2, n_3\rangle$, where

- m = The total number of rewrite steps in all the minimal-length basic rewrite proofs for equations in $P\theta$;
- n_1 = The number of distinct variables occurring in equations $u =^? v \in P$ such that $u\theta = v\theta$ and $u\theta$ is in $Gr^{\succ}(E)$-normal form;
- n_2 = The number of symbols occurring in equations $u =^? v \in P$ such that $u\theta = v\theta$ and $u\theta$ is in normal form;
- n_3 = The number of equations in P of the form $t =^? x$, where t is not a variable, and such that $t\theta = x\theta$ and $t\theta$ is in normal form.

The associated (well-founded) ordering is the lexicographic ordering using the natural ordering on positive integers.

We show by induction on this measure that if θ is a solution of a system $P;S'$, with S' in solved form, there exists a transformation sequence

$$P;S' \stackrel{*}{\Longrightarrow} \emptyset;S$$

where $\sigma_S \lessdot^{\mathcal{X}} \theta$ for $\mathcal{X} = \mathcal{V}ars(P, S')$.

The base case of the induction consists of a system $\emptyset; S$ and the result is trivial, since *a fortiori* $\sigma_S \lessdot \theta$. For the induction step, suppose $P = \{u =^? v\} \cup P'$. If $u\theta = v\theta$ with $u\theta$ in normal form; then we proceed as before with the inference system $\mathcal{U}$ to generate a transformation step to a smaller system containing the same set of variables, and with the same solution (cf. lemma 2.4). As with $\mathcal{U}$, any equation introduced into S must keep this set in solved form. Completing this with the induction hypothesis, we have

$$P; S' \implies_{\mathcal{U}} P''; S'' \stackrel{*}{\implies} \emptyset; S$$

such that $\sigma_S \lessdot^{\mathcal{X}} \theta$ with $\mathcal{X} = \mathcal{V}ars(P, S')$.

Otherwise, without loss of generality, pick a rewrite step from the term $u\theta$ in a minimal-length basic rewrite proof $u\theta \longrightarrow \stackrel{*}{\longrightarrow} \stackrel{*}{\longleftarrow} v\theta$, in which a reduced ground instance $l\rho \longrightarrow r\rho$ was used. If we let $\theta' = \theta\rho$, then this first step was in fact $u[u']\theta' = u[l]\theta' \longrightarrow u[r]\theta'$, where u' can not be a variable (since θ is reduced). In addition, the top symbols of u' and l are identical if l is not a variable. Hence, there exists some transformation step

$$\{u[u'] \stackrel{?}{=} v\} \cup P'; S' \implies_{lp} \{l \stackrel{?}{=} u', u[r] \stackrel{?}{=} v\} \cup P'; S'$$

to a new system which has a smaller complexity with respect to its new solution θ'. (It also contains additional variables, i.e., those in $\mathcal{V}ars(l, r)$). By the induction hypothesis we can continue this with:

$$\{l \stackrel{?}{=} u', u[r] \stackrel{?}{=} v\} \cup P'; S' \stackrel{*}{\implies} \emptyset; S$$

such that $\sigma_S \lessdot^{\mathcal{X}} \theta'$ with $\mathcal{X} = \mathcal{V}ars(l, r, P, S')$. But, since $x\theta = x\theta'$ for every $x \in \mathcal{V}ars(P, S')$, we are done. □

The second stage of our main completeness proof for $\mathcal{G}$ involves constructing a set of identities fitting the conditions of the previous lemma. We do this by a kind of abstract completion of E:

4.10. DEFINITION. Let $Cr(E)$ be the set of critical pairs w.r.t. $\succ$ of E, created from fresh variants of identities in E using the inference system $\mathcal{U}$ to calculate the requisite *mgu*'s. Then, for each $i \geq 0$, define

$$\begin{array}{rcl} E^0 & = & E \\ & \vdots & \\ E^{i+1} & = & E^i \cup Cr(E^i) \\ & \vdots & \\ E^\omega & = & \bigcup_{n \geq 0} E^n \end{array}$$

The entire point of this construction is contained in the following lemma, which can be proved using techniques familiar from [Dershowitz and Plaisted 2001], Chapter 9 of this Handbook (for a specific proof, see Theorem 6.1.7 in [Snyder 1991]).

4.11. LEMMA. *For any E, $Gr^{\succ}(E^{\omega})$ is ground convergent and equivalent to E on ground terms.*

Thus, we can (conceptually, at least) use E^{ω} to construct transformation sequences as just shown in lemma 4.9. The second main lemma of our completeness proof for $\mathcal{G}$ shows how to convert such a transformation sequence into one using only identities from E.

4.12. LEMMA. *For any sequence*

$$P; \emptyset \stackrel{*}{\Longrightarrow} \emptyset; S$$

introducing identities from E^{ω}, and such that σ_S is an E-unifier for P, there exists a sequence

$$P; \emptyset \stackrel{*}{\Longrightarrow} \emptyset; S'$$

introducing identities only from E, such that $S \subseteq S'$ and $x\sigma_{S'} = x\sigma_S$ for every $x \in \mathcal{V}ars(P)$.

PROOF. The basic idea is to use the calculus $\mathcal{G}$ itself to construct critical pairs. The complexity measure in our inductive proof is as follows. The *depth* of an identity $e \in E^{\omega}$ is the least k such that $e \in E^k$; the complexity of a transformation sequence is the (finite) multiset of the depths of all identities from E^{ω} introduced, with the associated (well-founded) multiset ordering.

The base case being trivial, we proceed directly to the induction step. Suppose the transformation sequence uses some identity $r_1\sigma \approx l_1[r_2]\sigma$ of non-zero depth, obtained by forming a critical pair from $l_1[l'] \approx r_1$ and $l_2 \approx r_2$ (each of smaller depth) with $\sigma = mgu(l', l_2)$. We show how the original use of the critical pair in a LP step can be simulated by two LP steps involving the component identities, plus some number of $\mathcal{U}$-transformations to simulate the construction of the critical pair. There are two cases, depending on which direction the critical pair was used in.

Case One. Suppose the critical pair was $r_1\sigma \approx l_1[r_2]\sigma$, e.g.,

$$\begin{array}{ll} \stackrel{*}{\Longrightarrow} & \{e[u]\} \cup P; S' \\ \Longrightarrow_{\mathsf{lp}} & \{r_1\sigma =^? u, e[l_1[r_2]\sigma]\} \cup P; S' \\ \stackrel{*}{\Longrightarrow} & \emptyset; S \end{array}$$

where an additional Decomposition is possibly applied afterwards to $r_1\sigma =^? u$ (if $r_1\sigma$ is not a variable). This sequence can be converted into:

$$\begin{array}{ll} \stackrel{*}{\Longrightarrow} & \{e[u]\} \cup P; S' \\ \Longrightarrow_{\mathsf{lp}} & \{r_1 =^? u, e[l_1[l_1']]\} \cup P; S' \\ \Longrightarrow_{\mathsf{lp}} & \{l_2 =^? l_1', r_1 =^? u, e[l_1[r_2]]\} \cup P; S' \\ \stackrel{*}{\Longrightarrow} & \{r_1\sigma =^? u, e[l_1[r_2]\sigma]\} \cup P; S \cup [\sigma] \\ \stackrel{*}{\Longrightarrow} & \emptyset; S \cup [\sigma'] \end{array}$$

(where by $[\sigma]$ we mean a set of equations representing the bindings in σ). This sequence has a smaller complexity, as it replaced a critical pair by two identities of strictly smaller depth. The second line from the bottom represents the calculation of the *mgu*; these bindings apply only to terms from the two equations, although as they are carried along in the solution set they may change as the result of additional substitutions (hence the change to σ'). The (possible) Decomposition step after the first LP step in the original is delayed until after the computation of σ.

Case Two. Suppose the critical pair was $l_1[r_2]\sigma \approx r_1\sigma$; in this case, we may assume that the overlap in this critical pair is not at the root, since otherwise we could apply case one. Our original sequence is thus:

$$\begin{array}{ll} \stackrel{*}{\Longrightarrow} & \{e[u]\} \cup P; S' \\ \Longrightarrow_{\mathsf{lp}} & \{l_1[r_2]\sigma =^? u, e[r_1\sigma]\} \cup P; S' \\ \stackrel{*}{\Longrightarrow} & \emptyset; S \end{array}$$

where Decomposition is applied to $l_1[r_1]\sigma \approx u$ at some point after the LP step (since l_1 has at least one function symbol above the overlap position). This sequence becomes:

$$\begin{array}{ll} \stackrel{*}{\Longrightarrow} & \{e[u]\} \cup P; S' \\ \Longrightarrow_{\mathsf{lp}} & \{l_1[l_1'] =^? u, e[r_1]\} \cup P; S' \\ \Longrightarrow_{\mathsf{lp}} & \{l_2 =^? l_1', l_1[r_2] =^? u, e[r_1]\} \cup P; S' \\ \stackrel{*}{\Longrightarrow} & \{l_1[r_2]\sigma =^? u, e[r_1\sigma]\} \cup P; S \cup [\sigma] \\ \stackrel{*}{\Longrightarrow} & \emptyset; S \cup [\sigma'] \end{array}$$

The Decomposition step is delayed until after the computation of σ. This sequence is, again, of smaller complexity than the original.

Note in both cases that the variables in $\mathcal{D}om(\sigma)$ are (effectively) fresh, as they occur in the component identities but not in the critical pair; thus, $x\sigma_{S'} = x\sigma_S$ for all $x \in \mathcal{V}ars(P)$ as required. □

We may now present the proof of our main completeness result.

Proof of theorem 4.2. First, note that we may assume that $P\theta$ contains only ground equations, using a straight-forward Skolemization argument (viz. [Snyder 1991], p.90). If θ is an E-unifier of P, we may construct an $Gr^{\succ}(E)$-reduced substitution θ' such that $\theta =_E \theta'$. We then apply lemma 4.9, using rules from E^ω, to obtain a sequence

$$P; \emptyset \stackrel{*}{\Longrightarrow} \emptyset; S$$

where $\sigma_S \leq^{\mathcal{X}} \theta'$ for $\mathcal{X} = \mathcal{V}ars(P)$. This is then converted, using the technique of lemma 4.12 to a new sequence using rules only from E:

$$P; \emptyset \stackrel{*}{\Longrightarrow} \emptyset; S'$$

where $x\sigma_S = x\sigma_{S'}$ for every $x \in \mathcal{V}ars(P)$. Thus, we may conclude that $\sigma_S \leq^{\mathcal{X}}_E \theta$, where $\mathcal{X} = \mathcal{V}ars(P)$, as required. □

4.2. Restrictions on E-unification in arbitrary theories

In this section we describe two refinements of the calculus $\mathcal{G}$ that have been suggested:

- The restriction on a equation $l =^? u$ introduced by LP, when l is not a variable, that the top symbol of l and u must be the same, can be strengthened so that the entire overlap of the non-variable positions in the two terms must be identical.
- The restriction in LP that u not be a variable may be strengthened so that u can not even be a term introduced into P by substitution (i.e., Variable Elimination) at any point in the sequence.

Both of these restrictions in some sense extend the original restrictions on $\mathcal{G}$ *hereditarily*, in the first case inheriting the restriction on top symbols down into the terms, and in the second, inheriting the non-variable restriction throughout the history of the equation, and regarding terms introduced by variable elimination as being second-class citizens which do not play a direct role in equational inferences, but only serve to constrain the application of rules. This is called the *basic* restriction, as it rests on the existence of basic rewrite proofs as shown above.

For lack of space, we do not consider these refinements to $\mathcal{G}$ in detail here, although the second will form an essential part of the calculus in the next section. For the first, see [Dougherty and Johann 1992], and also [Socher-Ambrosius 1994] (where a further refinement is presented); for the second see [Moser 1993].

4.3. Narrowing

In this section we consider the most important special case of the E-unification problem, when the equational theory can be represented by a ground convergent set of rewrite rules. In this case, the conversion of transformation sequences to simulate critical pair generation is not necessary, and we can take a closer look at the completeness proof and the restrictions that can be imposed on the calculus. In particular, we shall from the start consider the existence of basic rewrite proofs as fundamental, and develop a new representation for problems which prevents LP inferences at terms introduced by substitutions.

A *constraint system* (or simply *system* in the rest of the section) is either the symbol $\perp$ (representing failure) or a triple consisting of a multiset P of equations (representing the schema of the problem, in a sense that will become clear below), a set C of equations (representing constraints on variables in P), and a set S of equations (representing bindings in the solution). The set C plays a role similar to the multiset P in section 2.2.4, and rules from $\mathcal{U}$ will be applied to $C; S$ as before. The equational problems being worked on are in fact $P\sigma_S$, the separation into the schema P and constraints $C; S$ serving to enforce the *basic* restriction on the application of LP mentioned above. As expected, a substitution θ is said to be a solution (or E-unifier) of a system $P; C; S$ if it E-unifies each equation in P, and unifies each of the equations in C and S; the system $\perp$ has no E-unifiers.

We assume that our rewrite system R (representing E) is ground convergent with respect to a reduction ordering $\succ$, and consists of a numbered sequence of rules

$$\{l_1 \longrightarrow r_1, l_2 \longrightarrow r_2, \ldots, l_n \longrightarrow r_n\}.$$

The *index* of a rule will be its number in this sequence, and will be used in a certain refinement of our inference system.

4.3.1. The calculus $\mathcal{B}$

In this section we present the rules which are used in the calculus $\mathcal{B}$ for *basic narrowing*. We will first consider a simple set of rules and prove its completeness, and then consider refinements and modifications based on the details of the proof.

The set $\mathcal{B}$ consists of the following six rules.

Trivial:

$$P; \{s \stackrel{?}{=} s\} \cup C'; S \implies P; C'; S$$

Decomposition:

$$P; \{f(s_1, \ldots, s_n) \stackrel{?}{=} f(t_1, \ldots, t_n)\} \cup C'; S \implies P; \{s_1 \stackrel{?}{=} t_1, \ldots, s_n \stackrel{?}{=} t_n\} \cup C'; S$$

Orient:

$$P; \{t \stackrel{?}{=} x\} \cup C'; S \implies P; \{x \stackrel{?}{=} t\} \cup C'; S$$

if t is not a variable.

Basic Variable Elimination:

$$P; \{x \stackrel{?}{=} t\} \cup C'; S \implies P; C'\{x \mapsto t\}; S\{x \mapsto t\} \cup \{x \approx t\}$$

if x does not occur in t. (Note that the substitution is *not* applied to the set P.)

(Modulo the changes to Variable Elimination, these are just the non-failure rules from $\mathcal{U}$, adapted for constraint systems; we shall denote these first four rules as $\mathcal{S}$.)

Constrain:

$$\{e\} \cup P'; C; S \implies_{\mathsf{con}} P'; \{e\sigma_S\} \cup C; S$$

Lazy Paramodulation:

$$\{e[u]\} \cup P; C; S \implies_{\mathsf{lp}} \{e[r]\} \cup P; \{l\sigma_S \stackrel{?}{=} u\sigma_S\} \cup C; S$$

(with the exact same restrictions as given above in section 4.1.1).

Essentially, this calculus is no different from $\mathcal{G}$, except that it is designed to enforce the basic restriction, by separating out the parts of terms that were introduced into the problem by substitution (i.e., Variable Elimination) and those that were not (the "schema"). The latter constitute the only positions where equational inferences may take place in the basic strategy. The completeness proof is hence very similar to lemma 4.9. We will add more restrictions to the way that certain choices are made, however, which will give us the ability to restrict our calculus correspondingly.

4.13. THEOREM. *Let R be a ground convergent set of rewrite rules. If θ is an R-solution of $P;\emptyset;\emptyset$, then there exists a sequence*

$$P;\emptyset;\emptyset \stackrel{*}{\Longrightarrow}_{\mathcal{B}} \emptyset;\emptyset;S$$

such that $\sigma_S \lessdot_R^{\mathcal{X}} \theta$, where $\mathcal{X} = \mathcal{V}ars(P)$.

PROOF. As in our completeness proof for $\mathcal{G}$, we may assume that $P\theta$ is ground and that θ is R-reduced, since the relation $\lessdot_R$ does not distinguish between R-equivalent substitutions. Thus, we will prove a stronger result, that when θ is R-reduced, then in fact $\sigma_S \lessdot^{\mathcal{X}} \theta$.

The complexity of a system $P;C;S$ and associated solution θ is $\langle M, n_1, n_2, n_3\rangle$, where

M = The multiset of all terms occurring in $P\theta$;
n_1 = The number of distinct variables in C;
n_2 = The number of symbols in C;
n_3 = The number of equations in C of the form $t =^? x$, where t is not a variable.

The associated ordering is the lexicographic ordering using the multiset extension of the reduction ordering $\succ$ for the first component, and the ordering on natural numbers for the remaining components.

Our induction shows that if θ is a solution of a system $P;C;S'$, with S' in solved form, there exists a transformation sequence

$$P;C;S' \stackrel{*}{\Longrightarrow} \emptyset;\emptyset;S$$

where $\sigma_S \lessdot^{\mathcal{X}} \theta$, where $\mathcal{X} = \mathcal{V}ars(P, C, S')$.

The base case $\emptyset;\emptyset;S$ is again trivial. For the induction step, there are several overlapping cases.

(1) If $C = \{u =^? v\} \cup C'$, then $u\theta = v\theta$ and we use $\mathcal{S}$ to generate a transformation step to a smaller system containing the same set of variables, and with the same solution (cf. lemma 2.4). Completing this with the induction hypothesis, we have

$$P;C;S' \Longrightarrow_{\mathcal{S}} P'';C';S'' \stackrel{*}{\Longrightarrow} \emptyset;\emptyset;S$$

such that $\sigma_S \lessdot^{\mathcal{X}} \theta$ for $\mathcal{X} = \mathcal{V}ars(P, C, S')$.

(2) If $P = \{u =^? v\} \cup P'$ and $u\theta = v\theta$, then we may apply Constrain to obtain a smaller system (reducing the component M) with the same solution and the same set of variables, and we conclude as in the previous case.

(3) Suppose $P = \{u =^? v\} \cup P'$ and there is some redex in either $u\theta$ or $v\theta$; without loss of generally, assume the former. We may also assume that the redex is innermost, and that if more than one instance of a rule from R reduces this redex, we choose the rule $l\rho \longrightarrow r\rho$ with the smallest index in the set R. Note that, since θ is R-reduced, the redex must occur inside the non-variable positions of u; thus we have the following transformation:

$$\{u[u'] \stackrel{?}{=} v\} \cup P'; C; S' \implies_{lp} \{u[r] \stackrel{?}{=} v\} \cup P'; \{l\sigma_{S'} \stackrel{?}{=} u'\sigma_{S'}\} \cup C; S'$$

to a system which is smaller with respect to its new solution $\theta' = \theta\rho$ (since the new equation introduced into C is an identity modulo θ'). Note that θ' is still R-reduced. By the induction hypothesis we have

$$\{u[r] \stackrel{?}{=} v\} \cup P'; \{l\sigma_{S'} \stackrel{?}{=} u'\sigma_{S'}\} \cup C; S' \stackrel{*}{\Longrightarrow} \emptyset; \emptyset; S$$

such that $\sigma_S \trianglelefteq^{\mathcal{X}} \theta'$ with $\mathcal{X} = \mathcal{V}ars(l, r, P, C, S')$, and since $x\theta = x\theta'$ for every $x \in \mathcal{V}ars(P, C, S')$, the induction is complete. □

4.3.2. Standard narrowing

An interesting feature of this proof is that it also provides for the completeness of an alternate (and historically earlier) version of narrowing due to Fay [1979], which does not distinguish between substitution positions and other positions in the problem.

Let us define the calculus $\mathcal{N}$ for *standard narrowing* as the inference system $\mathcal{B}$ with the following change: Basic Variable Elimination is replaced by the following transformation:

Variable Elimination:

$$P; \{x \stackrel{?}{=} t\} \cup C'; S \implies P\{x \mapsto t\}; C'\{x \mapsto t\}; S\{x \mapsto t\} \cup \{x \approx t\}$$

if x does not occur in t.

(The Constrain rule might also be changed so that is does not instantiate an equation when moving it from P to C, however, since σ_S is always idempotent, the existing rule would have the same effect.)

The only difference is that the set P is kept instantiated with the substitution defined by S during the transformation process, so that substitution positions can be used for narrowing.

4.14. COROLLARY. *Let R be a ground convergent set of rewrite rules. If θ is an R-solution of $P; \emptyset; \emptyset$, then there exists a sequence*

$$P; \emptyset; \emptyset \stackrel{*}{\Longrightarrow}_{\mathcal{N}} \emptyset; \emptyset; S$$

in the calculus $\mathcal{N}$ such that $\sigma_S \trianglelefteq^{\mathcal{X}}_R \theta$ with $\mathcal{X} = \mathcal{V}ars(P)$.

The proof is essentially the same as the previous one, since the same transformation sequence can be used in each case.

The difference between the two inference systems is that $\mathcal{B}$ restricts the application of inference rules to a smaller set of positions than $\mathcal{N}$ does, and hence the search tree for solutions is narrower.

4.4. Strategies and refinements of basic narrowing

There is a variety of strategies and refinements that can be developed for the basic narrowing calculus without destroying completeness. Most of these, in one way or another, can be derived from a close examination of the completeness proof just given. In this section we briefly describe the most important of these.

4.4.1. Composite rules for basic narrowing

The first observation that can be made is that it is not necessary to consider all possible sequences of transformation rules, since we either solve (standard) unification problems (e.g., equations between two identical terms in $P\theta$) or simulate rewriting at the ground level by unifying left-hand sides of rules with non-variable positions in terms, at the non-ground level. Thus, we may use the following two composite rules as an alternate form of $\mathcal{B}$:

Solve ($\Longrightarrow_{\mathsf{sol}}$):

$$\{e\} \cup P'; C; S \;\Longrightarrow_{\mathsf{con}}\; P'; \{e\sigma_S\} \cup C; S \stackrel{*}{\Longrightarrow}_{\mathcal{S}} P'; C\eta; S\eta \cup [\eta]$$

(i.e., $\eta = mgu(e\sigma_S)$).

Narrow ($\Longrightarrow_{\mathsf{nar}}$):

$$\{e[u]\} \cup P; C; S \;\Longrightarrow_{\mathsf{lp}}\; \{e[r]\} \cup P; \{l\sigma_S \stackrel{?}{=} u\sigma_S\} \cup C; S \stackrel{*}{\Longrightarrow}_{\mathcal{S}} \;\{e[r]\} \cup P; C\eta; S\eta \cup [\eta]$$

(that is, $\eta = mgu(l\sigma_S, u\sigma_S)$), where $l \longrightarrow r$ is a fresh variant from R.

The completeness proof goes through with few changes. Note that in this formulation, no new equations remain in C after each step. A similar set of composite rules could be given for $\mathcal{N}$.

4.4.2. Simplification

The inference rules in $\mathcal{S}$ (like $\mathcal{U}$) are significant in that they can be applied whenever we want during a transformation sequence without affecting the outcome; in our inductive proof, we may observe that they make the problem smaller without changing the solution. Such rules are extremely important in reducing the search space for a solution.

4.15. DEFINITION. A transformation $\gg$ is called a *simplification rule* for $\mathcal{B}$ if whenever $P; C; S \gg P'; C'; S'$, then θ is an R-reduced solution of $P'; C'; S'$ iff $\theta|_{\mathcal{V}ars(P,C,S)}$ is an R-reduced solution to $P; C; S$, and $P'; C'; S'$ is smaller in the induction ordering used in Theorem 4.13 with respect to θ than $P; C; S$ w.r.t. $\theta|_{\mathcal{V}ars(P,C,S)}$.

The restrictions in this definition ensure that such a rule can be used any time it applies in the induction step to obtain a smaller system without changing the solution (w.r.t. the variables in the left side).

Thus, the rules in $\mathcal{S}$ are simplification rules in this respect. There are many other ad-hoc simplification rules that have been suggested for narrowing. For example, we may perform a form of Decomposition within P when we know that this does not remove a redex.

Problem Decomposition:

$$\{f(s_1, \ldots, s_n \stackrel{?}{=} f(t_1, \ldots, t_n)\} \cup P'; C'; S \implies \{s_1 \stackrel{?}{=} t_1, \ldots, s_n \stackrel{?}{=} t_n\} \cup P'; C; S$$

if the symbol f does not occur at the top of the left-side of a rule in R.

In the induction in the completeness proof this rule decreases the measure (specifically, it reduces the component M). Clearly it does not change the set of solutions. Therefore, we may apply this rule any time, in any context, without affecting the completeness properties of the calculus.

Such rules can be applied "eagerly" to produce smaller problems, hopefully reducing the search space.

4.16. DEFINITION. If $\mathcal{T}$ is a subset of rules for some calculus $\mathcal{C}$, then the *eager $\mathcal{T}$ strategy* requires that a rule from $\mathcal{C} \setminus \mathcal{T}$ may only be applied if no rule from $\mathcal{T}$ applies anywhere in the system.

Simplification rules can be performed eagerly.

4.17. THEOREM. *Let R be a ground convergent set of rewrite rules, and $\mathcal{A}$ be a set of simplification rules. If θ is an R-solution of $P; \emptyset; \emptyset$, then there exists a sequence*

$$P; \emptyset; \emptyset \stackrel{*}{\Longrightarrow}_{\mathcal{B} \cup \mathcal{A}} \emptyset; \emptyset; S$$

under the eager $\mathcal{A}$ strategy such that $\sigma_S \lessdot_R^{\mathcal{X}} \theta$, where $\mathcal{X} = \mathcal{V}ars(P)$.

The proof proceeds as before, with the exception that in the induction step, we must use a simplification step if one applies; as noted above, the conditions of a simplification rule ensure that the induction in the completeness proof goes through.

One of the most useful simplification rules is reducing the problem set by the set of rules R. From an abstract point of view, we may motivate such equational inferences as follows. If $u\theta \stackrel{*}{\longleftrightarrow}_E v\theta$ and $u' \stackrel{*}{\longleftrightarrow}_E u$, then, since equational proofs are closed under instantiation, we have $u'\theta \stackrel{*}{\longleftrightarrow}_E u\theta \stackrel{*}{\longleftrightarrow}_E v\theta$. Thus, we can not change the set of solutions by performing equational inferences on the problem terms themselves, for example, by reducing them.

From the point of view of our calculus, we might observe that in the rule Narrow just introduced, if no application of Variable Elimination is ever applied to a variable from the system on the left side, then the set of solutions is unchanged by this transformation: the substitution generated must in this case apply only to l and r, and hence we have, at the ground level, replaced $e[u]\theta\rho = e[l]\theta\rho = e[l\rho]\theta$ with $e[r\rho]\theta$. Since the properties of θ were not involved, this means that effectively we have done a rewrite step $u[l\rho] \longrightarrow_R u[r\rho]$. Alternately, we might say that if you end up doing Variable Elimination on $x =^? t$ for $x \in \mathcal{D}om(\theta)$ for some solution θ, then you are assuming that $x\theta = t\theta$; this cuts down on the number of possible solutions.

The resultant rule is:

Reduce ($\Longrightarrow_{\mathsf{red}}$):

$$\begin{aligned}\{e[u]\} \cup P; C; S \;&\Longrightarrow_{\mathsf{lp}}\; \{e[r]\} \cup P; \{l \stackrel{?}{=} u\sigma_S\} \cup C; S \\ &\stackrel{*}{\Longrightarrow} \{e[r\rho]\} \cup P; C; S \cup [\rho]\end{aligned}$$

where $l \longrightarrow r$ is a fresh variant from R (note that the variables in $\mathcal{D}om(\rho)$ occur only in r), and where the last line involves only Trivial, Decomposition, and Variable Elimination applied to the variables from l (i.e., $l\rho = u$).

Note that in the context of $\mathcal{B}$, we are losing some "basicness" by instantiating fully the right-hand side r; below we shall consider how to recover some of the basic restriction lost in this fashion.

4.18. PROPOSITION. *The Eager Reduce Strategy is complete for $\mathcal{B}$ and $\mathcal{N}$.*

Historically, the narrowing calculus was the first to be invented, by Fay [1979]; the basic narrowing calculus was developed by Hullot [1980], and it was observed by Réty [1987] that reduction needed to be modified in this setting. A study of basic narrowing with reduction, to which our treatment is heavily indebted, may be found in [Nutt, Réty and Smolka 1989]. In the next two sections we present further refinements which may also be found in [Bockmayr, Krischer and Werner 1992] and [Nutt et al. 1989]. For a comprehensive study of basic inference systems, the reader is referred to [Bachmair, Ganzinger, Lynch and Snyder 1995] and to [Nieuwenhuis and Rubio 2001] (Chapter 7 of this Handbook).

4.4.3. Redex orderings and variable abstraction

One of the useful properties of convergent systems mentioned above is that any strategy which can find a redex in a reducible term is sufficient for reducing terms to normal form, and hence for generating rewrite proofs. For example, at the ground level we might always look for redices in depth-first, left-to-right order. More generally, we may define a *redex ordering* $\prec_{red}$ as an ordering on the positions in an equation which contains the proper subterm ordering (i.e., for any $u[u']$ with $u \neq u'$, we have $u' \prec_{red} u$). Before considering whether a term t is reducible at a position π by some rule, we must consider all positions $\pi' \prec_{red} \pi$. The completeness proof could be sharpened by such an ordering simply by adding that we must choose the minimal redex according to the redex ordering (such a redex must be innermost). In such a case, the positions less than this redex may be assumed to be irreducible. No further narrowing steps need be performed at such positions, and in fact, we could remove these parts of the term and move them into the solved part of the system to enforce this.

Variable Abstraction ($\Longrightarrow_{\mathsf{abst}}$):

$$\{e[s]\} \cup P; C; S \implies \{e[x]\} \cup P; \{x \stackrel{?}{=} s\} \cup C; S$$

if x is a fresh variable.

A new version of the narrowing rule could then be presented which abstracts out terms which are known to be reduced.

Redex Ordered Narrow ($\Longrightarrow_{\mathsf{ron}}$):

$$\{e[u]\} \cup P; C; S \Longrightarrow_{\mathsf{lp}} \{e[r]\} \cup P; \{l\sigma_S \stackrel{?}{=} u\sigma_S\} \cup C; S \stackrel{*}{\Longrightarrow}_{\mathcal{S}} \{e[r]\} \cup P; C\eta; S\eta \cup [\eta]$$
$$\stackrel{*}{\Longrightarrow}_{\mathsf{abst}} \{e'[r]\} \cup P; C\eta \cup C'; S\eta \cup [\eta]$$

where u occurs at position π in e, and Variable Abstraction is applied eagerly to all positions $\pi' \prec_{red} \pi$ in e to obtain e'.

The substitution of this version of Narrow in $\mathcal{N}$ preserves completeness; the fundamental idea is that whenever a term (at the ground level in our completeness proof) may be assumed to be reduced, it may be moved into the constraint part of the system without losing completeness. This leads to a further use for Variable Abstraction in propagating what is known about reduced terms: if a term occurs in S, then (at the ground level) it may be assumed to be reduced, and hence other occurrences of this term may be abstracted out.

Propagation:

$$\{e[u]\} \cup P'; C; \{x \approx t[s]\} \cup S \Longrightarrow_{\mathsf{prop}} \{e[y]\} \cup P'; C; \{x \approx t[s], y \approx s\} \cup S$$

if $u\sigma_S = s$ is a non-variable and y is a fresh variable.

This rule is a simplification rule if we change the complexity measure in the proof to

$$\langle M, i, n_1, n_2, n_3 \rangle$$

where the additional component i is the number of non-variable symbols occurring in P. Clearly it changes the solution θ of a system to a new solution $\theta\{y \mapsto s\theta\}$ which satisfies the condition for a simplification rule.

Returning to our Reduce rule, we observe that in the context of $\mathcal{B}$, Reduce may instantiate terms into r that are known to be reduced; Propagation can remove these again. The combination of Reduction with Eager Propagation effectively gives us the more complex form of "basic simplification" described for example in [Bachmair et al. 1995] and [Nutt et al. 1989], see also [Nieuwenhuis and Rubio 2001] (Chapter 7 of this Handbook).

4.4.4. Failure rules

Unlike our presentation of the calculus $\mathcal{U}$, we have chosen here not to present failure rules from the outset, in order to highlight the essential issues first. The conditions under which sequences may fail are of two kinds. First, the failure rules for $\mathcal{U}$ (Symbol Clash and Occur Check) may be applied to the sets C and S as before, since these represent unification problems; however, in this case the corresponding Solve, Narrow, or Reduce would simply not be performed.

The second class of conditions basically amount to checking for violations of the reducibility conditions in a system. At the ground level during the completeness proof, the substitution θ is kept reduced, and in addition, certain assumptions can be made about the existence of redices in terms. However, we have to be careful, as our proof only allows us to assume that all substitutions are R-reduced, and that no redex may be reduced below its root, or at the root by an equation of lower index.

This leads to the following rule:

Blocking ($\Longrightarrow_{\mathsf{block}}$):

$$P; C; S \implies \bot$$

if some term in S is R-reducible, or if some term in C is reducible below the root.

The Eager Blocking Strategy is complete, since the completeness proof requires the converse of the condition of this rule at all times. Note that this rule could be applied in the middle of a composite rule, for example, just after moving the equation into the set C in Narrow.

In order to account for reduction at the top of equations in C, it is preferable to add a further restriction to our Narrowing rule:

Narrow ($\Longrightarrow_{\mathsf{nar}}$):

$$\{e[u]\} \cup P; C; S \implies_{\mathsf{lp}} \{e[r]\} \cup P; \{l\sigma_S \stackrel{?}{=} u\sigma_S\} \cup C; S \stackrel{*}{\Longrightarrow}_{\mathcal{S}} \{e[r]\} \cup P; C\eta; S\eta \cup [\eta]$$

where $l \longrightarrow r$ is a fresh variant from R and $l\sigma_S\eta$ is not the instance of the left-side of any rule of lower index from R.

This rule is consistent with Redex Orderings.

5. Semantic approaches to E-unification

The syntactic approaches to E-unification introduced above can be seen as extensions of the rule-based approach to syntactic unification, which use the identities defining the equational theory E to come up with additional transformation rules. In contrast, semantic approaches to E-unification try to utilize algebraic properties of the models of the equational theories. The two most prominent instances of the approach are

1. Unification in Boolean algebras and rings [Büttner and Simonis 1987, Martin and Nipkow 1989*b*, Martin and Nipkow 1989*a*], and its generalization to finite and to primal algebras [Büttner 1988, Büttner, Estenfeld, Schmid, Schneider and Tidén 1990, Nipkow 1990, Kirchner and Ringeissen 1994], and
2. Unification modulo the theories ACU, $ACUI$, and AG (see subsection 3.4 for references to result on unification modulo these theories).

In the following, we concentrate on the approach used in the second case since it can be generalized to a whole class of equational theories, called commutative theories in [Baader 1989*b*] and monoidal theories in [Nutt 1990]. For such theories, unification can be reduced to solving linear equations in a corresponding semiring.[13] In the following, we introduce the class of commutative/monoidal theories, show how the corresponding semiring is defined, and how unification in commutative/monoidal theories can be reduced to solving linear equations in this semiring. In contrast to the syntactic approaches introduced above, general unification problems cannot be solved directly by the semantic approach described below. However, for commutative/monoidal theories, the known techniques for combining unification algorithms can always be used to extend an algorithm for unification with constants to an algorithm for general unification [Baader and Nutt 1996].

The theories

$$\begin{aligned} ACU &:= \{f(x,y) \approx f(y,x),\ f(f(x,y),z) \approx f(x,f(y,z)),\ f(x,e) \approx x\}, \\ ACUI &:= ACU \cup \{f(x,x) \approx x\}, \\ AG &:= ACU \cup \{f(x,i(x)) \approx e\} \end{aligned}$$

will be used as examples throughout this section. The introduction of the class of commutative/monoidal theories was motivated by the observation that the known algorithms for unification modulo these three theories have many common features.

5.1. *Unification modulo ACU, ACUI, and AG: an example*

We will first restrict our attention to elementary unification, and then show how the methods can be extended to unification with constants.

Elementary unification

To illustrate how the algorithms for elementary unification modulo these three theories work, let us consider the problem of unifying the two terms $f(x, f(x,y))$ and $f(z, f(z,z))$.

Let us start with the theory ACU. Obviously, the substitution $\sigma_1 := \{x \mapsto z_1, y \mapsto z_1, z \mapsto z_1\}$ is a syntactic unifier of this pair of terms, and thus also an ACU-unifier of $\Gamma_{ACU} := \{f(x, f(x,y)) =^?_{ACU} f(z, f(z,z))\}$. There are, however, ACU-unifiers of Γ_{ACU} that are not syntactic unifiers of the two terms: $\sigma_2 := \{x \mapsto$

[13] A semiring is similar to a ring, with the only difference being that its addition is just required to form an Abelian monoid, and not necessarily an Abelian group.

e, $y \mapsto f(z_2, f(z_2, z_2))$, $z \mapsto z_2\}$ is an example of such a unifier, and $\sigma_3 := \{x \mapsto f(z_3, f(z_3, z_3))$, $y \mapsto e$, $z \mapsto f(z_3, z_3)\}$ is another one. None of these substitutions is a most general ACU-unifier of Γ_{ACU}, but their "combination"

$$\begin{aligned}\sigma := \quad & \{x \mapsto f(x\sigma_1, f(x\sigma_2, x\sigma_3)),\ y \mapsto f(y\sigma_1, f(y\sigma_2, y\sigma_3)),\\ & \ z \mapsto f(z\sigma_1, f(z\sigma_2, z\sigma_3))\}\\ =_{ACU} & \{x \mapsto f(z_1, f(z_3, f(z_3, z_3))),\ y \mapsto f(z_1, f(z_2, f(z_2, z_2))),\\ & \ z \mapsto f(z_1, f(z_2, f(z_3, z_3)))\}\end{aligned}$$

is. For example, σ_2 can be obtained as an ACU-instance of σ by applying the substitution $\{z_1 \mapsto e,\ z_3 \mapsto e\}$. More generally, any finite collection $\sigma_1, \ldots, \sigma_n$ of ACU-unifiers of a given ACU-unification problem can be combined in this way to a new ACU-unifier σ, which has all the unifiers σ_i as ACU-instances. In our example, there still remains the question of how we have found the three unifiers $\sigma_1, \sigma_2, \sigma_3$, and why their combination is a most general ACU-unifier of the problem.

In order to explain how we came up with these unifiers, assume that τ is an ACU-unifier of Γ_{ACU}, and that z' is a variable introduced by τ, i.e., z' occurs in (at least) one of the terms $x\tau, y\tau, z\tau$. It is easy to see that $f(x, f(x, y))\tau =_{ACU} f(z, f(z, z))\tau$ implies that the number of occurrences of z' in $f(x, f(x, y))\tau$ coincides with the number of occurrences of z' in $f(z, f(z, z))\tau$. Thus, if $|x\tau|_{z'}, |y\tau|_{z'}, |z\tau|_{z'}$ respectively denote the number of occurrences of z' in $x\tau, y\tau, z\tau$, then we have $2|x\tau|_{z'} + |y\tau|_{z'} = 3|z\tau|_{z'}$, i.e., the numbers $|x\tau|_{z'}, |y\tau|_{z'}, |z\tau|_{z'}$ are nonnegative integer solutions of the linear equation

$$2x + y = 3z.$$

Thus, every variable introduced by an ACU-unifier of a given ACU-unification problem yields a non-trivial[14] solution of the linear equation corresponding to the problem in the semiring of all nonnegative integers (with addition and multiplication as semiring operations). For the unifier σ introduced above, the variable z_1 yields the solution (1, 1, 1), z_2 yields (0, 3, 1), and z_3 yields (3, 0, 2). What makes these three solutions special is that they are the minimal non-trivial solutions of $2x + y = 3z$ (w.r.t. the component-wise $\leq$-ordering on triples). Consequently, any solution can be obtained as a (nonnegative) linear combination of these three solutions.

Conversely, a substitution that introduces only variables (or free constants) corresponding to solutions of the linear equation is an ACU-unifier of the corresponding ACU-unification problem. For example, the substitution $\tau := \{x \mapsto f(z' f(z'', f(z'', z'')))$, $y \mapsto f(z', f(z', f(z', z')))$, $z \mapsto f(z', f(z' f(z'', z'')))\}$ is an ACU-unifier of Γ_{ACU} since $2 \cdot 1 + 4 = 3 \cdot 2$ and $2 \cdot 3 + 0 = 3 \cdot 2$. The solutions (1, 4, 2) and (3, 0, 2) can be obtained as linear combination of the minimal solutions:

$$\begin{aligned}(1, 4, 2) &= 1 \cdot (1, 1, 1) + 1 \cdot (0, 3, 1) + 0 \cdot (3, 0, 2),\\ (3, 0, 2) &= 0 \cdot (1, 1, 1) + 0 \cdot (0, 3, 1) + 1 \cdot (3, 0, 2).\end{aligned}$$

[14] Variables not introduced by the unifier correspond to the trivial solution $(0, \ldots, 0)$.

This fact can be used to obtain a substitution λ such that $u\tau =_{ACU} u\sigma\lambda$ for all $u \in \{x, y, z\}$: $\lambda := \{z_1 \mapsto z',\, z_2 \mapsto z',\, z_3 \mapsto z''\}$.

To sum up, we have seen that a given elementary ACU-unification problem corresponds to a system[15] of linear equations, which must be solved in the semiring $\mathcal{N}$ of all nonnegative integers. A most general ACU-unifier of the problem is obtained by combining the unifiers corresponding to the (finitely many) minimal solutions of the system of linear equations. The important property of the set of minimal solutions is that it generates all solutions as linear combinations in $\mathcal{N}$. The fact that this set is always finite is an easy consequence of Dickson's Lemma [Dickson 1913]. Methods for computing this set can, for example, be found in [Huet and Lang 1978, Lambert 1987, Clausen and Fortenbacher 1989, Boudet et al. 1990, Pottier 1991, Domenjoud 1991, Contejean and Devie 1994, Filgueira and Tomás 1995].

The theory $ACUI$ can be treated similarly, with the only difference being that the semiring $\mathcal{N}$ must be replaced by the Boolean semiring $\mathcal{BS}$, which consists of the truth values 0 and 1, and has conjunction as its multiplication and disjunction as its addition operation. In fact, modulo $ACUI$ it is no longer necessary that the *numbers* of occurrences of variables on the left-hand side and the right-hand side of the equation coincide. It is sufficient that each variable that occurs on the right-hand side also occurs on the left-hand side and vice versa. Thus, the linear equation corresponding to the $ACUI$-unification problem $\Gamma_{ACUI} := \{f(x, f(x, y)) =^?_{ACUI} f(z, f(z, z))\}$ is $x + y = z$, and it is easy to see that all solutions in $\mathcal{BS}$ can be generated as linear combinations in $\mathcal{BS}$ of the solutions (1, 0, 1) and (0, 1, 1). The most general $ACUI$-unifier obtained from this generating set of solutions is $\sigma' := \{x \mapsto z_1,\, y \mapsto z_2,\, z \mapsto f(z_1, z_2)\}$. The ACU-unifier σ_1 from above is also an $ACUI$-unifier of Γ_{ACUI}, and it can be obtained as an $ACUI$-instance of σ' via the substitution $\lambda' := \{z_1 \mapsto z_1,\, z_2 \mapsto z_1\}$. Since the Boolean semiring $\mathcal{BS}$ is finite, there always exists a *finite* set of solutions that generates all solutions as linear combinations in $\mathcal{BS}$.

For the theory AG, the presence of the inverse operation leads to the fact that both the coefficients and the solutions of the linear equations corresponding to an AG-unification problem may also be negative integers. Thus, the semiring to be considered here is an fact a ring, namely the ring $\mathcal{Z}$ of all integers. The linear equation corresponding to the AG-unification problem $\Gamma_{AG} := \{f(x, f(x, y)) =^?_{AG} f(z, f(z, z))\}$ coincides with the one obtained from Γ_{ACU}, but in $\mathcal{Z}$ there exists a smaller set generating all solutions, consisting of (0, 3, 1) and (1, −2, 0). Thus, the substitution $\sigma'' := \{x \mapsto z_2, y \mapsto f(z_1, f(z_1, f(z_1, f(i(z_2), i(z_2))))),\, z \mapsto z_1\}$ is a most general AG-unifier of Γ_{AG}. General methods for computing such a finite generating set of solutions of systems of linear equations in $\mathcal{Z}$ can, for example, be found in [Knuth 1981, Kannan and Bachem 1979, Iliopoulos 1989*a*, Iliopoulos 1989*b*].

[15] Every equation in the unification problem yields one linear equation.

Unification with constants
For ACU-unification with constants, there are two different ways of extending the approach for elementary unification to the case of unification with constants. The approach originally proposed by Stickel [1975] and [1981] first solves an elementary ACU-unification problem, which is obtained by treating free constants as variables, and then modifies the solutions of the elementary problem to obtain solutions of the problem with constants. The other approach, due to Livesey and Siekmann [1975] and described in more detail in [Herold and Siekmann 1987], handles free constants with the help of inhomogeneous linear equations. In the following, we restrict our attention to this second method.

As an example, we slightly modify the ACU-unification problem from above. Let $\Gamma'_{ACU} := \{f(x, f(x,y)) =^?_{ACU} f(a, f(z, f(z,z)))\}$, where a is a (free) constant. Of course, the numbers of occurrences $|x\tau|_{z'}, |y\tau|_{z'}, |z\tau|_{z'}$ of a variable z' introduced by an ACU-unifier of this problem must still solve the (homogeneous) linear equation $2x + y = 3z$. For the free constant a, however, one must also take into account that a already occurs once on the right-hand side. Thus, the numbers $|x\tau|_a, |y\tau|_a, |z\tau|_a$ must solve the following *inhomogeneous* equation:

$$2x + y = 3z + 1.$$

The minimal (non-trivial) nonnegative integer solutions of this equation are (0, 1, 0) and (2, 0, 1). Every nonnegative integer solution of the equation can be obtained as the sum of one of the minimal solution and a solution of the corresponding homogeneous equation $2x + y = 3z$. Consequently, each of the minimal solutions of the inhomogeneous equation together with the set of all minimal solutions of the homogeneous equation gives rise to one element of the minimal complete set of ACU-unifiers of the problem:

$$\begin{aligned}
\{\{&x \mapsto f(z_1, f(z_3, f(z_3, z_3))),\ y \mapsto f(a, f(z_1, f(z_2, f(z_2, z_2)))),\\
&z \mapsto f(z_1, f(z_2, f(z_3, z_3)))\},\\
\{&x \mapsto f(a, f(a, f(z_1, f(z_3, f(z_3, z_3))))),\ y \mapsto f(z_1, f(z_2, f(z_2, z_2))),\\
&z \mapsto f(a, f(z_1, f(z_2, f(z_3, z_3))))\}\ \}.
\end{aligned}$$

In the general case, one must solve one inhomogeneous equation for each free constant occurring in the unification problem. The unifiers in the minimal complete set then correspond to all possible combinations of the minimal solutions of these inhomogeneous equations. For example, if the unification problem contains the free constants a, b, c, and if the sets of minimal solutions of the inhomogeneous equations induced by a, b, and c, respectively, have cardinality $2, 3$, and 5, then the minimal complete set is of cardinality $2 \cdot 3 \cdot 5 = 30$.

Unification with constants modulo the theories $ACUI$ and AG can be treated accordingly. In both cases, one works in the semiring corresponding to the theory, and first determines a generating set of solutions for the system of homogeneous equations corresponding to the unification problem. Then, one considers the systems of inhomogeneous equations induced by the free constants, and for each system determines finitely many solutions such that all solutions of this system of inhomogeneous equations can be represented as the sum of one of these particular solutions and a solution of the homogeneous equation. From these sets of solutions, the minimal complete set of unifiers can be computed, as illustrated in the above example.

For AG, the fact that the corresponding semiring is a ring implies that taking one particular solution for each system of inhomogeneous equations is sufficient. Consequently, AG is unitary both for elementary unification and for unification with constants, whereas the other two theories, though unitary for elementary unification, are only finitary for unification with constants.

5.2. The class of commutative/monoidal theories

In order to generalize this semantic approach to a whole class of theories, let us try to determine the relevant common features of the theories ACU, $ACUI$, and AG. Using a rather syntactic point of view, we may observe that all three theories are concerned with an associative-commutative binary function symbol f with a unit e. In addition, the signature of AG contains a unary function symbol i, which behaves like an endomorphism for f and e, i.e., $i(f(x,y)) =_{AG} f(i(x), i(y))$ and $i(e) =_{AG} e$. This observation motivates the following definition of monoidal theories [Nutt 1990]:

5.1. DEFINITION. An equational theory E is called *monoidal* iff it satisfies the following properties:
1. $\mathcal{S}ig(E)$ contains a binary function symbol f and a constant symbol e, and all other function symbols in $\mathcal{S}ig(E)$ are unary.
2. The symbol f is associative-commutative with unit e, i.e., $f(f(x,y),z) =_E f(x, f(y,z))$, $f(x,y) =_E f(y,x)$, and $f(x,e) =_E x$.
3. Every unary function symbol $h \in \mathcal{S}ig(E)$ is an endomorphism for f and e, i.e., $h(f(x,y)) =_E f(h(x), h(y))$ and $h(e) =_E e$.

Obviously, the theories ACU, $ACUI$, and AG are monoidal. Other examples of monoidal theories are the theories $E_{h,g} \cup E_{h,e} \cup ACU_g$, $E_{h,g} \cup E_{h,e} \cup ACUI_g$, and $E_{h,g} \cup E_{h,e} \cup AG_g$ introduced in subsection 3.4. The theory of Boolean rings and the theory of commutative rings are not monoidal since their signatures contain *two* binary function symbols.

A drawback of the above definition of monoidal theories is that the signature and the axioms defining a theory play an important rôle. In fact, the theory of Abelian groups allows for many different axiomatizations, some of which do not satisfy the definition of a monoidal theory. For example, let g be a binary function symbol and e be a constant symbol. The theory

$$AG' := \{g(x,x) \approx e,\ g(x,e) \approx e,\ g(g(x, g(e,y)), g(e,z)) \approx g(g(z, g(e,y)), g(e,x))\}$$

is not monoidal since g is neither associative nor commutative modulo AG'. Nevertheless, any model of AG' is an Abelian group, where the group operations f and i are defined as $f(x,y) := g(x, g(e,y))$ and $i(x) := g(e,x)$.

In order to capture theories like AG' as well, one must take a more semantic point of view. A common feature of the free algebras defined by ACU, $ACUI$, and AG is that the finitely generated free algebras are *direct powers* of the free algebras in one generator. For example, it is well known that the free Abelian group in one generator is just the additive group of the integers, and that the free Abelian group in n generators is the n-fold direct product of this group. As shown in [Baader 1989*b*], this common feature can nicely be generalized in the categorical setting introduced in subsection 3.3.3:

5.2. DEFINITION. Let E be an equational theory and $\mathcal{F} := \mathcal{S}ig(E)$. Then E is a *commutative* theory iff $C_{\mathcal{F}}(E)$ is a semi-additive category,[16] i.e.,

1. $C_{\mathcal{F}}(E)$ has a zero object.
2. For every pair of objects in $C_{\mathcal{F}}(E)$, their coproduct is also their product.

In algebraic terms, the first condition means that the initial algebra in $V(E)$, i.e., $\mathcal{T}(\mathcal{F},\emptyset)/_{=_E}$, is of cardinality 1. Since the coproduct of $\mathcal{T}(\mathcal{F},\mathcal{X})/_{=_E}$ and $\mathcal{T}(\mathcal{F},\mathcal{Y})/_{=_E}$ is simply $\mathcal{T}(\mathcal{F},\mathcal{X}\uplus\mathcal{Y})/_{=_E}$ (where $\uplus$ denotes disjoint union), the second condition means that the free algebra $\mathcal{T}(\mathcal{F},\mathcal{X}\uplus\mathcal{Y})/_{=_E}$ is isomorphic to the direct product $\mathcal{T}(\mathcal{F},\mathcal{X})/_{=_E} \times \mathcal{T}(\mathcal{F},\mathcal{Y})/_{=_E}$. In particular, this implies that the finitely generated E-free algebras are direct powers of the E-free algebra in one generator.

The theory of Abelian groups satisfies these properties (and thus is commutative). The theory of Boolean rings and the theory of commutative rings are not commutative in the sense of the above definition since the initial algebras contain two elements (the constants 0 and 1).

In order to obtain a more algebraic definition of commutative theories, which also makes clear that all monoidal theories are commutative, we need two more notions from universal algebra. A constant symbol $e \in \mathcal{F}$ is called *idempotent* in E iff $f(e,\ldots,e) =_E e$ holds for all $f \in \mathcal{F}$. Any term $t(x_1,\ldots,x_n)$ over the signature $\mathcal{F}$ defines an n-ary *implicit operation* o_t in $V(E)$: for an algebra $\mathcal{A} \in V(E)$, the result of applying o_t to elements $a_1,\ldots,a_n$ of the carrier of $\mathcal{A}$ is obtained by evaluating $t(a_1,\ldots,a_n)$ in $\mathcal{A}$. For example, the terms $g(x,g(e,y))$ and $g(e,x)$ define a binary and a unary implicit operation, which together with the constant e satisfy the axioms of Abelian groups in all models of AG', i.e., all algebras in $V(AG')$.

5.3. PROPOSITION. *Let E be an equational theory and $\mathcal{F} := \mathcal{S}ig(E)$. Then E is a commutative theory iff*

1. *The signature $\mathcal{F}$ contains a constant e that is idempotent in E.*
2. *There is a binary implicit operation $*$ in $V(E)$ such that*
   (a) *The constant e is a unit for $*$ in all algebras in $V(E)$.*
   (b) *For any n-ary function symbol $h \in \mathcal{F}$, the identity $h(x_1 * y_1,\ldots,x_n * y_n) \approx h(x_1,\ldots,x_n) * h(y_1,\ldots,y_n)$ holds in all algebras in $V(E)$.*

[16]See, e.g., [Herrlich and Strecker 1973, Baader 1989*b*] for a more precise definition of and more information on semi-additive categories.

Although it is not explicitly required by the proposition, the implicit operation $*$ turns out to be associative and commutative. Using this proposition, it is easy to show that the theory AG' is indeed commutative: the implicit operation $*$ is defined by the term $g(x, g(e, y))$.

Another easy consequence of the proposition is that every monoidal theory is commutative: just take the explicit associative-commutative binary operation f in the definition of monoidal theories as the implicit operation $*$. The theory AG' is an example of a commutative theory that is not monoidal. However, it can be shown [Baader and Nutt 1996] that every commutative theory can be turned into an "equivalent" monoidal theory with the help of a signature transformation. For this reason, one can in principle use both notions synonymously.

5.3. The corresponding semiring

Let E be a commutative theory with $\mathcal{S}ig(E) = \mathcal{F}$. The semiring S_E corresponding to E is obtained by considering the E-free algebra in one generator, say x, and then taking the set of all endomorphisms of this algebra. Each such endomorphism is uniquely determined by the image of the generator x. The multiplication operation "$\cdot$" in S_E is just composition of morphisms, and the addition operation "+" is obtained by argument-wise application of the implicit operation $*$ of the commutative theory E: $(\sigma + \tau)(x) := \sigma(x) * \tau(x)$.

As an example, we consider the commutative theory $ACUI$, where the explicit operation f serves as the implicit operation $*$. Since the $ACUI$-free algebra generated by x consists of two equivalence classes, with representatives x and e, respectively, there are two possible endomorphisms: 0, which is defined by $x \mapsto e$, and 1, which is defined by $x \mapsto x$. It is easy to see that the operation "+" in S_{ACUI} behaves like disjunction and "$\cdot$" like conjunction on the truth values 0 and 1. For example, $(0 \cdot 1)(x) = 1(0(x)) = 1(e) = e = 0(x)$ and $(0 + 1)(x) = f(0(x), 1(x)) = f(e, x) =_{ACUI} x = 1(x)$. Consequently, S_{ACUI} is the two-element Boolean semiring $\mathcal{BS}$.

A well-known result for semi-additive categories [Herrlich and Strecker 1973] says that morphisms σ in the semi-additive category $C_{\mathcal{F}}(E)$ can be represented as matrices M_σ over S_E such that composition of morphisms corresponds to matrix multiplication, i.e., $M_{\sigma\tau} = M_\sigma \cdot M_\tau$. For example, the morphism $\sigma\colon \mathcal{T}(\mathcal{F}, \{x_1, x_2\})/_{=_{ACUI}} \to \mathcal{T}(\mathcal{F}, \{y_1, y_2\})/_{=_{ACUI}}$ defined by $\sigma(x_1) := f(y_1, y_2)$, $\sigma(x_2) := y_2$ corresponds to the matrix

$$M_\sigma = \begin{pmatrix} \{x_1 \mapsto y_1\} & \{x_1 \mapsto y_2\} \\ \{x_2 \mapsto e\} & \{x_2 \mapsto y_2\} \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}.$$

The second equality depends on the fact that all E-free algebras in one generator are isomorphic, and thus a morphism $\sigma_{ij}\colon \mathcal{T}(\mathcal{F}, \{x_i\})/_{=_E} \to \mathcal{T}(\mathcal{F}, \{y_j\})/_{=_E}$ can be seen as an endomorphism of $\mathcal{T}(\mathcal{F}\{x\})/_{=_E}$, i.e., an element of S_E.

5.4. Results on unification in commutative theories

Let E be a commutative theory with $\mathcal{S}ig(E) = \mathcal{F}$. In subsection 3.3.3 we have seen that any E-unification problem over $\mathcal{F}$ corresponds to a parallel pair σ, τ: $\mathcal{T}(\mathcal{F}, \mathcal{I})/_{=_E} \to \mathcal{T}(\mathcal{F}, \mathcal{X})/_{=_E}$ of morphisms in $C_{\mathcal{F}}(E)$, and that an E-unifier corresponds to a morphism δ with domain $\mathcal{T}(\mathcal{F}, \mathcal{X})/_{=_E}$ such that $\sigma\delta = \tau\delta$ holds in $C_{\mathcal{F}}(E)$.

If we translate the morphisms into matrices over S_E, this means that an E-unifier of the parallel pair $\langle\sigma, \tau\rangle$ corresponds to a matrix M over S_E such that $M_\sigma \cdot M = M_\tau \cdot M$. This correspondence is used in [Nutt 1990, Baader 1993] to characterize the unification types of commutative theories by algebraic properties of the corresponding semirings. The rows of the matrix M are n-tuples of elements of S_E, written as row vectors. We will denote the set of all such n-dimensional row vectors over S_E by S_E^n.

5.4. THEOREM. *A commutative theory E is unitary w.r.t. elementary unification iff the corresponding semiring S_E satisfies the following condition: for all $m, n \geq 1$ and all $m \times n$-matrices M_1, M_2 over S_E the set*

$$U(M_1, M_2) := \{v \in S_E^n \mid M_1 \cdot v = M_2 \cdot v\}$$

is finitely generated, i.e., there exist $k \geq 0$ and $v_1, \ldots, v_k \in S_E^n$ such that $U(M_1, M_2) = \{v_1 \cdot s_1 + \cdots + v_k \cdot s_k \mid s_1, \ldots, s_k \in S_E\}$.

If $\{v_1, \ldots, v_k\}$ is such a finite generating set for $U(M_\sigma, M_\tau)$, then the matrix whose columns are the vectors $v_1, \ldots, v_k$ corresponds to the most general E-unifier of $\langle\sigma, \tau\rangle$.

Unification with constants can also be reformulated as a problem in $C_{\mathcal{F}}(E)$ for $\mathcal{F} = \mathcal{S}ig(E)$. To this end we view constants as special variables that must always be substituted for themselves. Let $\mathcal{C}$ be a finite set of free constants. We say that a morphism σ: $\mathcal{T}(\mathcal{F}, \mathcal{X} \cup \mathcal{C})/_{=_E} \to \mathcal{T}(\mathcal{F}, \mathcal{Y} \cup \mathcal{C})/_{=_E}$ respects the constants in $\mathcal{C}$ iff $c\sigma = c$ for all $c \in \mathcal{C}$. In this case, the matrix M_σ has a special form:

$$M_\sigma = \begin{pmatrix} M_\sigma^h & M_\sigma^i \\ 0 & U \end{pmatrix},$$

where M_σ^h is an $|\mathcal{X}| \times |\mathcal{Y}|$-matrix, M_σ^i is an $|\mathcal{X}| \times |\mathcal{C}|$-matrix, 0 is the $|\mathcal{C}| \times |\mathcal{Y}|$-matrix with all entries 0, and U is the $|\mathcal{C}| \times |\mathcal{C}|$-unit matrix. The 0-submatrix is due to the fact that σ does not substitute terms with variables for constants, and the unit matrix expresses that σ maps any constant to itself.

An E-unification problem with constants from a finite set $\mathcal{C}$ corresponds to a parallel pair $\langle\sigma, \tau\rangle$ of morphisms respecting the constants in $\mathcal{C}$, and each E-unifier δ of this pair also corresponds to a morphism respecting $\mathcal{C}$. For the components of the corresponding matrices, the fact that δ is a unifier of $\langle\sigma, \tau\rangle$, i.e., that $M_\sigma \cdot M_\delta = M_\tau \cdot M_\delta$, leads to the following equations:

$$M_\sigma^h M_\delta^h \quad = \quad M_\tau^h M_\delta^h,$$

$$M_\sigma^h M_\delta^i + M_\sigma^i \quad = \quad M_\tau^h M_\delta^i + M_\tau^i.$$

The first equation is a system of homogeneous equations in S_E, whereas the second is a system of inhomogeneous equations.

From these observations one can derive the following characterization of the type "at most finitary" for unification with constants in commutative theories:[17]

5.5. THEOREM. *Let E be a commutative theory that is unitary w.r.t. elementary unification. Then E is at most finitary w.r.t. unification with constants iff the corresponding semiring S_E satisfies the following condition: for all $m, n \geq 1$, all $m \times n$-matrices M_1, M_2 over S_E, and all $u_1, u_2 \in S_E^m$, there exist finitely many $v_1, \ldots, v_k \in S_E^n$ such that*

$$\{w \in S_E^n \mid M_1 \cdot w + u_1 = M_2 \cdot w + u_2\} = \{v_i + v \mid 1 \leq i \leq k,\, v \in U(M_1, M_2)\}.$$

This conditions means that finitely many particular solutions of the system of inhomogeneous equations, $M_1 \cdot x + u_1 = M_2 \cdot x + u_2$, together with the solutions $U(M_1, M_2)$ of the corresponding system of homogeneous equations, $M_1 \cdot x = M_2 \cdot x$, generate all solutions of the system of inhomogeneous equations. The assumption that E is unitary w.r.t. elementary unification implies that $U(M_1, M_2)$ is finitely generated. The complete set of E-unifiers can now be built from the generating set of $U(M_1, M_2)$ and the finitely many particular solutions of the systems of inhomogeneous equations corresponding to the free constants as illustrated in subsection 5.1.

We close this section by mentioning some additional results on unification in commutative theories. Let E be a commutative theory.

1. For elementary unification, E is either unitary or of type zero.
2. If S_E is finite, then E is unitary for elementary unification and at most finitary for unification with constants.
3. If S_E is a ring and E is unitary for elementary unification, then E is also unitary for unification with constants.
4. If E is at most finitary for unification with constants, then E is also at most finitary for unification with linear constant restrictions, and thus also for general unification.

Proofs of these and other interesting results on unification in commutative/monoidal theories can be found in [Baader 1989*b*, Nutt 1990, Baader 1993, Baader and Nutt 1996].

Compared to syntactic approaches to unification, the semantic approach introduced here has the disadvantage that it cannot treat general unification problems directly. In fact, for a commutative theory E, we have considered the category $C_{\mathcal{F}}(E)$ for $\mathcal{F} = \mathcal{S}ig(E)$, and have used the fact that this category is semi-additive. For an extended signature $\mathcal{F}_1 \supset \mathcal{F}$, the category $C_{\mathcal{F}_1}(E)$ would no longer be semi-additive, and thus the presented approach to unification in commutative theories cannot be applied directly. For unification with constants, we have shown that one can still work within the category $C_{\mathcal{F}}(E)$ by considering special morphisms. For

[17] Recall that "at most finitary" means unitary or finitary.

arbitrary free function symbols such an approach does not appear to be possible. The general methods for combining unification algorithms described in the next section can, however, overcome this problem (see result 4. from above).

6. Combination of unification algorithms

In applications of equational unification in automated deduction, one is often faced with the problem of unifying terms containing several function symbols whose properties are defined by equational theories. For example, associative-commutative function symbols often come in pairs (e.g., the addition operation $+$ and the multiplication operation $*$ of rings). However, a given AC- or ACU-unification algorithm can only treat terms containing one of these two symbols, but not both. In program verification one may encounter data structures such as sets and lists, and their combination (e.g., sets of lists). Since union of sets ($\cup$) is associative, commutative, and idempotent, and the append operation for lists (*app*) is associative, unification of terms containing both ACI- and A-symbols is of interest in this setting. Thus, the question arises whether we can use the known $ACI_\cup$- and A_{app}-unification algorithms for unifying terms containing both $\cup$ and *app* modulo $ACI_\cup \cup A_{app}$. This is an instance of the following *combination problem for unification algorithms:*

> Assume that $E_1, \ldots, E_n$ are equational theories over pairwise disjoint signatures. How can algorithms for unification modulo E_i $(i = 1, \ldots, n)$ be combined to an algorithm for unification modulo $E_1 \cup \cdots \cup E_n$?

To be more precise, there are two variants of this problem: one can either try to combine algorithms computing complete sets of unifiers or decision procedures. It should also be noted that without the disjointness condition there cannot exist a general combination method.[18] For example, as mentioned in section 3.4, $D^l_{f,g}$-unification and $D^r_{f,g}$-unification are unitary, whereas unification modulo their union $D_{f,g}$ is infinitary, which shows that algorithms computing finite complete sets of unifiers cannot be combined in the non-disjoint case. Section 3.4 also yields a negative example for the combination of decision procedures: $D_{f,g}$-unification and A_g-unification are decidable, whereas unification modulo their union is undecidable.

The formulation of the combination problem given above is still not quite precise since it does not specify which kind of E_i-unification problems (elementary, with constants, or general) the component algorithms must be able to handle. As we shall see below, algorithms for unification with constants are not quite sufficient: the combination method requires algorithms for unification with linear constant restrictions for the component theories E_i. In particular, algorithms for general E-unification can be obtained from algorithms for E-unification with lcr by combining them with an algorithm for syntactic unification (which treats the free function symbols).

[18] There are some approaches that try to weaken the disjointness assumption, but the theories to be combined must satisfy rather strong conditions [Ringeissen 1992, Domenjoud, Klay and Ringeissen 1994].

The research on the combination problem was triggered by the search for a unification algorithm that can deal with terms containing several associative-commutative function symbols and free symbols [Stickel 1975, Stickel 1981, Fages 1984, Fages 1987, Herold and Siekmann 1987]. It turned out that the methods used in this particular instance of the combination problem can easily be generalized to other equational theories, provided that they satisfy certain restrictions (such as collapse-freeness or regularity[19]) on the syntactic form of their defining identities, which make sure that the theories behave similarly to associativity-commutativity and syntactic equality [Kirchner 1985, Tidén 1986, Herold 1986, Yelick 1987, Boudet et al. 1989].

The problem of combining algorithms computing complete sets of unifiers was solved in a very general form by Schmidt-Schauß [1989*b*]. His approach imposes no restriction on the syntactic form of the identities. The only requirements on the component theories E_i are of an algorithmic nature: both E_i-unification problems with constants and so-called "constant elimination problems" (see [Schmidt-Schauß 1989*b*] for a definition) must be finitary solvable modulo E_i. Boudet [1993] describes a more efficient combination algorithm, which depends on the same requirements as the one by Schmidt-Schauß.

In the following, we will describe the combination method introduced in [Baader and Schulz 1992, Baader and Schulz 1996] in more detail, since it can be used both for combining algorithms computing complete sets of unifiers and for combining decision procedures. Instead of splitting the algorithmic problem to be solved for the component theories E_i into two parts (unification with constants and constant elimination), this method requires algorithms (decision procedures) for E_i-unification with lcr. In this setting, Schmidt-Schauß's condition that constant elimination problems must be finitary solvable modulo E_i can be seen as just one way of ensuring that E_i-unification with lcr is at most finitary provided that E_i-unification with constants is at most finitary.

6.1. *A general combination method*

Before describing the combination method of Baader and Schulz [1992] and [1996] formally, we illustrate the underlying ideas by a simple example. Let g be a unary and f be a binary function symbol. We consider the theories A_f and $F_g := \{g(x) \approx g(x)\}$,[20] and the (elementary) unification problem

$$\Gamma_0 := \{g(f(y,y)) \overset{?}{\underset{E}{=}} g(x),\ g(x) \overset{?}{\underset{E}{=}} g(y),\ x \overset{?}{\underset{E}{=}} f(y,y)\}$$

modulo their union $E := A_f \cup F_g$. In a first step, we transform Γ_0 into an equivalent unification problem in decomposed form, i.e., into a union of an (elementary) A_f-

[19] 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.

[20] Obviously, $=_{F_g}$ is just syntactic equality. The "dummy" axiom $g(x) \approx g(x)$ makes sure that g belongs to $\mathcal{S}ig(F_g)$.

unification problem and an (elementary) F_g-unification problem:

$$\Gamma := \{z \stackrel{?}{=}_{A_f} f(y,y),\, x \stackrel{?}{=}_{A_f} f(y,y)\} \cup \{g(z) \stackrel{?}{=}_{F_g} g(x),\, g(x) \stackrel{?}{=}_{F_g} g(y)\}.$$

This has been achieved by replacing "alien" subterms (in the example, just the term $f(y,y)$ occurring on the left-hand side of the first equation) by new variables and introducing appropriate new equations (see [Baader and Schulz 1996] for a formal definition of this decomposition step).

Unfortunately, it is not sufficient simply to test the "pure" unification problems obtained this way for solvability. The problem is that these unification problems still share variables, and the single solutions may instantiate these variables with incompatible terms. For example, $\sigma_1 := \{x \mapsto f(y,y),\, z \mapsto f(y,y)\}$ solves the A_f-subproblem, and $\sigma_2 := \{x \mapsto g(x),\, y \mapsto g(x),\, z \mapsto g(x)\}$ is a solution of the F_g-subproblem, but these solutions replace both x and z by different (even non-unifiable) terms. In order to avoid such incompatible assignments, we choose a theory label for each variable: in the subproblem corresponding to this theory, the variable may be instantiated, whereas in the other subproblem the variable must be treated as a constant. For example, if we assign

$$L(x) := L(z) := A_f \quad \text{and} \quad L(y) := F_g,$$

then y must be treated as a constant in the A_f-subproblem, whereas x and z must be treated as constants in the F_g-subproblem.

This avoids incompatible instantiations of shared variables, but also leads to a new problem: in the example, the equation $g(z) =^?_{F_g} g(x)$ is no longer solvable since both z and x must be treated as (different) constants. This problem can be overcome by choosing an appropriate variable identification. In the example, x must be identified with z, which can be achieved by replacing every occurrence of z by x:

$$\Gamma' := \{x \stackrel{?}{=}_{A_f} f(y,y)\} \cup \{g(x) \stackrel{?}{=}_{F_g} g(x),\, g(x) \stackrel{?}{=}_{F_g} g(y)\}.$$

Unfortunately, the solutions $\sigma_1' := \{x \mapsto f(y,y)\}$ and $\sigma_2' := \{y \mapsto x\}$ of the pure subproblems still cannot be combined to a solution of their union, since there is a cyclic dependency between the two substitutions: x is replaced by a term containing y, and y is replaced by a term containing x. Such cyclic dependencies between solutions of the pure subproblems can finally be avoided by choosing a linear ordering on the shared variables of the unification problem, which induces linear constant restrictions for the subproblems.

These ideas can be formalized as follows. Let $E_1, \ldots, E_n$ be non-trivial equational theories over disjoint signatures. An $(E_1 \cup \cdots \cup E_n)$-unification problem Γ is in *decomposed form* iff $\Gamma = \Gamma_1 \cup \cdots \cup \Gamma_n$ where each Γ_i is an elementary E_i-unification problem. As illustrated in the example, it is easy to transform a given elementary $(E_1 \cup \cdots \cup E_n)$-unification problem into an equivalent problem in decomposed form (see [Baader and Schulz 1996] for details). Thus, we may without loss of generality assume that all our $(E_1 \cup \cdots \cup E_n)$-unification problems are in decomposed form

$\Gamma = \Gamma_1 \cup \cdots \cup \Gamma_n$. A variable occurring in Γ is called a *shared variable* iff it occurs in at least two of the pure subproblems Γ_i.

Let $\mathcal{X}$ be the set of shared variables of $\Gamma = \Gamma_1 \cup \cdots \cup \Gamma_n$. A *variable identification* can be represented by a partition $\Pi = \{P_1, \ldots, P_k\}$ of $\mathcal{X}$. For each of the classes P_i, let $x_i \in P_i$ be a representative of this class, and let $\mathcal{X}_\Pi := \{x_1, \ldots, x_k\}$ be the set of these representatives. The substitution that replaces, for all $i = 1, \ldots, k$, each element of P_i by its representative x_i is denoted by σ_Π. We denote the result of applying σ_Π to each term in Γ_i by $\Gamma_i\sigma_\Pi$. For a given partition Π of the shared variables of Γ, let $L : \mathcal{X}_\Pi \to \{1, \ldots, n\}$ be a *labelling function*, which assigns a theory label to each variable in $\mathcal{X}_\Pi$, and let $<$ be a *linear ordering* on $\mathcal{X}_\Pi$. Using L and $<$, each of the elementary E_i-unification problems $\Gamma_i\sigma_\Pi$ can be turned into an E_i-unification problem with linear constant restrictions $\langle \Gamma_i\sigma_\Pi, L, < \rangle$: the variables $x \in \mathcal{X}_\Pi$ with label $L(x) \neq i$ are treated as (free) constants in $\langle \Gamma_i\sigma_\Pi, L, < \rangle$, whereas the other variables are still treated as variables, and the linear constant restrictions are induced by $<$.[21]

6.1. PROPOSITION. *Let $\Gamma := \Gamma_1 \cup \cdots \cup \Gamma_n$ be an $(E_1 \cup \cdots \cup E_n)$-unification problem in decomposed form. 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. *There exists a partition Π, a labelling function $L : \mathcal{X}_\Pi \to \{1, \ldots, n\}$, and a linear ordering $<$ on $\mathcal{X}_\Pi$ such that, for all $i = 1, \ldots, n$, the E_i-unification problem with linear constant restrictions $\langle \Gamma_i\sigma_\Pi, L, < \rangle$ is solvable.*

Assume that solvability of E_i-unification problems with lcr is decidable for $i = 1, \ldots, n$. For a given elementary $(E_1 \cup \cdots \cup E_n)$-unification problem Γ_0 one can compute an equivalent problem in decomposed form Γ in polynomial time. For Γ, there exist only finitely many different triples $(\Pi, L, <)$, which means that it is possible to compute all possible such triples, and then test the obtained E_i-unification problems with lcr for solvability. Thus, proposition 6.1 implies that solvability of elementary $(E_1 \cup \cdots \cup E_n)$-unification problems is decidable. To be more precise, instead of deterministically computing all possible triples $(\Pi, L, <)$, one can also employ a non-deterministic algorithm that "guesses the right tuple" in polynomial time.

6.2. THEOREM. *Let $E_1, \ldots, E_n$ be non-trivial equational theories over disjoint signatures. If solvability of E_i-unification problems with linear constant restrictions is decidable (in NP) for $i = 1, \ldots, n$, then solvability of elementary $(E_1 \cup \cdots \cup E_n)$-unification problems is decidable (in NP).*

In general, it is not possible to avoid the non-determinism inherent in this combination method [Schulz 1997]. For example, the decision problem is polynomial for *ACUI*-unification with lcr, but NP-complete for general *ACUI*-unification [Baader

[21] Non-shared variables are assumed to be larger than all shared variables, i.e., there are no restrictions for the images of these variables.

and Schulz 1993*b*, Kapur and Narendran 1992*a*]. This shows that the combination of an algorithm for syntactic unification with a decision procedure for *ACUI*-unification with lcr cannot be achieved with the help of a polynomial combination method. For regular and collapse-free theories for which, in addition, it is possible to compute most general unifiers in polynomial time, one can, however, design a (deterministic) polynomial combination procedure [Schulz 1999].

The naive combination algorithm obtained by a direct application of proposition 6.1 is highly non-deterministic, and thus does not lead to satisfactory results in practice. Optimizations of the combination algorithm (which avoid this unsatisfactory behavior in many cases) are described in [Kepser and Richts 1999].

Proposition 6.1 can also be used to obtain a method for combining unification algorithms, i.e., algorithms computing finite complete sets of unifiers. In fact, as we shall see below, given solutions σ_i of the E_i-unification problems with lcr induced by the triple $(\Pi, L, <)$ can effectively be combined into a solution $\sigma_1 \odot \cdots \odot \sigma_n$ of the original $(E_1 \cup \cdots \cup E_n)$-unification problem. For a given $(E_1 \cup \cdots \cup E_n)$-unification problem Γ in decomposed form, let $T_1, \ldots, T_k$ be all the triples consisting of a partition Π, a labelling function L, and a linear ordering $<$ on $\mathcal{X}_\Pi$, and let $C_{i,j}$ be a complete set of E_i-unifiers of the E_i-unification problem with lcr induced by T_j. Then the set

$$\bigcup_{j=1}^{k} \{\sigma_1 \odot \cdots \odot \sigma_n \mid \sigma_i \in C_{i,j}\}$$

is a complete set of $(E_1 \cup \cdots \cup E_n)$-unifiers of Γ (see [Baader and Schulz 1996] for a proof).

6.3. THEOREM. *Let $E_1, \ldots, E_n$ be non-trivial equational theories over disjoint signatures that are at most finitary for E_i-unification with linear constant restrictions. Then $E_1 \cup \cdots \cup E_n$ is at most finitary for elementary unification.*

Although the combination results (as formulated in theorem 6.2 and theorem 6.3) only apply to elementary unification in the combined theory, they can easily be extended to general unification. In fact, it is easy to see that syntactic unification with lcr is decidable and unitary: just compute the mgu of the unification problem without lcr, and then test whether it satisfies the constant restrictions. Thus, one can simply take as one of the E_i's a "free" theory F such that $\mathcal{S}ig(F)$ contains all the free function symbols occurring in the general unification problem and $=_F$ is the syntactic equality on $\mathcal{S}ig(F)$-terms.

6.2. Proving correctness of the combination method

In order to show *soundness of the combination method* (i.e., (2) $\rightarrow$ (1) of proposition 6.1), it is sufficient to show that given solutions σ_i of the E_i-unification problems with lcr induced by the triple $(\Pi, L, <)$ can indeed be combined into a solution $\sigma_1 \odot \cdots \odot \sigma_n$ of the original $(E_1 \cup \cdots \cup E_n)$-unification problem in decomposed form $\Gamma = \Gamma_1 \cup \cdots \cup \Gamma_n$. First, we combine $\sigma_1, \ldots, \sigma_n$ into a solution σ of $\Gamma\sigma_\Pi = \Gamma_1\sigma_\Pi \cup \cdots \cup \Gamma_n\sigma_\Pi$. Obviously, this implies that $\sigma_\Pi\sigma$ is a solution of Γ.

Without loss of generality, we may assume that the substitution σ_i maps all variables with label i to terms containing only variables with label $j \neq i$ (which are treated as free constants in $\Gamma_i\sigma_\Pi$) or new variables, i.e., variables not occurring in Γ. The combined solution σ of $\Gamma\sigma_\Pi$ is defined along the linear ordering $<$.

Let x be the least variable with respect to $<$, and let i be its label. Since the solution σ_i of $\Gamma_i\sigma_\Pi$ satisfies the constant restrictions induced by $<$, the term $x\sigma_i$ does not contain any variables with index $j \neq i$. Thus we can simply define $x\sigma := x\sigma_i$.

Now let x be an arbitrary variable with label i, and let $y_1, \ldots, y_m$ be the variables with labels different from i occurring in $x\sigma_i$. Since σ_i satisfies the constant restrictions induced by $<$, the variables $y_1, \ldots, y_m$ (which are treated as free constants in $\Gamma_i\sigma_\Pi$) must be smaller than x. This means that $y_1\sigma, \ldots, y_m\sigma$ are already defined. The term $x\sigma$ is now obtained from $x\sigma_i$ by replacing each y_k by $y_k\sigma$ $(k = 1, \ldots, m)$.

It is easy to see that the substitution σ obtained this way satisfies $\sigma = \sigma_i\sigma$ $(i = 1, \ldots, n)$, i.e., σ is an instance of all the substitutions σ_i. Since σ_i is an E_i-unifier of $\Gamma_i\sigma_\Pi$, this implies that σ is also an E_i-unifier of $\Gamma_i\sigma_\Pi$, and thus an E-unifier of $\Gamma_i\sigma_\Pi$. Consequently, σ is an E-unifier of $\Gamma\sigma_\Pi = \Gamma_1\sigma_\Pi \cup \cdots \cup \Gamma_n\sigma_\Pi$.

Proving *completeness of the combination method* (i.e, (1) $\rightarrow$ (2) of proposition 6.1) turns out to be a bit more complex. In the following, we only give a sketch of the proof. Assume that σ is a solution of the $(E_1 \cup \cdots \cup E_n)$-unification problem in decomposed form $\Gamma = \Gamma_1 \cup \cdots \cup \Gamma_n$. This solution can be used to define the correct triple $(\Pi, L, <)$:

1. Two shared variables x, y belong to the same class of Π iff $x\sigma =_E y\sigma$.
2. If $x\sigma$ is not a variables, then $L(x) = i$ iff the top symbol of $x\sigma$ belongs to $\mathcal{S}ig(E_i)$. Otherwise, $L(x) := 1$ (this is an arbitrary decision).
3. $<$ is an arbitrary linear extension of the strict partial ordering $\prec$ defined by $x \prec y$ iff $x\sigma$ is a strict subterm of $y\sigma$.

It is easy to see that σ is also a solution of $\Gamma\sigma_\Pi = \Gamma_1\sigma_\Pi \cup \cdots \cup \Gamma_n\sigma_\Pi$. For each i, the substitution σ (which is a substitution of the combined signature $\mathcal{S}ig(E_1) \cup \cdots \cup \mathcal{S}ig(E_n)$) can be turned into a $\mathcal{S}ig(E_i)$-substitution σ_i by replacing alien subterms in $x\sigma$ (i.e., subterms starting with a symbol not belonging to $\mathcal{S}ig(E_i)$) by new variables in such a way that $=_E$-equivalent subterms are replaced by the same variable. Unfortunately, for an arbitrary E-unifier σ of Γ, the substitution σ_i obtained this way need not be a solution of the E_i-unification problem with lcr $\langle\Gamma_i\sigma_\Pi, L, <\rangle$. For this to be true, σ must be normalized in a certain way. One possibility to obtain an appropriate notion of a normalized substitution is to apply unfailing completion to the equational theory $E_1 \cup \cdots \cup E_n$, and normalize w.r.t. the ordered rewrite system R obtained this way (see [Baader and Schulz 1996] for details). Since R may be infinite, it is not necessarily possible to compute the normal form of a given term, but this is irrelevant for the proof of completeness. Another possibility (which has the advantage that normalization is effective) is to compute a so-called "layer-reduced" form [Schmidt-Schauß 1989*b*, Kirchner and Ringeissen 1994]. In principles, this normal form is obtained by applying collapse-equations as much as possible.

A different way of proving soundness and completeness of the combination method described above was introduced in [Baader and Schulz 1995*a*]: it depends on a representation of the free algebra in $V(E_1 \cup \cdots \cup E_n)$ over countably many generators as the so-called free amalgamated product of the free algebras in $V(E_i)$ in countably many generators. This approach can also deal with the combination of constraint solvers in free structures (where the signature may also contain predicate symbols), and it has been generalized to structures that are not necessarily free [Baader and Schulz 1995*c*, Baader and Schulz 1998]. The combination method has also been extended to disunification [Baader and Schulz 1995*b*, Kepser 1999].

7. Further topics

In this article we have concentrated on unification of first-order terms, and have mentioned only applications in term rewriting and resolution-based theorem proving. However, unification is a broad paradigm with applications in almost every area of automated deduction, and we would like to draw the reader's attention in particular to the two chapters of this handbook where varieties of unification not covered here are treated: higher-order unification [Dowek 2001] and rigid E-unification [Degtyarev and Voronkov 2001*a*] (Chapters 16 and 10 of this Handbook). In addition, we briefly mention in this final section a number of important variants of the unification problem that have been studied in the literature.

Matching

Given a pair of terms s, t, the matching problem asks for a substitution σ such that $s\sigma = t$. Again, this syntactic matching problem can be generalized to matching modulo an equational theory E, where one asks for a substitution σ satisfying $s\sigma =_E t$.

If t does not contain variables, then matching and unification are obviously the same problem. In general, one can turn a given matching problem into an "equivalent" unification problem by replacing the variables in t by new free constants. This transformation shows that matching modulo E can be reduced to E-unification *with constants*. Bürckert [1989] has shown that there exists an equational theory for which elementary unification is decidable, but matching and unification with constants is undecidable. Also, if one is interested in complete sets of E-matchers, then one must be careful how to define the instantiation quasi-ordering [Bürckert 1989].

Semiunification

Semiunification is a deceptively simple combination of syntactic matching and syntactic unification on first-order terms.

A *semiunification problem* consists of a set of pairs of terms

$$\{s_1 \leq^? t_1, \; \ldots, \; s_n \leq^? t_n\}$$

and is called *uniform* if $n = 1$. A substitution σ is a solution (a *semiunifier*) of such a problem iff there exist substitutions $\rho_1, \ldots, \rho_n$ such that

$$s_1\sigma\rho_1 = t_1\sigma, \; \ldots, \; s_n\sigma\rho_n = t_n\sigma.$$

This simple definition belies the broad variety of applications of semiunification in term rewriting, type checking for programming languages, proof theory, and computational linguistics; in addition, proving the properties of the problem turned out to be extremely difficult. Although it is easy to show that so-called principal solutions (analogous to *mgu*s in syntactic unification) always exist for solvable semiunification problems, the proof that the non-uniform case is undecidable is exceedingly complex; the interested reader is referred to [Kfoury, Tiuryn and Urzyczyn 1993], where a review of the results on the non-uniform case is presented.

The uniform case is decidable, but it took a long time to develop a correct, efficient algorithm. A fast algorithm based on the unification-closure method, as well as a review of the various attempts to provide algorithms for the problem, may be found in [Oliart and Snyder 1998]. This paper shows that the uniform case can be decided in $O(n^2\,\alpha(n)^2)$, where n is the size of the two input terms, and α is the functional inverse of Ackermann's function; constructing a principal solution is somewhat more complex.

Disunification
A *disunification problem* is of the form

$$\{s_1 \stackrel{?}{=} t_1, \ldots, s_n \stackrel{?}{=} t_n, s_{n+1} \stackrel{?}{\neq} t_{n+1}, \ldots, s_{n+m} \stackrel{?}{\neq} t_{n+m}\},$$

where $s_1, \ldots, t_{n+m}$ are terms. A solution of such a problem is a substitution σ satisfying $s_i\sigma = t_i\sigma$ $(i = 1, \ldots, n)$ and $s_{n+j}\sigma \neq t_{n+j}\sigma$ $(j = 1, \ldots, m)$. Again, this problem can be generalized to disunification modulo an equational theory E.

In contrast to unification, one must distinguish between different types of solvability: for disunification it makes a difference whether solutions are required to be ground substitutions (i.e., substitution introducing only variable-free terms), or whether they may be arbitrary substitutions. Both types of solvability have been considered in the literature [Colmerauer 1984, Kirchner and Lescanne 1987, Bürckert 1988, Comon and Lescanne 1989, Comon 1988, Comon 1991, Buntine and Bürckert 1994, Baader and Schulz 1993*a*], but ground solvability appears to be more interesting for most applications. It should also be noted that sometimes more general problems than the one introduced above are still called disunification problems (see, e.g., [Comon 1991]).

Sorted unification
In many applications, the domain on which the function symbols operate is not one homogeneous set: it is divided into different subsets, which on the syntactic

level are represented as sorts. Sorted unification generalizes syntactic unification in that the domain of variables is restricted to certain sorts. Unifiers are then required to be well-sorted in the sense that variables can only be replaced by terms of a "compatible" sort. Results for sorted unification strongly depend on the expressiveness of the sort language. An overview on sorted unification can, for example, be found in [Weidenbach 1998]; other important references on the topic are [Walther 1983, Walther 1987, Schmidt-Schauß 1986*a*, Schmidt-Schauß 1989*a*, Comon 1989, Meseguer, Goguen and Smolka 1989, Tommasi 1991, Frisch and Cohn 1992, Weidenbach 1996].

Bibliography

ABDULRAB H. AND PÉCUCHET J.-P. [1989], 'Solving word equations', *J. Symbolic Computation* **8**(5), 499–521.

AUFFRAY Y. AND ENJALBERT P. [1992], 'Modal theorem proving: An equational viewpoint', *J. Logic and Computation* **2**(3), 247–295.

BAADER F. [1986], 'Unification in idempotent semigroups is of type zero', *J. Automated Reasoning* **2**(3), 283–286.

BAADER F. [1989*a*], Characterizations of unification type zero, *in* N. Dershowitz, ed., 'Proceedings of the 3rd International Conference on Rewriting Techniques and Applications', Vol. 355 of *Lecture Notes in Computer Science*, Springer-Verlag, Chapel Hill, North Carolina, pp. 2–14.

BAADER F. [1989*b*], 'Unification in commutative theories', *J. Symbolic Computation* **8**(5), 479–497.

BAADER F. [1991], Unification, weak unification, upper bound, lower bound, and generalization problems, *in* R. V. Book, ed., 'Proceedings of the 4th International Conference on Rewriting Techniques and Applications', Vol. 488 of *Lecture Notes in Computer Science*, Springer-Verlag, Como, Italy, pp. 86–97.

BAADER F. [1993], 'Unification in commutative theories, Hilbert's basis theorem and Gröbner bases', *J. of the ACM* **40**(3), 477–503.

BAADER F. [1998], 'On the complexity of Boolean unification', *Information Processing Letters* **67**(4), 215–220.

BAADER F. AND BÜTTNER W. [1988], 'Unification in commutative idempotent monoids', *Theoretical Computer Science* **56**(1), 345–352.

BAADER F. AND NARENDRAN P. [1998], Unification of concept terms in description logics, *in* H. Prade, ed., 'Proceedings of the 13th European Conference on Artificial Intelligence (ECAI-98)', John Wiley & Sons Ltd, Brighton, UK, pp. 331–335.

BAADER F. AND NUTT W. [1996], 'Combination problems for commutative/monoidal theories: How algebra can help in equational reasoning', *J. Applicable Algebra in Engineering, Communication and Computing* **7**(4), 309–337.

BAADER F. AND SCHULZ K. U. [1992], Unification in the union of disjoint equational theories: Combining decision procedures, *in* D. Kapur, ed., 'Proceedings of the 11th International Conference on Automated Deduction', Vol. 607 of *Lecture Notes in Artificial Intelligence*, Springer-Verlag, Saratoga Springs, NY, USA, pp. 50–65.

BAADER F. AND SCHULZ K. U. [1993*a*], Combination techniques and decision problems for disunification, *in* C. Kirchner, ed., 'Proceedings of the 5th International Conference on Rewriting Techniques and Applications', Lecture Notes in Artificial Intelligence, Springer-Verlag, Montreal, Canada, pp. 301–315.

BAADER F. AND SCHULZ K. U. [1993*b*], General A- and AX-unification via optimized combination procedures, *in* 'Proceedings of the Second International Workshop on Word Equations and

Related Topics', Vol. 677 of *Lecture Notes in Computer Science*, Springer-Verlag, Rouen, France, pp. 23–42.

BAADER F. AND SCHULZ K. U. [1995*a*], Combination of constraint solving techniques: An algebraic point of view, *in* 'Proceedings of the 6th International Conference on Rewriting Techniques and Applications', Vol. 914 of *Lecture Notes in Artificial Intelligence*, Springer-Verlag, Kaiserslautern, Germany, pp. 352–366.

BAADER F. AND SCHULZ K. U. [1995*b*], 'Combination techniques and decision problems for disunification', *Theoretical Computer Science* **142**, 229–255.

BAADER F. AND SCHULZ K. U. [1995*c*], On the combination of symbolic constraints, solution domains, and constraint solvers, *in* 'Proceedings of the International Conference on Principles and Practice of Constraint Programming, CP95', Vol. 976 of *Lecture Notes in Artificial Intelligence*, Springer-Verlag, Cassis, France, pp. 380–397.

BAADER F. AND SCHULZ K. U. [1996], 'Unification in the union of disjoint equational theories: Combining decision procedures', *J. Symbolic Computation* **21**, 211–243.

BAADER F. AND SCHULZ K. U. [1998], 'Combination of constraint solvers for free and quasi-free structures', *Theoretical Computer Science* **192**, 107–161.

BAADER F. AND SIEKMANN J. H. [1994], Unification theory, *in* D. M. Gabbay, C. J. Hogger and J. A. Robinson, eds, 'Handbook of Logic in Artificial Intelligence and Logic Programming', Oxford University Press, Oxford, UK, pp. 41–125.

BACHMAIR L. AND GANZINGER H. [2001], Resolution theorem proving, *in* A. Robinson and A. Voronkov, eds, 'Handbook of Automated Reasoning', Vol. I, Elsevier Science, chapter 2, pp. 19–99.

BACHMAIR L., GANZINGER H., LYNCH C. AND SNYDER W. [1995], 'Basic paramodulation', *Information and Computation* **121**(2), 172–192.

BENANAV D., KAPUR D. AND NARENDRAN P. [1985], Complexity of matching problems, *in* J.-P. Jouannaud, ed., 'Proceedings of the 1st International Conference on Rewriting Techniques and Applications', Vol. 202 of *Lecture Notes in Computer Science*, Springer-Verlag, Dijon, France, pp. 417–429.

BOCKMAYR A. [1992], Algebraic and logical aspects of unification, *in* K. U. Schulz, ed., 'Proceedings of the 1st International Workshop on Word Equations and Related Topics (IWWERT '90)', Vol. 572 of *Lecture Notes in Computer Science*, Springer-Verlag, Tübingen, Germany, pp. 171–180.

BOCKMAYR A., KRISCHER S. AND WERNER A. [1992], An optimal narrowing strategy for general canonical systems, *in* 'Proceedings of the 3rd International Workshop on Conditional and Typed Term Rewriting Systems', Vol. 656 of *Lecture Notes in Computer Science*, Springer-Verlag, Pont à Mousson, France.

BOUDET A. [1993], 'Combining unification algorithms', *J. Symbolic Computation* **8**, 449–477.

BOUDET A., CONTEJEAN E. AND DEVIE H. [1990], A new AC-unification algorithm with a new algorithm for solving diophantine equations, *in* 'Proceedings of the 5th Annual IEEE Symposium on Logic in Computer Science', Philadelphia, USA, pp. 141–150.

BOUDET A., JOUANNAUD J.-P. AND SCHMIDT-SCHAUSS M. [1989], 'Unification in Boolean rings and Abelian groups', *J. Symbolic Computation* **8**, 449–477.

BUNTINE W. L. AND BÜRCKERT H.-J. [1994], 'On solving equations and disequations', *J. of the ACM* **41**(4), 591–629.

BÜRCKERT H.-J. [1988], Solving disequations in equational theories, *in* E. Lusk and R. Overbeek, eds, 'Proceedings of the 9th International Conference on Automated Deduction', Vol. 310 of *Lecture Notes in Computer Science*, Springer-Verlag, Argonne, IL.

BÜRCKERT H.-J. [1989], 'Matching—a special case of unification?', *J. Symbolic Computation* **8**(5), 532–536.

BÜRCKERT H.-J. [1991], *A Resolution Principle for a Logic with Restricted Quantifiers*, Vol. 568 of *Lecture Notes in Artificial Intelligence*, Springer-Verlag.

BÜRCKERT H.-J., HEROLD A. AND SCHMIDT-SCHAUSS M. [1989], 'On equational theories, unification, and decidability', *J. Symbolic Computation* **8**(3,4), 3–49.

BURRIS S. AND LAWRENCE J. [1990], 'Unification in commutative rings is not finitary', *Information Processing Letters* **36**(1), 37–38.

BÜTTNER W. [1986*a*], 'Unification in the data structure multiset', *J. Automated Reasoning* **2**(1), 75–88.

BÜTTNER W. [1986*b*], Unification in the data structure sets, *in* J. H. Siekmann, ed., 'Proceedings of the 8th International Conference on Automated Deduction', Vol. 230 of *Lecture Notes in Computer Science*, Springer-Verlag, Oxford, UK, pp. 470–488.

BÜTTNER W. [1988], Unification in finite algebras is unitary(?), *in* E. Lusk and R. Overbeek, eds, 'Proceedings of the 9th International Conference on Automated Deduction', Vol. 310 of *Lecture Notes in Computer Science*, Springer-Verlag, Argonne, IL, pp. 368–377.

BÜTTNER W., ESTENFELD K., SCHMID R., SCHNEIDER H.-A. AND TIDÉN E. [1990], 'Symbolic constraint handling through unification in finite algebras', *Applicable Algebra in Engineering, Communication and Computing* **1**(2), 97–119.

BÜTTNER W. AND SIMONIS H. [1987], 'Embedding Boolean expressions into logic programming', *J. Symbolic Computation* **4**(2), 191–205.

CHAMPEAUX D. [1986], 'About the Paterson-Wegman linear unification algorithm', *J. Computer and System Sciences* **32**, 79–90.

CLAUSEN M. AND FORTENBACHER A. [1989], 'Efficient solution of linear diophantine equations', *J. Symbolic Computation* **8**(1,2), 201–216.

COHN P. M. [1965], *Universal Algebra*, Harper & Row, New York.

COLMERAUER A. [1984], Equations and inequations on finite and infinite trees, *in* 'Proceedings of the International Conference on Fifth Generation Computer Systems', North Holland, Tokyo, Japan, pp. 85–99.

COMON H. [1988], Unification et Disunification: Théorie et Applications, Ph.D. thesis, Institut National Polytechnique de Grenoble, Grenoble, France.

COMON H. [1989], Inductive proofs by specification transformation, *in* N. Dershowitz, ed., 'Proceedings of the 3rd International Conference on Rewriting Techniques and Applications', Vol. 355 of *Lecture Notes in Computer Science*, Springer-Verlag, Chapel Hill, North Carolina, pp. 76–91.

COMON H. [1991], Disunification: A survey, *in* J.-L. Lassez and G. Plotkin, eds, 'Computational Logic: Essays in Honor of Alan Robinson', MIT Press, Cambridge, MA, pp. 322–359.

COMON H. AND LESCANNE P. [1989], 'Equational problems and disunification', *J. Symbolic Computation* **7**, 371–425.

CONTEJEAN E. [1993], 'Solving $*$-Problems Modulo Distributivity by a Reduction to AC1-Unification', *J. Symbolic Computation* **16**(5), 493–521.

CONTEJEAN E. AND DEVIE H. [1994], 'An efficient incremental algorithm for solving systems of linear diophantine equations', *Information and Computation* **113**, 143–172.

CORBIN J. AND BIDOIT M. [1983], A rehabilitation of Robinson's unification algorithm, *in* R. E. A. Mason, ed., 'Proceedings of the 9th World Computer Congress, IFIP'83', Elsevier, Paris, France, pp. 909–914.

DAVIS M. [1973], 'Hilbert's tenth problem is unsolvable', *American Mathematical Monthly* **80**, 233–269.

DEGTYAREV A. AND VORONKOV A. [2001], Equality reasoning in sequent-based calculi, *in* A. Robinson and A. Voronkov, eds, 'Handbook of Automated Reasoning', Vol. I, Elsevier Science, chapter 10, pp. 609–704.

DERSHOWITZ N. AND JOUANNAUD J.-P. [1990], Rewrite systems, *in* J. van Leeuwen, ed., 'Handbook of Theoretical Computer Science', Vol. B: Formal Methods and Semantics, North-Holland, Amsterdam, chapter 6, pp. 243–320.

DERSHOWITZ N. AND JOUANNAUD J.-P. [1991], 'Notations for rewriting', *Bulletin of the European Association for Theoretical Computer Science* **43**, 162–172.

DERSHOWITZ N. AND PLAISTED D. [2001], Rewriting, *in* A. Robinson and A. Voronkov, eds, 'Handbook of Automated Reasoning', Vol. I, Elsevier Science, chapter 9, pp. 533–608.

DICKSON L. E. [1913], 'Finiteness of the odd perfect and primitive abundant numbers with r distinct prime factors', *Amer. Journal of Math.* **35**, 413–422.

DOMENJOUD E. [1991], Outils pour la déduction automatique dans les théories associatives-commutatives, Thèse de Doctorat, Université de Nancy I, France.

DOMENJOUD E., KLAY F. AND RINGEISSEN C. [1994], Combination techniques for non-disjoint equational theories, *in* A. Bundy, ed., 'Proceedings of the 12th International Conference on Automated Deduction', Vol. 814 of *Lecture Notes in Artificial Intelligence*, Springer-Verlag, Nancy, France, pp. 267–281.

DOUGHERTY D. J. AND JOHANN P. [1992], 'An improved general E-unification method', *J. Symbolic Computation* **14**, 303–320.

DOWEK G. [2001], Higher-order unification and matching, *in* A. Robinson and A. Voronkov, eds, 'Handbook of Automated Reasoning', Vol. II, Elsevier Science, chapter 16, pp. 1009–1062.

DWORK C., KANELLAKIS P. AND MITCHELL J. C. [1984], 'On the sequential nature of unification', *J. Logic Programming* **1**, 35–50.

EDER E. [1985], 'Properties of substitutions and unifications', *J. Symbolic Computation* **1**, 31–46.

FAGES F. [1983], Formes Canoniques dans les Algèbres Booléennes, et Application à la Démonstration Automatique en Logique de Premier Ordre, Thèse de 3ème cycle, Université Paris VI.

FAGES F. [1984], Associative-commutative unification, *in* R. E. Shostak, ed., 'Proceedings of the 7th International Conference on Automated Deduction', Vol. 170 of *Lecture Notes in Computer Science*, Springer-Verlag, Napa, USA, pp. 194–208.

FAGES F. [1987], 'Associative-commutative unification', *J. Symbolic Computation* **3**, 257–275.

FAGES F. AND HUET G. P. [1983], Complete sets of unifiers and matchers in equational theories, *in* 'Proceedings of the 5th Colloquium on Automata, Algebra, and Programming', Vol. 159 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 205–220.

FAGES F. AND HUET G. P. [1986], 'Complete sets of unifiers and matchers in equational theories', *Theoretical Computer Science* **43**(1), 189–200.

FAY M. [1979], First-order unification in an equational theory, *in* 'Proceedings 4th Workshop on Automated Deduction', Austin, Texas, pp. 161–167.

FILGUEIRA M. AND TOMÁS A. P. [1995], 'A fast method for finding the basis of nonnegative solutions to a linear diophantine equation', *J. Symbolic Computation* **19**, 507–526.

FORTENBACHER A. [1985], An algebraic approach to unification under associativity and commutativity, *in* J.-P. Jouannaud, ed., 'Proceedings of the 1st International Conference on Rewriting Techniques and Applications', Vol. 202 of *Lecture Notes in Computer Science*, Springer-Verlag, Dijon, France, pp. 381–397.

FRANZEN M. [1992], 'Hilbert's tenth problem is of unification type zero', *J. Automated Reasoning* **9**, 169–178.

FRISCH A. M. AND COHN A. G. [1992], An abstract view of sorted unification, *in* D. Kapur, ed., 'Proceedings of the 11th International Conference on Automated Deduction', Lecture Notes in Artificial Intelligence, Springer-Verlag, Saratoga Springs, NY, USA, pp. 178–192.

GALLIER J. AND SNYDER W. [1989], 'Complete sets of transformations for general E-unification', *Theoretical Computer Science* **67**(2,3), 203–260.

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

GHILARDI S. [1997], 'Unification through projectivity', *J. Logic and Computation* **7**(6), 733–752.

GOGUEN J. A. [1989], What is unification?, *in* H. Aït-Kaci and M. Nivat, eds, 'Resolution of Equations in Algebraic Structures, Volume 1, Algebraic Techniques', Academic Press, pp. 217–261.

GRÄTZER G. [1979], *Universal Algebra, Second Edition*, Springer-Verlag, New York.

Guard J. R. [1964], Automated logic for semi-automated mathematics, Scientific Report 1, AFCRL 64-411, Air Force Cambridge Lab.

Guard J. R. [1969], 'Semi-automated mathematics', *J. of the ACM* **16**(1), 49–62.

Guo Q., Narendran P. and Shukla S. K. [1998], Unification and matching in process algebras, *in* T. Nipkow, ed., 'Proceedings of the 9th International Conference on Rewriting Techniques and Applications', Vol. 1379 of *Lecture Notes in Computer Science*, Springer-Verlag, Tsukuba, Japan, pp. 91–105.

Gutiérrez C. [1998], Solvability of word equations is in exponential space, *in* 'Proceedings of the 39th Annual IEEE Symposium on Foundations of Computer Science FOCS'98', IEEE Computer Society Press, Palo Alto, California.

Herbrand J. [1930*a*], Recherches sur la Théorie de la Démonstration, Ph.D. thesis, Sorbonne, Paris. Reprinted in W. D. Goldfarb, editor, *Logical Writings.* Reidel, 1971.

Herbrand J. [1930*b*], 'Recherches sur la théorie de la Démonstration', *Travaux de la Société des Sciences et des Lettres de Varsovie, Classe III* **33**(128).

Herbrand J. [1967], Investigations in proof theory: The properties of true propositions, *in* J. van Heijenoort, ed., 'From Frege to Gödel: A Source Book in Mathematical Logic, 1879–1931', Harvard University Press, Cambridge, MA, pp. 525–581.

Herbrand J. [1971], Recherches sur la théorie de la démonstration, *in* W. D. Goldfarb, ed., 'Logical Writings', Reidel, Dordrecht.

Hermann M. and Kolaitis P. G. [1996], Unification algorithms cannot be combined in polynomial time, *in* M. A. McRobbie and J. K. Slaney, eds, 'Proceedings of the 13th International Conference on Automated Deduction', Vol. 1104 of *Lecture Notes in Artificial Intelligence*, Springer-Verlag, pp. 246–260.

Hermann M. and Kolaitis P. G. [1997], On the complexity of unification and disunification in commutative idempotent semigroups, *in* G. Smolka, ed., 'Proceedings of the 3rd International Conference on Principles and Practice of Constraint Programming (CP'97)', Vol. 1330 of *Lecture Notes in Computer Science*, Springer-Verlag, Linz, Austria, pp. 283–297.

Herold A. [1986], Combination of unification algorithms, *in* J. H. Siekmann, ed., 'Proceedings of the 8th International Conference on Automated Deduction', Vol. 230 of *Lecture Notes in Computer Science*, Springer-Verlag, Oxford, UK, pp. 450–469.

Herold A. [1987], Combination of Unification Algorithms in Equational Theories, Ph.D. thesis, Universität Kaiserslautern, Kaiserslautern, Germany.

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

Herrlich H. and Strecker G. E. [1973], *Category Theory*, Allyn and Bacon, Boston.

Huet G. P. [1976], Résolution d'Équations dans des Langages d'ordre 1,2,...,ω, Thèse d'État, Université de Paris VII.

Huet G. P. [1978], 'An algorithm to generate the basis of solutions to homogeneous linear diophantine equations', *Information Processing Letters* **7**(3), 144–147.

Hullot J.-M. [1980], Canonical forms and unification, *in* W. Bibel and R. Kowalski, eds, 'Proceedings of the 5th International Conference on Automated Deduction', Vol. 87 of *Lecture Notes in Computer Science*, Springer-Verlag, Les Arcs, France, pp. 318–334.

Iliopoulos C. S. [1989*a*], 'Worst-case complexity bounds on algorithms for computing the canonical structure of finite Abelian groups and the Hermite and Smith normal forms of an integer matrix', *SIAM J. Computing* **18**(4), 658–669.

Iliopoulos C. S. [1989*b*], 'Worst-case complexity bounds on algorithms for computing the canonical structure of infinite Abelian groups and solving systems of linear diophantine equations', *SIAM J. Computing* **18**(4), 670–678.

Jaffar J. [1990], 'Minimal and complete word unification', *J. of the ACM* **37**(1), 47–85.

Jouannaud J.-P. and Kirchner C. [1991], Solving equations in abstract algebras: A rule-based survey of unification, *in* J.-L. Lassez and G. Plotkin, eds, 'Computational Logic: Essays in Honor of A. Robinson', MIT Press, Cambridge, MA.

JOUANNAUD J.-P. AND KIRCHNER H. [1986], 'Completion of a set of rules modulo a set of equations', *SIAM J. Computing* **15**(4), 1155–1194.

KANNAN R. AND BACHEM A. [1979], 'Algorithms for computing the Smith and Hermite normal forms of an integer matrix', *SIAM J. Computing* **8**(4), 499–507.

KAPUR D. AND NARENDRAN P. [1992*a*], 'Complexity of unification problems with associative-commutative operators', *J. Automated Reasoning* **9**, 261–288.

KAPUR D. AND NARENDRAN P. [1992*b*], Double exponential complexity of computing complete sets of AC-unifiers, *in* 'Proceedings of the 7th Annual IEEE Symposium on Logic in Computer Science', Santa Cruz, California, pp. 11–21.

KEPSER S. [1999], Negation in combining constraint systems, *in* D. Gabbay and M. de Rijke, eds, 'Frontiers of Combining Systems 2, Papers presented at FroCoS'98', Research Studies Press/Wiley, Amsterdam, pp. 117–192.

KEPSER S. AND RICHTS J. [1999], Optimisation techniques for combining constraint solvers, *in* D. Gabbay and M. de Rijke, eds, 'Frontiers of Combining Systems 2, Papers presented at FroCoS'98', Research Studies Press/Wiley, Amsterdam, pp. 193–210.

KFOURY A., TIURYN J. AND URZYCZYN P. [1993], 'The undecidability of the semi-unification problem', *Information and Computation* **102**(1), 83–101.

KIRCHNER C. [1985], Méthodes et Outils de Conception Systématique d'Algorithmes d'Unification dans les Théories Equationelles., Thèse d'État, Université de Nancy I, France.

KIRCHNER C. AND KIRCHNER H. [1989], Constrained equational reasoning, *in* 'Proceedings of the ACM-SIGSAM 1989 International Symposium on Symbolic and Algebraic Computation', ACM Press, Portland, Oregon.

KIRCHNER C. AND LESCANNE P. [1987], Solving disequations, *in* 'Proceedings of the Annual IEEE Symposium on Logic in Computer Science', Ithaca, NY, pp. 347–352.

KIRCHNER H. AND RINGEISSEN C. [1994], 'Combining symbolic constraint solvers on algebraic domains', *J. Symbolic Computation* **18**(2), 113–155.

KNUTH D. E. [1981], *The Art of Computer Programming, Vol. 2: Seminumerical Algorithms*, Computer Science and Information Processing, second edn, Addison-Wesley, Reading.

KNUTH D. E. AND BENDIX P. B. [1970], Simple word problems in universal algebras, *in* J. Leech, ed., 'Computational Problems in Abstract Algebra', Pergamon Press, Oxford.

KOSCIELSKI A. AND PACHOLSKI L. [1990], Complexity of unification in free groups and free semigroups, *in* 'Proceedings of the 31st Annual IEEE Symposium on Foundations of Computer Science', Los Alamitos, pp. 824–829.

KOZEN D. [1981], 'Positive first-order logic is NP-complete', *IBM Journal for Research and Development* **25**, 327–332.

LAMBERT J.-L. [1987], 'Une borne pour les générateurs des solutions entières positives d'une équation diophantienne linéaire', *Comptes Rendus de l'Académie des Sciences de Paris* **305**, 39–40.

LANKFORD D. S., BUTLER G. AND BRADY B. [1984], 'Abelian group unification algorithms for elementary terms', *Contemporary Mathematics* **29**, 193–199.

LASSEZ J.-L., MAHER M. AND MARIOTT K. [1987], Unification revisited, *in* J. Minker, ed., 'Foundations of Deductive Databases and Logic Programming', Morgan Kaufman, Los Altos, California, pp. 587–625.

LINCOLN P. AND CHRISTIAN J. [1989], 'Adventures in associative-commutative unification', *J. Symbolic Computation* **8**(1,2), 217–240.

LIVESEY M. AND SIEKMANN J. H. [1975], Unification of AC-terms (bags) and ACI-terms (sets), Internal report, University of Essex. Also published as Technical Report 3-76, Universität Karlsruhe, 1976.

MAKANIN G. S. [1977], 'The problem of solvability of equations in a free semigroup', *Math. Sbornik* **103**, 147–236. English translation in Math. USSR Sbornik 32, 1977.

MAL'CEV A. I. [1971], *The Metamathematics of Algebraic Systems*, Vol. 66 of *Studies in Logic and the Foundation of Mathematics*, North Holland, Amsterdam.

MAL'CEV A. I. [1973], *Algebraic Systems*, Vol. 192 of *Die Grundlehren der mathematischen Wissenschaften in Einzeldarstellungen*, Springer-Verlag, Berlin.

MARTELLI A. AND MONTANARI U. [1976], Unification in linear time and space: A structured presentation., Technical Report B76-16, University of Pisa.

MARTELLI A. AND MONTANARI U. [1982], 'An efficient unification algorithm', *ACM Transactions on Programming Languages and Systems* **4**(2), 258–282.

MARTIN U. AND NIPKOW T. [1989*a*], 'Boolean unification—The story so far', *J. Symbolic Computation* **7**(3,4), 275–293.

MARTIN U. AND NIPKOW T. [1989*b*], 'Unification in Boolean rings', *J. Automated Reasoning* **4**, 381–396.

MATIYASEVICH Y. [1971], 'Diophantine representation of recursively enumerable predicates', *Ak. Nauk USSR, Ser. Math.* **35**, 3–30.

MESEGUER J., GOGUEN J. A. AND SMOLKA G. [1989], 'Order-sorted unification', *J. Symbolic Computation* **8**, 383–413.

MOSER M. [1993], Improving transformation systems for general E-unification, *in* C. Kirchner, ed., 'Proceedings of the 5th International Conference on Rewriting Techniques and Applications', Lecture Notes in Artificial Intelligence, Springer-Verlag, Montreal, Canada, pp. 92–105.

NARENDRAN P. [1996*a*], Solving linear equations over polynomial semirings, *in* 'Proceedings of the 11th Annual IEEE Symposium on Logic in Computer Science', IEEE Computer Society Press, New Brunswick, New Jersey, pp. 466–472.

NARENDRAN P. [1996*b*], 'Unification modulo ACI+1+0', *Fundamenta Informaticae* **25**(1), 49–57.

NARENDRAN P. AND OTTO F. [1990], Some results on equational unification, *in* M. E. Stickel, ed., 'Proceedings of the 10th International Conference on Automated Deduction', Vol. 449 of *Lecture Notes in Artificial Intelligence*, Springer-Verlag, Kaiserslautern, Germany, pp. 276–291.

NIEUWENHUIS R. AND RUBIO A. [1994], AC-superposition with constraints: No AC-unifiers needed, *in* A. Bundy, ed., 'Proceedings of the 12th International Conference on Automated Deduction', Vol. 814 of *Lecture Notes in Artificial Intelligence*, Springer-Verlag, Nancy, France, pp. 545–559.

NIEUWENHUIS R. AND RUBIO A. [2001], Paramodulation-based theorem proving, *in* A. Robinson and A. Voronkov, eds, 'Handbook of Automated Reasoning', Vol. I, Elsevier Science, chapter 7, pp. 371–443.

NIPKOW T. [1990], 'Unification in primal algebras, their powers and their varieties', *J. of the ACM* **37**(1), 742–776.

NUTT W. [1990], Unification in monoidal theories, *in* M. E. Stickel, ed., 'Proceedings of the 10th International Conference on Automated Deduction', Vol. 449 of *Lecture Notes in Artificial Intelligence*, Springer-Verlag, Kaiserslautern, Germany, pp. 618–632.

NUTT W. [1991], 'The unification hierarchy is undecidable', *J. Automated Reasoning* **7**(3), 369–381.

NUTT W., RÉTY P. AND SMOLKA G. [1989], 'Basic narrowing revisited', *J. Symbolic Computation* **7**(3,4), 295–317.

OLIART A. AND SNYDER W. [1998], A fast algorithm for uniform semiunification, *in* H. Kirchner and C. Kirchner, eds, 'Proceedings of the 15th International Conference on Automated Deduction', Vol. 1421 of *Lecture Notes in Artificial Intelligence*, Springer-Verlag, Lindau, Germany, pp. 239–253.

PATERSON M. S. AND WEGMAN M. N. [1978], 'Linear unification', *J. Computer and System Sciences* **16**(2), 158–167.

PÉCUCHET J. P. [1981], Équations avec constantes et algorithme de Makanin, Thèse de doctorat, Laboratoire d'Informatique, University of Rouen.

PETERSON G. [1983], 'A technique for establishing completeness results in theorem proving with equality', *SIAM J. Computing* **12**(1), 82–100.

PETERSON G. AND STICKEL M. E. [1981], 'Complete sets of reductions for equational theories with complete unification algorithms', *J. of the ACM* **28**(2), 233–264.

PIERCE B. C. [1991], *Basic Category Theory for Computer Scientists*, MIT Press, Cambridge, Mass.

PLANDOWSKI W. [1999*a*], Satisfiability of word equations with constants is in NEXPTIME, *in* T. Leighton, ed., 'Proceedings of the Thirty-First Annual ACM Symposium on Theory of Computing (STOC'99)', ACM Press, Atlanta, Georgia.

PLANDOWSKI W. [1999*b*], Satisfiability of word equations with constants is in PSPACE, *in* P. Beame, ed., 'Proceedings of the Fortieth Annual IEEE Symposium on Foundations of Computer Science (FOCS'99)', IEEE Computer Society Press, New York City, NY.

PLOTKIN G. [1972], 'Building in equational theories', *Machine Intelligence* **7**, 73–90.

POTTIER L. [1991], Minimal solutions of linear diophantine equations: Bounds and algorithms, *in* R. V. Book, ed., 'Proceedings of the 4th International Conference on Rewriting Techniques and Applications', Vol. 488 of *Lecture Notes in Computer Science*, Springer-Verlag, Como, Italy, pp. 162–173.

PRAWITZ D. [1960], 'An improved proof procedure', *Theoria* **26**, 102–139.

RÉTY P. [1987], Improving basic narrowing techniques, *in* P. Lescanne, ed., 'Proceedings of the 2nd International Conference on Rewriting Techniques and Applications', Vol. 256 of *Lecture Notes in Computer Science*, Springer-Verlag, Bordeaux, France, pp. 216–227.

RINGEISSEN C. [1992], Unification in a combination of equational theories with shared constants and its application to primal algebras, *in* A. Voronkov, ed., 'Proceedings of the Conference on Logic Programming and Automated Reasoning', Lecture Notes in Artificial Intelligence, Springer-Verlag, St. Petersburg, Russia, pp. 261–272.

ROBINSON J. A. [1963], 'Theorem proving on the computer', *J. of the ACM* **10**(2), 163–174.

ROBINSON J. A. [1965], 'A machine oriented logic based on the resolution principle', *J. of the ACM* **12**(1), 23–41.

ROBINSON J. A. [1971], 'Computational logic: The unification computation', *Machine Intelligence* **6**, 63–72.

RYDEHEARD D. E. AND BURSTALL R. M. [1985], A categorical unification algorithm, *in* 'Proceedings of the Workshop on Category Theory and Computer Programming', Vol. 240 of *Lecture Notes in Computer Science*, Springer-Verlag, Guildford, UK, pp. 493–505.

SCHMIDT R. A. [1998], *E*-unification for subsystems of S4, *in* T. Nipkow, ed., 'Proceedings of the 9th International Conference on Rewriting Techniques and Applications', Vol. 1379 of *Lecture Notes in Computer Science*, Springer-Verlag, Tsukuba, Japan, pp. 106–120.

SCHMIDT-SCHAUSS M. [1986*a*], Unification in many-sorted equational theories, *in* 'Proceedings of the 8th International Conference on Automated Deduction', Vol. 230 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 538–552.

SCHMIDT-SCHAUSS M. [1986*b*], 'Unification under associativity and idempotence is of type nullary', *J. Automated Reasoning* **2**(3), 277–282.

SCHMIDT-SCHAUSS M. [1989*a*], *Computational Aspects of an Order Sorted Logic With Term Declarations*, Vol. 395 of *Lecture Notes in Artificial Intelligence*, Springer-Verlag.

SCHMIDT-SCHAUSS M. [1989*b*], 'Unification in a combination of arbitrary disjoint equational theories', *J. Symbolic Computation* **8**(1,2), 51–99.

SCHMIDT-SCHAUSS M. [1992], Some results for unification in distributive equational theories, Internal Report 7/92, Universität Frankfurt, Frankfurt, Germany.

SCHMIDT-SCHAUSS M. [1996*a*], An algorithm for distributive unification, *in* H. Ganzinger, ed., 'Proceedings of the 7th International Conference on Rewriting Techniques and Applications (RTA-96)', Vol. 1103 of *Lecture Notes in Computer Science*, Springer-Verlag, New Brunswick, NJ, USA, pp. 287–301.

SCHMIDT-SCHAUSS M. [1996*b*], 'Decidability of unification in the theory of one-sided distributivity and a multiplicative unit', *Journal of Symbolic Computation* **22**(3), 315–344.

SCHULZ K. U. [1992], Makanin's algorithm for word equations: Two improvements and a generalization, *in* K. U. Schulz, ed., 'Proceedings of the 1st International Workshop on Word Equations and Related Topics (IWWERT '90)', Vol. 572 of *Lecture Notes in Computer Science*, Springer-Verlag, Berlin, Germany, pp. 85–150.

SCHULZ K. U. [1993], 'Word unification and transformation of generalized equations', *J. Automated Reasoning* **11**, 149–184.

SCHULZ K. U. [1997], A criterion for intractability of E-unification with free function symbols and its relevance for combination of unification algorithms, *in* H. Comon, ed., 'Proceedings of the 8th International Conference on Rewriting Techniques and Applications', Vol. 1232 of *Lecture Notes in Computer Science*, Sitges, Spain, pp. 284–307.

SCHULZ K. U. [1999], 'Tractable and intractable instances of combination problems for unification and disunification', *J. Logic and Computation* . To appear.

SIEKMANN J. H. [1979], Unification of commutative terms, *in* 'Proceedings of the International Symposium on Symbolic and Algebraic Manipulation, EUROSAM'79', Vol. 72 of *Lecture Notes in Computer Science*, Springer-Verlag, Marseille, France, pp. 531–545.

SIEKMANN J. H. [1989], 'Unification theory: A survey', *J. Symbolic Computation* **7**(3,4), 207–274.

SIEKMANN J. H. AND SZABÓ P. [1989], 'The undecidability of the D_A-unification problem', *J. Symbolic Computation* **54**(2), 402–414.

SNYDER W. [1991], *A Proof Theory for General Unification*, Birkhäuser, Boston, Basel, Berlin.

SOCHER-AMBROSIUS R. [1994], A refined version of general E-unification, *in* A. Bundy, ed., 'Proceedings of the 12th International Conference on Automated Deduction', Vol. 814 of *Lecture Notes in Artificial Intelligence*, Springer-Verlag, Nancy, France, pp. 665–677.

STICKEL M. E. [1975], A complete unification algorithm for associative-commutative functions, *in* 'Proceedings of the 4th International Joint Conference on Artificial Intelligence', Tblisi, USSR, pp. 71–82.

STICKEL M. E. [1981], 'A unification algorithm for associative commutative functions', *J. of the ACM* **28**(3), 423–434.

SZABÓ P. [1982], Unifikationstheorie Erster Ordnung, Ph.D. thesis, Universität Karlsruhe. In German.

TIDÉN E. [1986], Unification in combinations of collapse-free theories with disjoint sets of function symbols, *in* J. H. Siekmann, ed., 'Proceedings of the 8th International Conference on Automated Deduction', Vol. 230 of *Lecture Notes in Computer Science*, Springer-Verlag, Oxford, UK, pp. 431–449.

TIDÉN E. AND ARNBORG S. [1987], 'Unification problems with one-sided distributivity', *J. Symbolic Computation* **3**(1–2), 183–202.

TOMMASI M. [1991], Automates avec tests d'égalités entre cousins germains, Master's thesis, Universié de Lille, France.

VENTURINI-ZILLI M. [1975], 'Complexity of the unification algorithm for first order expressions', *Calcolo* **12**(4), 361–371.

VOGEL E. [1978], Unification von Morphismen, Diploma thesis, Univ. Karlsruhe, Institut für Informatik I, Karlsruhe, Germany. In German.

WALTHER C. [1983], A many-sorted calculus based on resolution and paramodulation, *in* A. Bundy, ed., 'Proceedings of the 8th International Joint Conference on Artificial Intelligence', Vol. 2, Karlsruhe, Germany, pp. 882–891.

WALTHER C. [1987], *A Many-Sorted Calculus Based on Resolution and Paramodulation*, Research Notes in Artificial Intelligence, Pitman Ltd.

WEIDENBACH C. [1996], 'Unification in sort theories and its applications', *Annals of Mathematics and Artificial Intelligence* **18**(2–4), 261–293.

WEIDENBACH C. [1998], Sorted unification and tree automata, *in* W. Bibel and P. Schmidt, eds, 'Automated Deduction – A Basis for Applications, Vol. I: Foundations – Calculi and Methods', Vol. 8 of *Applied Logic Series*, Kluwer Academic Publishers, Dordrecht, NL, pp. 291–320.

YELICK K. [1987], 'Unification in combinations of collapse-free regular theories', *J. Symbolic Computation* **3**(1,2), 153–182.

Index

T

U

V

Z