An Algorithm for Distributive Unification

Manfred Schmidt-Schauß

INTERNER BERICHT 13/94

An Algorithm for Distributive Unification

Manfred Schmidt-Schauß

Fachbereich Informatik, Postfach 11 19 32,
J.W.-Goethe Universität Frankfurt,
60325 Frankfurt
Germany
e-mail: schauss@informatik.uni-frankfurt.de

Abstract
We consider unification of terms under the equational theory of two-sided distributivity D with the axioms x*(y+z) = x*y + x*z and (x+y)*z = x*z + y*z. The main result of this paper is that D-unification is decidable by giving a non-deterministic transformation algorithm. The generated unification are: an AC1-problem with linear constant restrictions and a second-order unification problem that can be transformed into a word-unification problem that can be decided using Makanin's algorithm. This solves an open problem in the field of unification.
Furthermore it is shown that the word-problem can be decided in polynomial time, hence D-matching is NP-complete.

Introduction

Unification (solving equations) in equationally defined theories has several applications in computer science, an overview and further references can be found in [Si89], [Ki89], [BS94]. Since 1982, there was interest in appying methods of unification also if distributive axioms were present. These Distributive axioms are very common in algebraic structures and every-day mathematics, for example in solving Diophantine equations, but solving equations in these structures is a hard task, for example there is no algorithm for solving Diophantine equations. P. Szabó [Sz82] has considered unification in several equational theories where some axioms are dropped (not the distributive ones), proved undecidability results for several combinations, a minimal set of axioms being two-sided distributivity plus associativity of addition. Peter Szabó left open the question of a unification algorithm for the theory D, where all axioms but the two distributive ones are dropped.

Tidén and Arnborg [TA87] investigated other combinations motivated by the occurrence of distributive axioms in a modelling of communicating processes by Bergstra and Klop [BK86]. In [TA87] there is a unification algorithm for one distributive axiom and complexity results for unification in combinations of one distributive axiom with a multiplicative unit element. In particular they showed that there is another small set of axioms, one-sided distributivity, associativity of addition and a multiplicative unit, for which unification is undecidable. The most recent results on D-unification are in [Con93], where Contejean gave a unification algorithm for the subset of product terms.

The author gave an algorithm for the theory of one distributive axiom with a multiplicative unit ([Sch93]). It turned out that the unification problem for this theory has a complexity that is comparable with the complexity of associative unification.

Another area where unification in a theory with a one-sided distributive axiom is of interest is retrieval of functions by similarity of their type [Ri90], which is closely related to unification in cartesian closed categories [NS89].

From a theoretical point of view, there is no justification for the hope that results for unification in theories can be obtained from results about theories that are defined by a subset of the axioms. However, pragmatically, there are some hints that sometimes the structure does not change too much, at least for the subsets of the Peano axioms of integer arithmetic. Furthermore, the methods developed for one theory may help in understanding unificaton in another theory. This observation is supported by the results in this paper on two-sided distributivity. Furthermore, a decision algorithm for a theory provides at least a sufficient criterion for unifiability in the theory with added axioms.

In this paper, we use the usual notions of universal algebra, the notation is consistent with [Si89, BHS89, BS94]. Nevertheless, to make the paper as self contained as possible, we repeat some notations and notions.

We consider free term-algebras over a signature, where a signature is simply a set of function symbols together with their fixed arities. An equational theory is a congruence relation on the set of terms, usually defined by a set of equational axioms. If a set of axioms E is given, we will denote the corresponding equational theory by $=_E$. A substitution σ is an endomorphism on the term algebra with $\sigma x \neq x$ for at most finitely many variables. A substitution can be represented as a finite set of pairs $\{x_1 \rightarrow t_1,\dots, x_n \rightarrow t_n\}$, and the operation on a term is the parallel replacement of the variables x_i by the term t_i. An equational system Γ consists of a set of equations to be solved, written $<s_1 = t_1,\dots,s_n = t_n>$, an E-unifier (or an E-solution) is a substitution σ such that $\sigma s_i =_E \sigma t_i$ for all i = 1,…,n. Two substitutions σ,τ are equal modulo a set of variables W, ($\sigma = \tau$ [W]) if $\sigma x = \tau x$ for all variables $x \in W$. A unifier σ of Γ is more general than a unifier τ ($\sigma \leq_E \tau$ [W] for W = VARS(Γ)) if there exists a substitution λ with $\lambda\sigma =_E \tau$ [W]. During the process of solving equational systems, we consider also equational systems extended by constraints.

A **complete** set of unifiers is a set of unifers that generates all unifers by further instantiating them. If a minimal and complete subset of such a complete set of unifiers exist, we speak also of a set of most general unifiers.

We will also consider special unification problems, called matching problems, where every equation in the problem has a ground term.

Now we consider the axioms and the equational theories that we will address in this paper. The signature contains the symbol + and *, which are binary operators. For easy readability we write them as infix symbols. Sometimes the signature is extended by the symbol 1, which is a constant. Furthermore there may be infinitely many constants, also called free constants, since they don´t occur in axioms. The theory of two-sided distributivity (D) has the two axioms:

$(x+y)*z = x*z + y*z$ and

$z*(x+y) = z*x + z*y$

We denote equality of terms in the free term algebra by $=_D$.

If the axioms for a mulitplicative unit $1*x = x*1 = x$ are also present, then we speak of the theory D1 and denote equality by $=_{D1}$.

Some known properties of D are: the cardinality of an equivalence class of $=_D$ is finite, D-unification is NP-hard [AT85] and the number of most general unifiers may be infinite [Sz82]. A complete unification algorithm based on AC1-unification for equational systems consisting only of products has been given by Contejean [Co93].

We use the axioms also as a term-rewriting system by directing the axioms above to transform terms into a "sum of products":

$(x+y)*z \rightarrow x*z + y*z$

$z*(x+y) \rightarrow z*x + z*y$

In the theory D1, we also have the rewrite rules

$x*1 \rightarrow x$ and $1*x \rightarrow x$.

Terms that can´t be further simplified are called **irreducible**. The two TRS´es are terminating, but not confluent, since there are D-equal sums of products, that are not syntactically equal, for example (a+b)*(c+d) can be simplified to the two sums of products (a*c + a*d) + (b*c + b*d) and (a*c + b*c) + (a*d + b*d). This example serves as a counterexample for some naive ideas for D-unification procedures. For example the idea to unify two terms s and t by testing all the D-variants of s and t whether they are Robinson-unifiable, has as a simple counterexample the equational system $<(a*c+x*d)+(b*c+b*d) =_D (a*c+b*c)+(x*d+b*d)>$.
Homomorphisms from the algebra of terms into the natural numbers can be used as sizes of terms, such that D-equal terms have the same value. If the value of the homomorphism on variables and constants is greater than zero, then the size of a term is always greater than the size of its proper subterms. A specialization of this measure is to count the leaves of the irreducible terms.

We also want to point to subterms of a term and do this with positions (tree addresses), i.e., words over $\mathbb{N}_0$. The notation t/π is used to denote a subterm of t at position π, and $t[\pi \rightarrow s]$ means the term that is constructed from t by replacing the subterm of t at position π with s. This notation is only meaningful, if π is a position in t.
We say a **pure product** is a term that does not contain any +´s.

The paper is structured as follows: first we give a decision algorithm for unification of terms over the algebra of 1-sums using AC1-unification with linear constant restrictions. Section 2 are devoted to clarifying the structure of free D-algebras. In section 3 we show that D-unification can be solved if unifiability w.r.t. $=_E$, a related theory, can be decided. In section 4 we give a decision algorithm for unification problems w.r.t. $=_E$, and show that it is correct.

1. Unification of 1-sums in D and D1.

In this section we will analyse the initial algebra for the theory D1 in order to construct a unification algorithm for terms over this algebra. We will call this algebra I_{D1} **algebra of 1-sums**, since every term in this algebra has a representative that consists of 1's and +'es only, elements are called **1-sums**. We assume for the purposes of this section that there are no free constants.
The theory AC1 is the theory of associativity $x*(y*z) = (x*y)*z$ and commutativity $x*y = y*x$ and a unit 1 with axioms $1*x = x*1 = x$.

1.1 Lemma. Multiplication of 1-sums is AC1.
Proof. By induction on the size of terms.

A 1-sum $s \neq_{D1} 1$ is called **prime**, if there is no nontrivial product representation $s =_{D1} s_1*s_2$. A 1-sum s' is a **factor** of s, if $s =_{D1} s'*s''$ or $s =_{D1} s''*s'$. An element in I_{D1} has a **unique factorization**, if there is only one representation as a product of prime elements, up to commutativity. Once it is shown that I_{D1} has a uniwue factorization, we can consider the algebra as a free AC1-algebra, where the free generators are the primes. The notion of a **greatest common divisor** for a free AC1-algebra is defined in the obvious way. We can use this notion also for two terms that have a unique factorization.

1.2 Lemma: Let s and t be 1-sums that have no common factor. Then s+ t is prime.
Proof. If $s + t =_{D1} r_1 * r_2$ then consider a deduction from s+t to r_1*r_2. The first toplevel deduction exhibits a common nontrivial factor of s and t. ♦

The following lemma is formulated such that it can be used in an induction proof.

1.3 Lemma. Let s_i, s, t be elements in a AC1-algebra (where multiplication is AC1), such that s_i, s, t, s_1*s, s_3*t, s_2*s, s_4*t have a unique factorization. Then the equations $s_1*s =_{AC1} s_3*t$ and $s_2*s =_{AC1} s_4*t$ imply that there exists elements r_i, i=1,…,4 and r such that r_3 and r_4 have no common factor and $s_1 =_{AC1} r_1*r_3$, s_2□$=_{AC1} r_2*r_3$, $s_3 =_{AC1} r_1*r_4$, $s_4 =_{AC1} r_2*r_4$, $s =_{AC1} r*r_4$ and $t =_{AC1} r*r_3$.
Proof. Let r_1 be the greatest common divisor of s_1 and s_3, and let r_3 and r_4 be the corresponding rest. Then $r_1*r_3*s =_{AC1} r_1*r_4*t$. The unique factorization assumption implies $s =_{AC1} r * r_4$ and t□$=_{AC1}$□$r * r_3$ for some r. Applied to the second equation in the assumption, we get $s_2*r*r_4 =_{AC1} s_4*r*r_3$. This gives $s_2*r_4 =_{AC1} s_4*r_3$, and since r_3 and r_4 have no common factor, there is an element r_2, such that $s_2 =_{AC1} r_2*r_3$ and $s_4 =_{AC1} r_2*r_4$.♦

1.4 Lemma The decomposition of a 1-sum into prime factors is unique with respect to $=_{D1}$. I.e., the algebra I_{D1} is a free AC1-algebra with the primes as free generators.
Proof. Induction on the length of a toplevel deduction and the size of terms. The base cases are trivial.
For the induction step consider the deduction from s_1*s_2 to a product. Let $s_1 =_{D1} s_{11}+ s_{12}$. Distributivity gives $s_{11}*s_2 + s_{12}*s_2$. There are two possibilities for a next top level deduction step. However, these steps are symmetric, since multiplication is commutative for 1-sums.
Let $s_{11}*s_2 =_{D1} t_2*t_1$ and $s_{12}*s_2 =_{D1} t_3*t_1$. By induction the multiset of factors is not changed, hence we can apply Lemma 1.3 and get terms r_i such that $s_{11} =_{D1} r_1*r_3$, $t_2 =_{D1} r_1*r_4$, $s_{12} =_{D1} r_2*r_3$, $t_3 =_{D1} r_2*r_4$, $s_2 =_{D1} r*r_4$, $t_1 =_{D1} r*r_3$.
Hence $(s_{11} + s_{12})*s_2 =_{D1} (r_1*r_3 + r_2*r_3)*r*r_4$ and $t_1*(t_2+ t_3) =_{D1} r*r_3*(r_1*r_4 + r_2*r_4)$. Now we compute the multiset of prime factors of the first one, this is the union of the factors of (r_1+r_2), r_3*r*r_4 , the factorization of the second is the the same. ♦

1.5 Lemma. For 1-sums s_1, s_2, t_1, t_2, we have
If $s_1 + s_2$□is prime, then
$s_1 + s_2$□ $=_{D1} t_1 + t_2 \Rightarrow s_1 =_{D1} t_1$ and s_2□ $=_{D1} t_2$
Proof. We can assume that $s_1 \neq_{D1} t_1$ or s_2□$\neq_{D1} t_2$. This means, that there must be a toplevel deduction. Thus $s_1 = s_{11}*s_{12}$ and $s_2 = s_{11}*s_{22}$, where s_{11} is not trivial. We have $s_1 + s_2 = s_{11}*(s_{12}$□+□$s_{22})$ which contradicts the assumption. ♦

Now we present a unification algorithm for multi-equations over the algebra of 1-sums. The terms that are permitted in equations are terms from eht free D1-term algebra. The algorithm is a set of nondeterministic transformation rules for systems of multi-equations and constraints:

The permitted constraints are of the form $x < y$, $x \leq y$ for variables, the corresponding transitive closure is denoted by $\leq_\Gamma$, the strict part of the ordering as $<_\Gamma$. We have constraints of the form prime(x). In the AC1-unification step, we have linear constant restrictions.
A ground substitution θ is a unifer of Γ, if it unifies every multi-equation in Γ, if for every constraint $y \leq x$: $\text{size}(\theta x) \leq \text{size}(\theta y)$, for $y < x$: $\text{size}(\theta x) < \text{size}(\theta y)$, if for a constraint prime(y) the term θy is a prime 1-sum, and if for every constraint $\gcd(x_2, x_3) = 1$, we have that θx_2 and θx_3 do not contain a common prime factor.

1.6 Algorithm (for unifying equations over the algebra of 1-sums).

Step 1: Unfold every sum.

$\{\{t[s_1 + s_2]\} \cup M\} \cup \Gamma$

□□□□□ □ □

$\{\{t[x]\} \cup M, \{x, y_1 + y_2\}, \{y_1, s_1\}, \{y_2, s_2\}\}\}\} \cup \Gamma$

where x, y_1, y_2 are a new variables.

Step 2:

The following control is necessary The rule Merge is deterministic, the other rules are non-deterministic. The rule application can terminate, if Merge is not applicable, all +-symbols are in multi-equations that are marked as prime, and those multi-equations contain at most one top-level variable, and at most one + -symbol.

Merge

$\{M_1, M_2\} \cup \Gamma$

□□ □□ □

$\{M_1 \cup M_2\} \cup \Gamma$

if M_1 and M_2 contain a common variable.

Variable replacement

$\{\{x,y,M\}\} \cup \Gamma$

□□ □□ □

$\{\{x, \{y \to x\}M\}\} \cup \{y \to x\}\Gamma$

Decomp-plus

$\{\{s_1 + s_2, t_1 + t_2\} \cup M\} \cup \Gamma$

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

$\{\{s_1 + s_2\} \cup M\}, \{s_1, t_1\}, \{s_2, t_2\}\} \cup \Gamma$

Decomp-Mix

$\{\{s_1 + s_2\} \cup M\} \cup \Gamma$

□□□□□□□□□□□□□□□□□□□□□□□□□ □

$\{\{s_1, x*x_1\}, \{s_2, x*x_2\}, \{x*y\} \cup M\}, \{y, x_1 + x_2\}\} \cup \Gamma$, $x_1 < y$, $x_2 < y$, prime(y)

where x, y, x_1, x_2 are new variables .

This rule can only be applied, if the multi-equation $\{s_1 + s_2\} \cup M$ is not marked as prime.

Identify-primes:

$\{M_1, M_2\} \cup \Gamma$

<u>□□ □□ □</u>

$\{M_1 \cup M_2\} \cup \Gamma$

if M_1 and M_2 are marked as prime

Step 3. Guess a total quasi-ordering $\leq_\Gamma$ on all the variables.

If $x < y$ is in Γ and $y \leq_\Gamma y$, then FAIL.

If x,y are in the same multi-equation, and $x <_\Gamma y$, then FAIL.

Step 4. Construct the following AC1-unification problem.

Let Γ_{AC1} consist of all multi-equations that are not marked prime. All top-level variables in multi-equations marked as primes are considered as constants.

Furthermore, there are constraints on the occurrences of constants in variables. These are called linear constant restrictions in [BS91].

If $x <_\Gamma a$, then a should not occur in σx for every unifier σ.

If the AC1-system with linear constant restrictions is unifiable, then the algorithm stops with "unifiable", otherwise with FAIL. ♦

Note that there may be AC1-equations of the form $1 = x*y*z$.

1.7 Lemma Algorithm 1.6 is a correct unification algorithm for systems of multi-equation over the algebra of 1-sums.

Proof. <u>Soundness</u>. Suppose we have a ground AC1-unifier θ_0 of the final system, satisfying the constant-restrictions. Then we have to construct a unifier for the system after step 3. The constant restrictions permit us to construct bottom up a unifier θ as follows. For a prime x with equation $x = y_1 + y_2$, we define $\theta x = \theta y_1 + \theta y_2$. Note that every prime is defined by such a sum. Since the constant restriction are generated by a total quasi-ordering, this defines a unifier θ, if we ignore the constraints. The soundness of the other rules is obvious, if we ignore the constraints. Thus we get a unifier θ of the input system.

<u>Termination</u>. Termination is non-trivial only fo step 2. The measure is: μ_1: number of +-symbols in non-prime multi-equations. μ_2: number of +-symbols, μ_3. number of multi-equations, and the ordering is lexicographically.

<u>Completeness.</u> Let θ be a solution of the input equations. Note that θ instantiates only 1-sums for variables. Consider step 2 and a multi-equation swith the terms $s_1 + s_2$, $t_1 + t_2$. If $\theta s_1 = \theta t_1$ and $\theta s_2 = \theta t_2$, then we apply Decomp-plus. Otherwise, $\theta(s_1 + s_2)$ must be equal to some nontrivial product. In this case we apply Decomp-mix. We extend θ by defining $\theta x := \gcd(\theta s_1, \theta s_2)$ and θx_1 and θx_2 as the corresponding rest. Obviously, θy is not a factor of θx_1 nor of θx_2, because of the size. Furthermore, θy is prime by Lemma 1.3. If θ makes two multi-equations equals, and these are both marked as prime, then we apply Identify-primes.

We arrive at step 3 and arrange the guessing step for the ordering, such that the ordering reflects the size(θx). This is possible and consistent with the already chosen ordering relations.

Since all the introduced prime-variables are really different and prime, there is an AC1-solution of the nonprime multi-equations, by simply putting θ into a representation where for every variable x, θx is represented as a factor of primes. This unifier respects all the constant restrictions. ♦

1.8 Theorem.
i) Algorithm 1.6 is a complete unification algorithm for constant-free D1-terms over the algebra of 1-sums.
ii) Algorithm 1.6 is a decision algorithm for unification of constant-free D1-equations.
Proof. i) follows from 1.7. Claim ii) follows, since a constant free D1-unification problem is unifiable, iff there exists a unifier that substituted 1-sums only. ♦

1.9 Theorem. Algorithm 1.6 is a non-deterministic polynomial algorithm.
Proof. Unfolding is a linear process, hence step 1 takes linear time. Decomp-plus doesn't copy terms, since after unfolding, s_i and t_i are variables. Decomp-mix increases the size at most by a constant factor times the number of +-symbols in the input. Step 3 may generate a quadratic number of pairs. The final AC1-unification problem wih linear constant restrictions can be solved in NP time (cf [BS91) ♦

1.10 Corollary. The unification problem for constant free D1-unification problems is NP-complete.
Proof. Theorem 1.9 shows that it is in NP. It is NP-hard, since every AC1-unification problem can be considered as a D1-unification problem without free constants: Given any AC1-unification problem, the constants in the AC1-unification problem can be encoded as prime 1-sums, such that different constants give different prime 1-sums. The resulting problem is a D1-unification problem withou free constants. ♦

1.11 Example: Is there a unifer of $x*x + y*y = z*z$ over the algebra of 1-sums?
We make the transformations by adding equations and replacing the sum. This gives three equations:
$x*x = x_1*x_2$, $y*y = x_1*x_3$, $z*z = x_1*v$, $v = x_2 \square + \square x_3$, prime(v), $x_2 < v$, $x_3 < v$.
The AC1-problem is: $x*x = x_1*x_2$, $y*y = x_1*x_3$, $z*z = x_1*a$, together with a linear constant restrictions, such that a does not occur in x_2 nor in x_3. Solving first $z*z = x_1*a$ gives one most general unifier: $\{z \rightarrow z_1*a, x_1 \rightarrow a*z_1*z_1\}$.
This solution applied to the other equations gives: $x*x = a*z_1 *z_1 *x_2$, $y*y = a*z_1*z_1*x_3$,
Every solution must instantiate x_2 and x_3 with a term containing some occurrences of a, which contradicts the constant restrictions.
Hence the equation $x*x + y*y = z*z$ is not unifiable over the algebra of 1-sums. ♦

2. Structure of free D-algebras

The example $(a+a)*b =_D a*(b+b)$ shows that the unification process can't use a naïve decomposition method for *, since $(a+a) \neq_D a$. However, if we could write this as $((1+1)*a)*b =_D a*((1+1)*b)$, then there is a chance for a decomposition that first decomposes into 1-sum part and rest and then uses *-decomposition.
The main goal of this section is to show that this decomposition method can be applied. As a preparation, there is the need to analyse the algebra of D-terms, where D-terms are the terms built from +,*,variables and constants. This analysis of the free D-algebra requires a further term-algebra

built from D-terms and 1-sums. For the purposes of this section we will call these 1-sums also operators. The terms in this algebra are D-terms t, products o•t of an operator term (1-sum) o with a D-term t, and sums and products. Note that there is no term t + o permitted where o is a 1-sum, and t a D-term.
Now we define an operation • of 1-sums on D-terms. We have to define the operation in a syntactic way and will then prove that on the subset of non-prime D-terms the Theory DAC1 of 1-sums and the theorey $=_D$ on D-terms are compatible. Let the operation be defined as follows:

$$(o_1+o_2)\bullet t \quad := \; o_1\bullet t + o_2\bullet t$$
$$1\bullet t \quad := t$$
$$(o_1 * o_2)\bullet t) \quad := \; o_1\bullet(o_2\bullet t)$$

Thus every expression o•t can be evaluated to a D-term (a term in the free D-algebra).
We say a D-term t is **prime**, iff $t =_D t_1 * t_2$ is impossible for D-terms t_1 and t_2.
A coherent handling of all terms forces us to introduce a new equality on D-terms, denoted by $=_E$, which is induced by the theory DAC1 for 1-terms.

2.1 Definition. The theory of equality on D-terms that is induced by the theory DAC1 for 1-terms, is called E, the corresponding equality on D-terms is denoted as $=_E$.

Obviously, we have $s =_D t \Rightarrow\; s =_E t$ for D-terms s and t, but the converse is false, see the Example 2.2 below. Furthermore, every $=_E$-equivalence class splits into at most finitely many $=_D$-equivalence classes.

2.2 Example. This example shows that there are terms s,t, such that $\neg(s =_E t \Rightarrow\; s =_D t)$, and furthermore that the cancellation rule $s*t_1 =_D s*t_2 \Rightarrow t_1 =_D t_2$ doesn't hold. This example is also the minimal example with this property.
We have $((a+a)+ ((a+a)+(a+a))) * b \;=_D ((a+(a+a)) + (a+(a+a))) * b$, which can be seen be some applications of distributive axioms, but $((a+a)+ ((a+a)+(a+a))) \;\neq_D ((a+(a+a)) + (a+(a+a)))$. However, we have $((1+1)+((1+1)+(1+1)))\bullet a =_D ((a+a)+ ((a+a)+(a+a)))$ and $(((1+1)+1) + ((1+1)+1))\bullet a = ((a+(a+a)) + (a+(a+a)))$. Since $((1+1)+((1+1)+(1+1))) =_{DAC1} (1+1)*((1+1)+1) =_{DAC1} (((1+1)+1) + ((1+1)+1))$, the equation $((a+a)+ ((a+a)+(a+a))) \;=_E ((a+(a+a)) + (a+(a+a)))$ holds.

2.3 Example. This example shows that the equality relation $=_E$ is different from $=_{D1}$ on D-terms: Multiplying out $(((a+1)*(b+1))*c)$ in two different ways gives:
$((ab)c+ac)+(bc+c) =_{D1} ((a+1)*(b+1))*c \;=_{D1}\; ((ab)c+bc)+(ac+c)$. Applying distributive axioms on the right term shows that the $=_D$-equivalence class contains exactly two terms, $((ab)c+bc)+(ac+c)$ and $((ab)+b)c+(ac+c)$. There is also no possibility to extract 1-sums. Hence $((ab)c+ac)+(bc+c) \neq_D ((a+1)*(b+1))*c$ and $((ab)c+ac)+(bc+c) \neq_E ((a+1)*(b+1))*c$.

We clarify the relationship between $=_E$ and $=_D$. The following holds:

2.4 Lemma. $o\bullet(s*t) =_D (o\bullet s)*t =_D s*(o\bullet t)$
Proof. By induction on the size of o.
If o = 1, then the Lemma holds.
Let $o = o_1+o_2$. Then $(o_1+o_2)\bullet(s*t) = o_1\bullet(s*t) + \; o_2\bullet(s*t) =_D\; (o_1\bullet s)*t + \;(o_2\bullet s)*t)$ by induction. Hence $(o_1\bullet s)*t + \;(o_2\bullet s)*t) =_D ((o_1\bullet s)+ \;(o_2\bullet s))*t \;=_D (o\bullet s)*t$.

Let $o = o_1*o_2$, then $(o_1*o_2)\bullet(s*t) = o_1\bullet(o_2\bullet(s*t))$
$=_D o_1\bullet((o_2\bullet s)*t)$ (by induction)
$=_D (o_1\bullet(o_2\bullet s))*t)$ (again by induction)
$=_D ((o_1*o_2)\bullet s)*t)$
The second equation $o\bullet(s*t) =_D s*(o\bullet t)$ is shown in a similar way. ♦

2.5 Lemma. For products s*t of D-terms, the following holds:

i) $(o_1*o_2)\bullet(s*t) =_D (o_2*o_1)\bullet(s*t)$
ii) $(o_1*(o_2+o_3))\bullet(s*t) =_D (o_1*o_2+o_1*o_3)\bullet(s*t)$
iii) $((o_1+o_2)*o_3)\bullet(s*t) =_D (o_1*o_3+o_2*o_3)\bullet(s*t)$

Proof. i) we have $(o_1*o_2)\bullet(s*t) =_D (o_1\bullet(o_2\bullet(s*t)))$ by definition. Lemma 2.3 shows that
$(o_1\bullet(o_2\bullet(s*t))) =_D o_1\bullet((o_2\bullet s)*t) =_D (o_2\bullet s)*(o_1\bullet t)$
$=_D o_2\bullet(s*(o_1\bullet t) =_D o_2\bullet(o_1\bullet(s*t))) =_D (o_2*o_1)\bullet(s*t)$
ii) $(o_1*(o_2+o_3))\bullet(s*t) =_D ((o_2\bullet s+o_3\bullet s)*(o_1\bullet t) =_D (o_2\bullet s)*(o_1\bullet t) + (o_3\bullet s)*(o_1\bullet t)$
$=_D (o_1*o_2)\bullet(s*t) + (o_1*o_3)\bullet(s*t)$
iii) follows from i) and ii). ♦

We have proved that the operation of 1-terms under AC1 is faithful for product terms:

2.6 Theorem. For products s*t, we have $s*t =_D r$ iff $s*t =_E r$

In the following we we do not differentiate between • and *, which is justified for non-primes by the Lemmas 2.4 and 2.5.

2.7 Lemma. With respect to the theory $=_E$, 1-sum factors can be pushed to toplevel:
If π is a position in t, such that $t|_\tau$ is a product for every prefix τ of π, and $t|\pi$ is a 1-sum,
then $t =_E (t|\pi) * t[\pi \rightarrow 1]$

Proof. Repeated application of Lemma 2.4. ♦

In the following we will prove a fundamental structure theorem for the theory $=_E$, namely that there exists a unique factorization. This notion is not the same as the usual one, since * is not associative, but it is a straightforward generalization.

2.8 Definition.
i) we say a term t has **no 1-sum factor** (is **1-sum free**), iff $t =_E o*s$ for a 1-sum o implies $o\square= 1$.
ii) An **E-factorization** of a term t is a representation $t =_E o*t_0$, where o is a 1-sum, and t_0 has no 1-sum factor, and either t_0 is a prime term, or $t_0 =_E t_1*t_2$, and t_1 and t_2 are E-factorizations.
iii) We say a term t has a **unique E-factorization**, if for two E-factorizations $t =_E o_1*t_{01}$ and $t =_E o_2*t_{02}$, we have $o_1 =_{DAC1} o_2$ and $t_{01} =_E t_{02}$. Furthermore for every position π of t_{01}, such that every term at a subposition τ of π in t_{01} is a product, the π is a position in t_{02}, and every term at a subposition τ of π in t_{02} is a product, and we have $t_{01}|\pi =_E t_{02}|\pi$. ♦

Now we prove that every D-term has a unique E-factorization. A similar theorem has been proved in [Co93] for a related algebra. We will give a proof, since it is different from the one in [Co93] and furthermore is an additional contribution to clarifying the structure of the algebra of D-terms.

2.9 Theorem. Every D-term has a unique E-factorization.

Proof. By induction on the size of D-terms, where size is defined by a homonorphism into the natural numbers with the usual sum and product of numbers.
First note that we can use the cancellation lemma in the inductive proofs for equations that are in the scope of the induction hypothesis, since terms that have a unique E-factorization also permit cancellation.

Let s be a non-prime term and let $s =_E s_1*s_2$ be a nontrivial factorization of s. We show by induction on the number of toplevel rewrites that the E-factorization remains the same.
If we now use induction on the lengths of derivations, it is sufficient to distinguish between toplevel rewrites and other ones, and thus we have to consider only two types of deductions, namely a deduction with two top-level applications of D_r or a deduction with a top-level application of D_l followed by a top-level application of D_r. The other cases are symmetric.
According to the lemmas above, we can denote the 1-term part of terms by O(.) and the 1-term free factor with N(.). Note that we permit in this proof that N(.) may be 1, which helps to cut down the number of cases.

a) two top-level applications of D_r.
Let $s_1 = s_{11} + s_{12}$ such that $s =_E s_{11}*s_2 + s_{12}*s_2$, and $s_{11}*s_2 =_E t_1*t_3$, $s_{12}*s_2 =_E t_2*t_3$. The resulting term is $t = (t_1+t_2)*t_3$.
By induction we get that $O(s_{11})*O(s_2) =_{DAC1} O(t_1)*O(t_3)$, $N(s_{11})*N(s_2) =_E N(t_1)*N(t_3)$, and $O(s_{12})*O(s_2) =_{DAC1} O(t_2)*O(t_3)$, $N(s_{12})*N(s_2) =_E N(t_2)*N(t_3)$.
By Lemma 1.3 there are O-terms $r_1,\ldots,r_4,r$ such that $O(s_{11}) =_{DAC1} r_1*r_3$, $O(s_2) =_{DAC1} r*r_4$, $O(t_1) =_{DAC1} r_1*r_4$, $O(t_3) =_{DAC1} r*r_3$, $O(s_{12}) =_{DAC1} r_2*r_3$, $O(t_2) =_{DAC1} r_2*r_4$.
Applied to $(s_{11}+s_{12})*s_2$ this gives: $(r_1*r_3*N(s_{11}) + r_2*r_3*N(s_{12})) * r*r_4*N(s_2) =_E$
$r*r_3*r_4* ((r_1*N(s_{11}) + r_2*N(s_{12})) *N(s_2))$.
Application to $(t_1+ t_2)*t_3$ gives: $(r_1*r_4*N(t_1) + r_2*r_4*N(t_2))*r*r_3*N(t_3) =_E$
$r*r_3*r_4* ((r_1*N(t_1) + r_2*N(t_2))*N(t_3))$.
If $r*r_3*r_4 \neq 1$ we can use induction, since $((r_1*N(s_{11}) + r_2*N(s_{12})) *N(s_2)) =_E ((r_1*N(t_1) + r_2*N(t_2))*N(t_3))$, to prove that the E-factorization of s and t is equal.
Hence we can assume that $r*r_3*r_4 =_{DAC1} 1$
In order to avoid the N's and O's, we can make the following notational simplification: we assume that the deduction is

$$(r_1*s_{11} + r_2*s_{12}) *s_2 \rightarrow (r_1*t_1 + r_2*t_2)*t_3,$$

and that by cancellation $s_{11}*s_2 =_E t_1*t_3$, $s_{12}*s_2 =_E t_2*t_3$ and that r_1 and r_2 have no common factor.
First we treat some cases where terms may be equal to 1.
$s_{11} = 1$: then $s_{12} = 1$, since we consider D-terms. Hence by cancellation $t_1 =_E t_2$, The deduction is $(r_1 + r_2) *s_2 =_E (r_1*t_1 + r_2*t_1)*t_3$, which implies that the factorization remains the same.
$t_1 = 1$: then $t_2 = 1$. The arguments are the same as above.
From now on we can assume that all the terms s_{11}, s_{12}, t_1 , t_2 are not equal to 1.
Now we have by decomposition that $s_{11} =_E t_1$, $s_2 =_E t_3$, then $s_{12} =_E t_2$, $s_2 =_E t_3$, by the cancellation rule and the deduction is then $(r_1*s_{11} + r_2*s_{12}) *s_2 =_E (r_1*s_{11} + r_2*s_{12})*s_2$, and the E-factorization of s and t is equal..

b) application of D_l followed by an application of D_r.
Let $s_1 =s_{11} + s_{12}$ such that $s = s_{11}*s_2 + s_{12}*s_2$, and $s_{11}*s_2 =_E t_1*t_2$, $s_{12}*s_2 =_E t_1*t_3$. The resulting term is $t_1*(t_2+t_3)$.
By induction we get that $O(s_{11})*O(s_2) =_{DAC1} O(t_1)*O(t_2)$, $N(s_{11})*N(s_2) =_E N(t_1)*N(t_2)$, and $O(s_{12})*O(s_2) =_{DAC1} O(t_1)*O(t_3)$, $N(s_{12})*N(s_2) =_E N(t_1)*N(t_3)$. By Lemma 1.3 there are O-terms

$r_1,\ldots,r_4,r$ such that $O(s_{11}) =_{DAC1} r_1*r_3$, $O(s_2) =_{DAC1} r*r_4$, $O(t_2) =_{DAC1} r_1*r_4$, $O(t_1) =_{DAC1} r*r_3$, $O(s_{12}) =_{DAC1} r_2*r_3$, $O(t_3) =_{DAC1} r_2*r_4$. Applied to $(s_{11}+s_{12})*s_2$ this gives:
$(r_1*r_3*N(s_{11}) + r_2*r_3*N(s_{12}))*r*r_4 *N(s_2) =_E r_3*r*r_4*(r_1*N(s_{11}) + r_2*N(s_{12})) *N(s_2)$.
Applied to $t_1*(t_2+ t_3)$ this gives: $r*r_3*N(t_1)*(r_1*r_4*N(t_2) + r_2*r_4*N(t_3)) =_E$
$r*r_3*r_4*N(t_1)*(r_1*N(t_2) + r_2*N(t_3))$
If $r*r_3*r_4 \neq 1$, then we can use induction.
If $r *r_3*r_4 =_{DAC1} 1$, then we have $O(s_2) =_{DAC1} O(t_1) =_{DAC1} 1$ and $O(s_{11}) =_{DAC1} O(t_2) =_{DAC1} r_1$, $O(s_{12}) =_{DAC1} O(t_3) =_{DAC1} r_2$.
Similar as in the case a) above, we simplify notation and assume that the deduction is

$(r_1*s_{11} + r_2*s_{12})*s_2 \to t_1*(r_1*t_2+r_2*t_3)$

and that by cancellation $s_{11}*s_2 =_E t_1*t_2$, $s_{12}*s_2 =_E t_1*t_3$ and r_1 and r_2 have no common factor.
Now consider the cases where some terms may be equal to 1.
case $s_{11} = 1$. Then $s_{12} = 1$, since we consider only D-terms. Cancellation gives $t_2 =_E t_3$, and we have $(r_1 + r_2)*s_2 \to t_1*(r_1*t_2+r_2*t_2)$, and the factorization remains the same.
case $t_2 = 1$. Then $t_1 = 1$ by construction of terms, cancellation gives $s_{11} =_E s_{12}$ and the deduction is $(r_1*s_{11} + r_2*s_{11})*s_2 \to t_1*(r_1+r_2)$, and the factorization remains the same.
Now we can assume that all the terms $s_{11}, s_{12}, s_2, t_1, t_2, t_3$ are not equal to 1. By decomposition we get $s_{11} =_E s_{12} =_E t_1$, s_2□ $=_E t_2 =_E t_3$, and the deduction is:
$(r_1*s_{11} + r_2*s_{11})*s_2 \to s_{11}*(r_1*s_2+r_2*s_2)$ and the E-factorization remains the same. ♦

2.10 Lemma. The cancellation rules hold, i.e.,

a) $r*s =_E r*t \Rightarrow s =_E t$

b) $s*r =_E t*r \Rightarrow s =_E t$

Proof. Is a consequence of Theorem 2.9. ♦

Note that the cancellation rules do not hold with respect to $=_D$, see example 2.2

2.11 Lemma. Let $s_1 + s_2$ be a D-term that is not prime, then there are two possibilities:
i) the 1-sum free parts of s_1 and s_2 are E-equal.
ii) there is a term t, such that $t =_E s_1 + s_2$, and a prime factor $p = p_1 + p_2$, such that

$t|\pi = p$, s_1□$= t[\pi \to p_1]$, $s_2 = t[\pi \to p_2]$,

Proof. Theorem 2.9 shows, that $s_1 + s_2$ has a unique factorization $o*s$. Now either the 1-term part is responsible for the sum, i.e. we have $o = o_1+o_2$, and $s_1 = o_1*s$, and $s_2 = o_2*s_2$ or the 1-sum free part is responsible for the sum, i.e., $s_1+ s_2 =_E o*s$ with $s =_E s' + s''$, and $s_1 =_E o*s'$ and $s_2 =_E o*s''$, in which case induction can be used if $o \neq 1$. If $s_1 + s_2$ is 1-sum free, then there is at least one top-level deduction to the E-factorization, and w.l.o.g.we have $s_1 =_E s_{11}*s_{12}$, $s_2 = s_{21}*s_{12}$, and $s_1+s_2 =_E (s_{11}+s_{21})*s_{12}$, and we can use the induction hypothesis for $s_{11}+s_{21}$. ♦

2.12 Lemma.
i) If a sum $s_1 + s_2$ is prime and $s_1+s_2 =_D t_1 + t_2$, then $s_1 =_D t_1$ and s_2□$=_D t_2$.
ii) If a sum $s_1 + s_2$ is prime and 1-sum free and $s_1+s_2 =_E t_1 + t_2$, then $s_1 =_E t_1$ and s_2□$=_E t_2$.
Proof. If $s_1 \neq_D t_1$ or s_2□$\neq_D t_2$, then a deduction from s_1+s_2 to $t_1 + t_2$ must have at least one top level step. But this must be an application of a distributive axiom, which shows that the sum is not prime, a contradiction.
The same argumentation applies to ii). ♦

2.13 Lemma. The O- and N-parts of a D-term can be recursively computed by the following rules:

$(s*t)_O = s_O*t_O$
$(s*t)_N = s_N*t_N$
$(s + t)_O = r_O$, where $r_O = \gcd(s_O, t_O)$ if $s_N \neq_E t_N$
$(s+t)_N = s'*s_N + t'*t_N$, where $s'r_O = s_O$ and $t'r_O = t_O$. if $s_N \neq_E t_N$
$(s + t)_O = s_O + t_O$ if $s_N =_E t_N$
$(s+t)_N = s_N = t_N$, if $s_N =_E t_N$

Proof. For products, Lemma 2.7 shows that the rules are correct.
Now consider a sum s+t, and assume that the O- and N-parts can be computed.
If $s_N =_E t_N$, then the last two equations hold.
Now assume that $s_N \neq_E t_N$. If $r_O = \gcd(s_O, t_O)$ is not trivial, then s + t has r_O as a factor in its O-part. and we can use induction on the size.
If s_O and t_O have no common 1-sum factor, and the term s + t has a nontrivial O-factor, then there must be a deduction from $s_O*s_N + t_O*t_N$ to a term with nontrivial O-factor. Since $\gcd(s_O, t_O) = 1$, the only possibility is that s_N and t_N have a common right or left factor. We can assume, that there is a common right factor. This means $s_N =_E s_{N,1}*s_{N,2}$ and $t_N =_E t_{N,1}*s_{N,2}$, and we obtain $s_O*s_N + t_O*t_N =_E (s_O*s_{N,1} + t_O*t_{N,1})*s_{N,2}$. Now induction shows, that the O-part of the first term is trivial, a contradiction to the assumption that s + t has a non-trivial O-part. Thus the rules 3 and 4 are also correct. ♦

Now we show as a corollary how the word-problem for D can be efficiently solved.

2.14 Lemma.
The computation of O- and N-part can be performed in time polynomial in the size of terms.
Proof. Let $\varphi(n)$ be the time required for computing the normal form for a term t of size n.
Then we can give an estimation of $\varphi(n)$ as follows. We assume that the O-part is always in a product of primes rerpesentation. Then:
$\varphi(n) \leq \varphi(m) + \varphi(n\text{-}m) + d*m^3$ with $n/2 \leq m \leq n\text{-}1$, where the cubic summand is the contribution of comparison of prime products and the gcd-computation.
We show by induction that $\varphi(n) \leq d*n^4$.
Since the right hand side is maximal if m = n+1, we can estimate as follows:
$\varphi(n) \leq d*(n\text{-}1)^4 + d + d*n^3 \leq dn^4$ for $n \geq 2$. Hence $\varphi(n)$ is polynomial. ♦

2.15 Lemma. Let s be a term, $s' =_D s$, and t' be a subterm of s'. Then there exists a term $t =_D t'$, such that the size of t is smaller than the size of s.
Proof. Let $s =_D s_O*s_N$ be the normal form of s. Then the size of s_O*s_N is minimal in the $=_D$ equivalence class of s. Let t' be a subterm t' of some $s' =_D s$. If $s' = s'_1*s'_2$, then t' is a subterm of s'_1 or s'_2. By decomposition theorem 2.9, we see that the normal form of s'_1 and s'_2 is smaller than s_O*s_N, hence we can use induction. If $s' = s'_1 + s'_2$, then after cancellation of the common O-parts, we have $s'' = s''_1 + s''_2$. Then Lemma 4.3 shows that there is a prime $p = p_1 + p_2$, such that s_1'' and s_2'' are constructed from the normal form of s'' by replacing p by $p_1$or p_2, respectively. Hence t' is contained as subterm in a term with a smaller normal form, and we can use induction. ♦

2.16 Theorem.
i) The word problem in D can be solved in poynomial time.
ii) D-matching is NP-complete.

Proof. Part i) follows from 2.14., since the given two terms s and t, we check D-eqaulity of s and t by computing s_O, s_N, t_O and t_N and testing $s_O = t_O$, i.e., whether the multi-set of prime factors is identical, and testing $s_N = t_N$ for syntactic equality, which is possible in polynomial time.
D-matching is in NP, since we can guess an instantiation of at most polynomial size due to Lemma 2.15, and then check equality in polynomial time. D-matching is NP-hard as shown in [TA87]. ♦

3. Reduction of D-Unification to E-Unification

The goal of this section is to show that if we have an algorithm for deciding unification problems with respect to $=_E$, then we can decide D-unifiability of equations between D-terms. By Theorem 2.6 the relation $=_D$ and $=_E$ are different only for prime terms. Hence the goal of a trasnformation algorithm is to work on the multi-equations that are prime w.r.t $=_D$ and transform them into equations between product terms.

For the input of a system of equations we can always assume that it is unfolded, i.e., every proper subterm is represented by a variable.

We use as basic datastructure labelled multiequations, i.e., multisets of terms. There are multiequations $\{s_1,\ldots,s_n\}_D$ and multiequations $\{s_1,\ldots,s_n\}_E$. Multi-equations with suffix D can have an additional label prime: $\{s_1,\ldots,s_n\}_{D,prime}$. The constraint system Γ is a set of these three types of multi-equations. A substitution σ is a **unifier** of Γ iff

i) for every multiequation $\{s_1,\ldots,s_n\}_E$ the equations $\sigma s_1 =_E \sigma s_2 =_E \ldots =_E \sigma s_n$ hold.
ii) for every multiequation $\{s_1,\ldots,s_n\}_D$ the equations $\sigma s_1 =_D \sigma s_2 =_D \ldots =_D \sigma s_n$ hold.
iii) for every multiequation $\{s_1,\ldots,s_n\}_{D,prime}$, σs_1 is a prime.

The system of multi-equations is partitioned into a part Γ_D and Γ_E.

3.1 Algorithm for D-unification using $=_E$-unification.

Input is a set of multi-equations of D-terms.
Step 1. The following rules are non-deterministically used until application is no longer possible.

Rule "Variable-replacement":
If $x,y \in M_{D,prime}$, then remove x from $M_{D,prime}$ and replace all occurrences of x by y in all terms of Γ.

Rules (Basic Rules)
"Merge": If two multiequations M and L contain the same variable, then replace them by their union. The label of the union is the union of the labels.
If the resulting label is {D,E}, then replace it by E.
If the resulting label is {D,E,prime}, then FAIL
"Occur-check". Stop with FAIL, if the following conditions are satisfied:
There is a cycle x_1, s_1,…, x_n, s_n,x_{n+1}, s_{n+1} such that s_{n+1} is identical with s_1 and x_1 is identical with x_{n+1}; x_i□and s_i are in the same multiequation and $x_{i+1} \in VARS(s_i)$ for i = 1,…,n, and at least one s_i is not a variable.

"Trivial". Delete multiequations that contain only one element.

Rule "Select-prime"

Select some multi-equation M labelled with D, but not with prime.
Choose non-deterministically one of the following possibilities:
i) change the label of M from D to E.
ii) mark the multi-equation as prime.

Rule "Process-prime"

Select a multi-equation M labelled D and prime.
Let s, t be two different nonvariable terms in M
i) If s or t is a product, then FAIL
ii) If $s = x_1 + x_2$ and $t = y_1 + y_2$, then remove s from M and add $\{x_1, y_1\}_D$ and $\{x_2, y_2\}_D$ to Γ.

Step 2. If there are remaining multi-equations labelled D, we change their label into E.
The result is a set of multi-equations with respect to $=_E$.
If this set is unifiable w.r.t. $=_E$, then return "unifiable", else FAIL. ♦

3.2 Lemma. The algorithm 3.1 is sound. I.e., if there is a unifier of the final system, then there is also a unifier of the input system.

Proof.

i) Step 1 is sound: It is sufficient to prove that every rule is sound. For proving soundness, we implicitly assume that multi-equations labelled E are also labelled as non-prime. Then Select-prime is sound. It is obvious that the rules Process-prime, Merge, and Trivial are sound. Variable-replacement is also sound, but this is nontrivial, since there are different types of equality relations. Let σ be a unifier of the system after application of $\{x \to y\}$. In order to get a unifier before the application, let σ'□be as σ, and let $\sigma' x := \sigma y$ be syntactically the same term. Since $s =_D t$ implies $s =_E t$, we have that σ'□is a unifier of the part Γ_E before the replacement. It is also a unifier of the Γ_D-part, since σx and σy are syntactically equal.

The final step 2 in algorithm 3.1 is sound:

If in step 1 of algorithm 3.1 no more rule is applicable, either all multi-equations are labelled E, and hence are non-primes, and we are ready. Assume some multi-equations labelled D and prime are in the result before step 2. All such multi-equations are of the form $\{x, y_1+y_2\}$, and this is the only occurrence of x as top-level term in a multi-equation. Now let σ be a unifier of the final result. Note that this unifier solves the equations with respect to $=_E$. Since the final system is unifiable, the rule Occur-check is not applicable to the systems before and after the label-modification. But then the variables in Γ_D can be ordered with the ordering $x > y_1$ and $x > y_2$, if $\{x, y_1+y_2\}$ is a multi-equation labelled with D and prime. We can now construct a unifier for Γ_D as follows: We start with the substitution σ_0, which is the restriction of σ to the non-prime variables. Then we construct σ_i, starting with the smallest prime variable in the ordering. We proceed bottom-up, a recursive step is that we construct a unifier σ_i for the variable x_i in the D-multi-equation $\{x_i, y_{i1}+y_{i2}\}$ such that σ_i is σ_{i-1} and in addition $\sigma_i x_i := \sigma_{i-1}(y_{i1}+y_{i2})$. ♦

3.3 Lemma. The algorithm 3.1 terminates.

Proof. The algorithm makes the following well-founded measure strictly smaller in every step. Let μ_1 be the multiset of the depths of the terms in Γ_D; μ_2 is the number of multi-equations labelled D, but not labelled prime; The measure we use is the lexicographical combination. Now Process-prime, Variable-replacment, Merge, and Trivial, make the first component strictly smaller, where Select-prime either makes μ_1 strictly smaller or leaves μ_1 invariant, and decreases μ_2, depending on the choice. ♦

3.4 Lemma Algorithm 3.1 can transform every unifiable D-unification problem into a unifiable E-unification problem.
Proof. Let σ be unifier of a system Γ during transformation by Algorithm 3.1. We implicitly assume, that multi-equations labelled E are non-prime. If the rules Merge, Trivial, or Variable-replacement are applied, then σ is also a unifier of the resulting system respecting the labels. The rule Merge cannot FAIL, since {D,E,prime}□is not possible, and the union of the labels {D,E} means that the multi-equations contains non-primes.
Using the rule Select-prime, we can assume that σs is a non-prime for terms s in multi-equations M labelled E, and that σs is a prime for terms s in multi-equations labelled D and prime. Furthermore, multi-equations labelled D, but not prime can immediately be relabelled to E or D and prime.
If the rule Process-prime is applied, then it cannot fail, since σ is a unifer, and we have assumed that the prime-labels are correct. Then σ is also a unifier of the result, since prime sums are decomposable. Let M be a multi-equation M labelled D, but not prime. If σs is a prime, then we perform alternative ii) , else the alternative i).
The final step to label all multi-equations as E leaves σ as a unifier of the final result. ♦

3.5 Theorem. If we have an algorithm for deciding unification w.r.t $=_E$, then we can construct a unification decision algorithm for D-unification.

3.6 Corollary. D-unification is decidable, iff E-unification is decidable.
Furthermore the complexity of D-unification is a non-deterministic algorithm that requires polynomial time, and calls an subalgorithm for $=_E$-unification.
Proof. If we want to decide an E-unification problem using a D-unification algorithm, we simply add a new variable y, and instead of s $=_E$ t, we try to solve s*y $=_E$ t*y. After this transformation, D-unifiability is equivalent to E-unifiability (cf. Lemma 2.10). and Theorem 2.6.
The other direction follows from Lemmas 3.2 to 3.5. The claim on the complexity holds, since Select-prime is applied at most once for every multi-equation, and the rule Process-prime removes a +-symbol on every application. ♦

4. Transformation of E-unification problems to Unification Problem of Second-Order Terms

The goal of this section is to show that E-unifiability of D-terms can be decided by transforming and decomposing the problem into an AC-unification problem and a second-order problem, which can be solved using the same methods as for the unification algorithm of stratified second order problems [Sch94]. Syntactically, the system is split into three parts: equations between 1-sums, equations that define prime sums, and associative equations.

We extend terms to second-order terms as follows. Assume there is an infinite supply of second-order monadic (function) variables, which we denote by upper-case letters like X, Y, Z. We permit the symbol Ω similar to a syntactic constant with the meaning that Ω stands for the abstracted variable in a term. For example $f(g(\Omega))$ denotes the function $\lambda x.\ f(g(x))$ mapping terms t to $f(g(t))$.

4.1 Definition. We define second order terms and parametric terms in a mutual recursive way as follows.
A **parametric term** can be:

i) Ω, the trivial parametric term,
ii) a **basic parametric term** $t*\Omega$ or $\Omega*t$, where t is a second order terms (t is the **parameter**).
iii) a concatenation of two parametric terms $p\bullet q$.
iv) A power p^n of a parametric term p, where n is a variable representing a positive natural number.

A **second order term** can be:

i) a 1-sum, or
ii) a variable or a constant,
ii) $s*t$ for second-order terms s and t,
iii) $X(t)$, where X is a second-order variable and t a second-order term,
iv) $p(t)$, where p is a parametric term and t a second-order term. ♦

The semantics of second order variables is that that they stand for a nested product, in which Ω occurs exactly once, all parameters are ground product terms, and the product is 1-sum free. These terms are also called **product contexts** (terms with one hole). Note that in second-order terms the +-symbol is permitted only as part of 1-sums.
A **second-order substitution** is a finite set of pairs (x_i,t_i) and (X_i,p_i), where t_i is a ground second-order term and p_i is a product context.
The application of a substitution σ to a second-order term is as follows: If (x,t) is a pair in σ, then all x's are replaced by t; if (X,p) is in σ, then every occurrence X(t) is replaced by $p[\Omega \rightarrow t]$.

4.2 Lemma.

i) For every substitution σ, O-terms s_O and N-terms t_N, we have $\sigma(X(s_O*s_N)) =_E \sigma(s_O*X(s_N))$
ii) For every substitution σ, we have $\sigma(X(s+t)) =_E \sigma(X(s)) + \sigma(X(t))$

Proof. Since an SO-variable X can only be instantiated with product contexts, we can push the O-part to the top-level. This proves i).
The equation ii) can be proved by successively applying distributive axioms bottom up. ♦

4.3 Lemma. Every 1-sum free ground term s_1+s_2 can be represented as r[p], where r is a product context, and $p = p_1+p_2$ is a 1-sum free prime. Furthermore $s_1 = r[p_1]$ and $s_2 = r[p_2]$.
Proof. If s_1+s_2 is a prime ground term, then r can be chosen as Ω, the trivial context.
Otherwise, we make induction on the size. If $s = t_1*t_2$, then t_1 and t_2 are 1-sum free and smaller than s, and at least one of them is equal to a sum. W.l.o.g. we can assume that t_2 is equal to a sum. By induction we get that $t_2 = r_2[p]$,where r_2 is a context. The desired context is then $t_1*r_2[\Omega]$. ♦

4.4 Definition. Let S be a set of second-order terms. For every occurrence π, the second-order-prefix of π is the string of second-order variables on a path from the root to π. The length of this prefix is also denoted as SO-depth(π), or SO-depth(x), if this is not ambiguous. A set of terms is **stratified**, if the second-order-prefixes of occurrences of first-order variables have length at most 1, the second order-prefixes of second-order variables have length 0, and if additionally, for every first order variable x, the second-order prefix of x is the same for every occurrence of x. ♦

The datastructure for the decision algorithm in this section is a set of equations, where these equations can be of three kinds:

i) equations between second order terms,

ii) equations of the form $x = s_1*t_1 + s_2*t_2$, where s_i are first-order terms over the algebra of 1-sums, and t_i are second-order terms,

iii) equations between terms over the algebra of 1-sums (cf. section 1.)

In addition there is a system of constraints of the form N(x), O(x), and prime(x) for first-order variables x, y.

We sometimes speak of an equation marked as prime (or N, O, respectively), if it contains a top-level variable x with prime(x), (or O(x), N(x), respectively). If a system Γ of equations and constraints, we denote by $N_\Gamma(x)$, $O_\Gamma(x)$, $prime_\Gamma(x)$, that N(x), O(x), or prime(x), respectively, is in Γ.

A second-order ground substitution σ is an **unconstrained unifier** of Γ, iff σ solves all equations, σx is a 1-term for $O_\Gamma(x)$, σx is 1-sum free for $N_\Gamma(x)$.

An unconstrained unifier σ of Γ is a **unifier** of Γ, if σx is a 1-sum free prime for $prime_\Gamma(x)$.

An algorithm for deciding unifiability is in general constructed as a set of non-determinstic transformations composed of rules and steps. Such a rule is called **sound**, iff $\Gamma \rightarrow \Gamma'$ and σ' is an unconstrained unifier of Γ' implies that there exists a unifier of Γ. A rule (or a set of rules) is called complete, iff in case that σ is a unifier of Γ, and the rule provides alternative transformations $\Gamma \rightarrow \Gamma_1,\ldots,\Gamma_n$, there is some i, such that Γ_i has a unifier. In general, we prove that the same unifier, perhaps with additional pairs, is a witness for soundness and/or completeness.

The predicates $O_\Gamma()$ and $N_\Gamma()$ can be extended to terms in Γ as follows: If $O_\Gamma(s)$ and $O_\Gamma(t)$, then $O_\Gamma(s*t)$ and $O_\Gamma(s+t)$. Furthermore, we have $O_\Gamma(1)$ for the constant 1 (in the algebra of 1-sums). If $N_\Gamma(s)$ and $N_\Gamma(t)$, then $N_\Gamma(s*t)$. Furthermore, we have $N_\Gamma(a)$ for all constants a in the signature. Note that $N_\Gamma(s)$ and $N_\Gamma(t)$ do not imply $N_\Gamma(s+t)$, since for example $a + a =_E (1+1)*a$.

We will use the methods and rules from [Sch94]. First we repeat some notions.

An **SO-cycle** is a set of equations $X_i(s_i) = t_i$, such that X_i occurs in $t_{i-1(\mathrm{mod}\ n)}$ and at least one occurrence is not at top level.

Since there may be cycles between SO-variables and primes, we need another type of cycle:

An **SO-prime-cycle** is a set of equations e_i, i=1,…,n, every equation is either of the form $X_i(s_i) = t_i$ or $x_i = t_i$, where x_i is a prime. Furthermore, in case i) X_i occurs in $t_{i-1(\mathrm{mod}\ n)}$ and in case ii) x_i occurs in $t_{i-1(\mathrm{mod}\ n)}$. At least one prime x_i should occur in the cycle, otherwise it is an SO-cycle.

An **SOV-cluster** is an equivalence class of SO-variables, that is maximal w.r.t to the ordering generated by the following relations: X > Y, if X(s) =t is in Γ, and Y occurs in t not at top level; X > x, if X(s) = t, and x occurs in t, and $prime_\Gamma(x)$; x > X, if x = s + t is in Γ, and X occurs in s or t.

The stratifiedness condition assures that SO-cycles, and SO-prime-cycles are of a simple structure, for example, X(x) = Y(X(z)) is not possible.

4.5 Transformation Algorithm.

Input is an arbitrary system of equations w.r.t. $=_E$. We assume that the system is unfolded, such that all sums appear at top-level.

Step 1.

In this step we first apply Split-Sums until it is no longer applicable. Then we apply the rule Select-N/O until all variables are constrained by O(.) or N(.).

Rule "Split-sums".

Replace every equation $s_1 + s_2 = t_1 + t_2$ by two equations
$x = s_1 + s_2$ and $x = t_1 + t_2$.

Rule "Select-N/O".

Replace every variable x for which no constraint N(x) or O(x) is in Γ, by x_1*x_2, and add $O(x_1)$ and $N(x_2)$, where x_1 and x_2 are new variables.

Rule "Constants are N"

Add N(a) for all free constants a in Γ.

Step 2.

Until now there are no second-order variables, and for every first order variable x, there is either a constraint N(x) or O(x) in the system Γ. We denote by the suffix O and N the corresponding O- and N-parts of the terms, which can easily be computed using lemma 2.7 and Theorem 2.9. For free constants a, we assume that the O-part is 1. We can also identify the O-part and N-part of Γ, denoted by Γ_O and Γ_N, respectively , which consist of all the equations s = t, where $O_\Gamma(s)$ and $O_\Gamma(t)$, ($N_\Gamma(s)$ and $N_\Gamma(t)$, respectively.
The goal of this step is to reduce the +-symbols, and the equations that are neither in Γ_N nor in Γ_O.

Rule "Decompose O in products"

If there is an equation s = t in $\Gamma\backslash\Gamma_O$ and s_O or t_O are not trivial,
then remove s = t from Γ, and add $s_O = t_O$ and $s_N = t_N$ to Γ.

Rule "Decompose O in sums"

If there is an equation r = s + t, such that r_O is not trivial, then we choose non-deterministically one of the following alternatives:
i) remove r = s + t from Γ,
add $r_O = s_O+t_O$, $r_N = s_N$, and $s_N = t_N$ to Γ.
ii) remove r = s + t from Γ,
Add the following equations to Γ:
$r_N = y*s_N + z*t_N$, $s_O = y*r_O$, $t_O = y*r_O$, O(y)
where y is new variable.

Rule (FAILURE)

If there is an equation a = s*t, and a is a constant, then FAIL.
If there is an equation a = b where a and b are two different constants, then FAIL.
If there is an equation a = s+ t, where a is a constant , then FAIL.

Rule (SOV-Introduction)

If s + t = r is an equation in Γ, and we have N(r), but r is not a prime variable, then
remove the equation from Γ,
add r = X(u), $u = s_O*u_1 + t_O*u_2$, prime(u), $X(u_1) = s_N$, $X(u_2) = t_N$, N(u), $N(u_1)$, $N(u_2)$.

where X is a new second-order variable, u, u_1, u_2 are new first order variables.

Step 3.

The system now contains three disjoint subsystems of equations: Γ_O, Γ_N, Γ_{prime}.
Γ_O consists of all equations that contain only O-terms
Γ_N consists of all equations that contain only N-terms and no +-symbol.
Γ_{prime} consists of all equations of the form $x = s_1*s_2+t_1*t_2$, where x is marked as prime and s_1, t_1 are O-terms, and s_2, t_2 are N-terms. Furthermore for every prime variable x, there is exactly one equation in Γ_{prime}.
In Γ_N we consider all prime-variables as constants.
The subsystem Γ_A that consists of equations between parametric terms is now empty, but will be filled during step 3.

The following control is used. Apply the rules from [Sch94] to Γ_N, until FAIL is returned, or Γ_N is empty, with the following modifications:
If a prime variable x becomes a top level variable, i.e. SO-depth(x) = 0, then apply the rule prime-identification below.
Furthermore consider all prime variables x with SO-depth(x) = 0 as constants.
The replacements of variables in Γ_N must also be done in Γ_{prime}.
Before trying to eliminate SOV-clusters , all SO-prime-cycles must all be removed.
In order to eliminate SO-prime-cycles, use the rule Split-SO-Variable on a shortest SO-prime-cycle in an analogous way to shortening SO-cycles as follows.

The base case is that there is a cycle between primes only, which leads to a FAIL using Check-pure-prime-cycles. The exits are that some SO-variable is guessed as trivial.
If there is an equation X(s) = t[x] in the cycle, where X and x contribute to the cycle, and $prime_\Gamma(x)$, then apply Split-SO-Variable and either get a shorter cycle, if X does not follow the occurrence to x, or a smaller term t'[x]. The minimal case is that X(s) = x, in which case we can apply the rules for constants.

This the overall strategy is:

i) transform Γ_N into decomposed normal form.
ii) remove SO-cycles
iii) remove SO-prime cycles
iv) remove SOV-cluster

Rule "Identify primes"

Choose one of the two alternatives
i) do nothing
ii) Let x be the prime variable, that has now SO-depth = 0, and $x = s_1*s_2$□$+ s_3*s_4$ be the equation for x in Γ_{prime}.
Select some other prime-variable y with SO-depth(y) = 0, and $y = t_1*t_2$□$+ t_3*t_4$ be the equation for y in Γ_{prime}.
Remove $x = s_1*s_2$□$+ s_3*s_4$. Replace x by y everwhere.
Add $s_1 = t_1$, $s_3 = t_3$ to Γ_O. Add $s_2 = t_2$, $s_4 = t_4$ to Γ_N.

Rule□ "Check-pure-prime-cycles".

If there is a cycle of the form $x_1 = t_1, \ldots, x_n = t_n$, where x_i are prime N-variables, and x_i occurs in t_{i-1} for all i = 2,…,n and x_1 occurs in t_n, then FAIL.

Step 4

Now the system Γ_N is empty.
In the same way as in [Sch94] transform Γ_A into an associative unification problem and check unifiability.
If this is not unifiable, then FAIL.
The part Γ_O is now checked for unifiability using Algorithm 1.6.
If it is not unifiable, then FAIL.
Otherwise, the system is unifiable. ♦

4.6 Lemma. The algorithm 4.5 terminates.
Proof. Step1 terminates, since the number of variables without constraint N(x), or O(x) is decreased.
step 2: The well-founded measure, that is strictly decreased in every rule application is as follows: We restrict the measure to equations that are not in Γ_O. μ_1 is the number of +-symbols in $\Gamma \setminus \Gamma_O$, μ_2 is the number of +-symbols in equations that are not prime, μ_3 is the number of +-symbols in equations that contain a top level term with a non-trivial O- and N-part, μ_4 is the number of toplevel products that have a nontrivial O-and N-part. The measure is the lexicographical combination $(\mu_1,\mu_2,\mu_3,\mu_4)$.
Rule "Decompose O in products" decreases μ_4.
Rule "Decompose O in sums" either decreases μ_1 or decreases μ_3.
Rule "SOV-introduction" decreases μ_2.
Step3. Termination follows from [Sch94], the additional removal of SO-prime-cycles also terminates, which can be proved in an analogous way as for SO-cycles. ♦

4.7 Lemma. The system after step 2 has the following properties: Γ can be partitioned into three different kinds of equations: Γ_O, Γ_{prime}, and Γ_N. Γ_O contains only equations between O-terms, Γ_N contains second-order terms, and is stratified, even if the N-subterms in Γ_{prime} are also taken into account. Γ_{prime} contains equations that contain one variable that is marked prime, and one sum-term of the form $s_O*s_N + t_O*t_N$.
Proof. We consider the situation, that after step2 no rule is applicable, but FAIL has not been returned. Then equations in Γ_{prime} cannot contain a constant. The rules enforce that only one variable and one sum is in such an equation. There are no +-symbols in Γ_N, since otherwise, SOV-introduction applies. There are no mixed (N-O)-terms outside of Γ_{prime}, since mixed products can be decomposed, and sums with mixed products are shifted to Γ_{prime}. The stratification property holds, since SOV-introduction adds three new variables, and all have the same second-order prefix and no rule in step 2 of algorithm 4.5 can extract these variables. ♦

4.8 Lemma. Algorithm 4.5 is sound.
Proof. This is obvious for the rules of step 1.
The rules in step 2 are sound: The only nontrivial case is the rule SOV-introduction. If σ is a unifier after application of the rule, then we have to use the assumption, that X means a product context. Hence $\sigma X(s_O*u_1 + t_O*u_2) =_E \sigma X(s_O*u_1) + \sigma X(t_O*u_2) =_E \sigma s_O*\sigma X(u_1) + \sigma t_O*\sigma X(u_2) =_E \sigma s_O*\sigma x_2 + \sigma t_O*\sigma y_2 =_E \sigma(s_O*x_2 + y_1*y_2)$.
Step 3 is sound: This follows from the corresponding lemmas in [Sch94].
Step 4 is sound: Let σ_A be a unifier of Γ_A and σ_{AC} be a unifier of Γ_O. Then we can put these together and get a common unifier σ. We have to show that σ can be extended to a unifier of Γ_{prime} as well. Since the rule SO-prime-cycle-check does not apply, we can start with the smallest prime, and for a

prime multi-equation $z = x_O*y_N + u_O*v_N$ we construct σz as $\sigma(x_O*y_N + u_O*v_N)$. Working upwards, we get a unifier of the system before step 4. ♦

4.9 Lemma. If the input system Γ is unifiable, then we can reach some final system that is unifiable.
Proof. Let θ be a unifier of the input system. For free constants we have that $\theta(a) =_E a$ has no 1-sum part. For variables x, we split θx into the 1-sum part s_O and s_N and in rule Split-N/O we extend θ such that $\theta x_1 = s_O$ and $\theta x_2 = s_N$. We have $\theta x =_E \theta(x_1*x_2)$ and the introduced constraints are satisfied.
Now consider step 2. The rule “decompose O in products” doesn't change the unifiers. Lemma 2.13 shows that if θ is a unifier of Γ before application of rule “Decompose O in sums”, then we can select an atlernative such that there exists a unifier θ' after the application. The rule “decompose-primes” doesn't change the unifers due to Lemma 2.12. The rule FAILURE doesn't apply, since the preconditions cannot be satisfied for a system Γ that has a unifier. For every sum-term that is not a prime, we can apply rules in order to satisfy the preconditions of SOV-introduction. Lemma 4.2 and Lemma 4.3 now show that we can use the rule SOV-introduction, and that the resulting system has a unifier θ' where $\theta'X$ is a product context.
In step 3, we assume that all different primes with SO-depth(x) = 0 are different under σ. Now if some prime variables x changes its SO-depth to 0, then we apply the rule “Identify primes”. If $\theta x =_E \theta y$, for some primes x and y, then we can add the decomposed equations of the equations in Γ_{prime}. The rule check-pure-prime-cycles is not applicable, since the size of the terms θx prevents such a cycle.
The rule Split-SO-Variable can be used to eliminate SO-prime-cycles analogously to the method in [Sch94] to eliminate SO-cycles.
In step 4, the systems Γ_A and Γ_O are unifiable, hence the final answer will be “unifiable”. ♦

4.10 Theorem. Algorithm 4.5 is a decision algorithm for unifiability of systems of equations w.r.t $=_E$. I.e., for every unifiable system of equations w.r.t. $=_E$, there is some sequence of applications of rules in algorithm 4.5, such that the final answer is unifiable, and for every system that is not unifiable, the answer will always be FAIL, (or not unifiable). ♦

4.11 Corollary. D-unification is decidable. The unification algorithm is non-deterministic and uses finally an AC1-unification algorithm with linear constant restriction and the decision algorithm for word equations of Makanin. ♦

Remaining **open questions** are:
i) Give better upper and lower bounds for the complexitiy of D-unification: The best known lower bound for the complexity of D-unification is that it is NP-hard (TA87).
ii) Give a unification algorithm for D1 or show undecidability of unification w.r.t. D1. Note that the technique developed in this paper does not work, since $a*(a+1) =_{D1} (a+1)*a$, but $a+1 \neq_{D1} a$.

4.12 Example. We want to demonstrate a run of the algorithm for the example $x*x + y*y =_D z*z$. Obviously, this is equivalent to $x*x + y*y =_E z*z$.
Step 1 splits variables into O and N-part:
$(x_1*x_2)* (x_1*x_2) + (y_1*y_2)*(y_1*y_2) = (z_1*z_2)*(z_1*z_2)$ where x_1, y_1, z_1 are O-variables and x_2, y_2, z_2 are N-variables.
Computation of O- and N-part gives the problem:

$(x_1*x_1)*(x_2*x_2) + (y_1*y_1)*(y_2*y_2) = (z_1*z_1)*(z_2*z_2)$

We apply “decompose O in sums”. There are two alternatives:

i) $x_1*x_1 + y_1*y_1 = z_1*z_1$, $x_2*x_2 = y_2*y_2$, $y_2*y_2 = z_2*z_2$.

Example 1.11 shows, that the Γ_O is not unifiable over the algebra of 1-sums.

Hence we can stop here.

ii) $(x_3*x_3)*(x_2*x_2) + (y_3*y_3)*(y_2*y_2) = z_2*z_2$, $x_3*x_3*z_1*z_1 = x_1*x_1$, $y_3*y_3*z_1*z_1 = y_1*y_1$.

SOV-Introduction gives:

$X(u) = z_2*z_2$, $u = (x_3*x_3)*u_1 + (y_3*y_3)*u_2$, $X(u_1) = x_2*x_2$, $X(u_2) = y_2*y_2$, …

We apply elimination of the SOV-cluster {X}

There are two alternatives

i): $X = \Omega$. This leads to FAIL, since $u = z_2*z_2$ is not unifiable, if u is considered as constant.

ii) $X = z_2*X'(\Omega)$ or $X = X'(\Omega)*z_2$.

Replacement of X gives:

$z_2*X'(u) = z_2*z_2$, $u = (x_3*x_3)*u_1 + (y_3*y_3)*u_2$, $z_2*X'(u_1) = x_2*x_2$, $z_2*X'(u_2) = y_2*y_2$, …

Decomposition and Variable-Replacement reduces this to

$X'(u) = z_2$, $u = (x_3*x_3)*u_1 + (y_3*y_3)*u_2$, $X'(u_1) = z_2$, $X'(u_2) = z_2$, …

Further Varaiable-Replacement gives the equations

$u = (x_3*x_3)*u_1 + (y_3*y_3)*u_2$, $X'(u_1) = X'(u)$, $X'(u_2) = X'(u)$, …

Now we can eliminate X' by decomposition, which gives

$u = (x_3*x_3)*u_1 + (y_3*y_3)*u_2$, $u_1 = u$, $u_2 = u$, …

Now we have opportunity to apply Prime-Identification, however, there is only one prime.

Variable-Replacement gives

$u = (x_3*x_3)*u + (y_3*y_3)*u$, …

This leads to a FAIL by the rule Check-SO-prime-cycles.

We can conclude that the equations $x*x + y*y =_D z*z$ is not unifiable. ♦

5. D-Unification of terms together with linear constant restrictions.

In order to use the decision algorithm for D-unification in a disjoint combination of equational theories, we have to construct an extension that also respects an additional arbitrary linear constant restriction as input ([BS92, Sch89]).

This is also related to the open question, whether for every equational theory with a decidable unification problem the corresponding unification problem with constant restrictions is also decidable. Since D-unification has a high complexity, there is a slight chance that it is a counterexample. However, we will show that D-unification has a normal beahviour: D-unification with linear constant restriction is decidable.

We do not do this in a formal way. We will inspect all steps of the D-unification algorithm given in this paper and note the necessary modification.

A linear constant restriction in the input is a chain $a_1 \leq x_1 \leq \ldots\ a_n \leq x_n$, where ≤ and < may occur in the chain. A unifier σ must respect the restriction that if $x < a$, then a does not occur in σx. If $x \leq a$, then either $\sigma x = a$, or a does not occur in x.

We can easily restrict this problem to chains, where only < occurs by a preparating guessing step, that either guesses $x = a$, or $a \notin x$.

Hence we can concentrate on a set of conditions $a \notin x$, meaning that a does not occur in σx.

1) The unification of 1-sums can remain unchanged, since constants do not interfere with the subproblem of unifying 1-sums.
2) Consider the reduction of D-unification to E-unification.
  There is the need of a check-rule that tests all equations and constant restrictions:
    If there is an equation $x = s$, a occurs in s, and $a \notin x$ is a constant restriction,
    then FAIL
  In case of a Variable-Replacement $x = y$, the constant restrictions of x and y are joined.
  If $x = s$ is an equation, and $a \notin x$ is a constant restriction, then for every variable y in $Var(s)$, we add $a \notin y$.

This shows that the transformation of D-unification to E-unification respects the constant restrictions.

3) Transformation to a second order-problem
  The same check as above will be applied.
    If there is an equation $x = s$, a occurs in s, and $a \notin x$ is a constant restriction,
    then FAIL
    If there is an equation $X(\Omega) = s(\Omega)$, a occurs in s, and $a \notin X$ is a constant restriction,
    then FAIL
  The inheritance of constant restrictions is slightly different:
  If $x = s$ is an equation, $a \notin x$ a constant restriction, and y a variable in s with $N_\Gamma(y)$,
    then add $a \notin y$.
  If $x = s$ is an equation, $a \notin x$ a constant restriction, and Y an SO-variable in s ,
    then add $a \notin Y$.
  We apply the same inheritance, if replacements of variables are replaced, since such a replacement can be viewed as an intermediary introduction of an equation, a replacement, and then a removal.
  The final associative equations and their solutions do not instantiate variables in the problem, rather they instantiate the integer variables in the exponents, hence this step has nothing to check.

Hence we have:
5.1 Theorem. Unifiability of equations together with linear constant restrictions w.r.t. the theory D is decidable.

Acknowledgements

I would like to thank Hubert Comon for a discussion on second-order systems, and Evelyne Contejean for discussions on D-unification.

Conclusion.

We have provided a decision algorithm for unifiability of equations under the theory of two-sided distributivity, answering a question that was open for some years. The description of the algorithm is rather complex, and requires unifying AC1-equations with linear constant restrictions, a certain kind of second order equations, and unifying word quations.
Though this algorithm may be impractical, it demonstrates the power of the tools developed in the field of unification algorithms. Furthermore it provides a theory generated by a subset of the Peano-axioms, which has a decidable unification problem.

The efficient algorithm for deciding the word problem, and the NP-completeness of D-matching may have some practical usage in the applications mentioned in the introduction.

References.

BS91 Baader, F., Schulz, K.U., General A- and AX-Unification via Optimized Combination Procedures, Proceedings of the Second International Workshop on Word Equations and Related Topics 1991, LNCS 677, pp. 23-42, (1992)

BS92 Baader, F., Schulz, K.U., Unification in the union of disjoint equational theories: Combining decision procedures, Proc. 11th CADE, LNCS 607, Springer-Verlag, pp. 50-65, (1992)

BS93 Baader, F, Siekmann, J., Unification Theory, in D.M. Gabbay, C.J. Hogger, J.A: Robinson (eds.), Handbook of Logic in Artifiial Intelligence and Logic Programming, Oxford University Press, (1994)

BHS89 Bürckert, H.-J., Herold, A., Schmidt-Schauß, M., On Equational Theories, Unification and (Un)Decidability, J. Symbolic Computation 8, pp. 3-49, 1989

Con93 Contejean, E., Solving *-problems modulo Distributivity by a reduction to AC1-Unification, J. of Symbolic Computation 16, pp. 493-521. (1993)

Fa88 Farmer, W., A unification algorithm for second order monadic terms, Annals of Pure and Applied Logic, 39, pp. 131-174, (1988)

Fa91 Farmer, W., A., Simple second-order languages for which unification is undecidable. Theoretical Computer Science 87, pp. 173-214, (1991)

Gol81 Goldfarb, W. The undecidability of the second-order unification problem, J. TCS 13, pp. 225-230, (1981)

Jaf90 Jaffar, J., Minimal and complete word unification , J. of the ACM, 37, 47-85, (1990)

KN92 Kapur, Narendran, Complexity of AC1-unification, JAR 92: pp 261-288, (1992)

KiC89 Kirchner, C. (ed.), Unification, reprint from J. of Symbolic Computation, Academic Press, (1989)

Ma77 Makanin G.S:, The problem of solvability od equations in a free semigroup. Math. Sbornik, 103, 147-236, (1977), english translation in Math USSR Sbornik 32, (1977)

Sch89 Schmidt-Schauß, M., Unification in a combination of arbitrary disjoint equational theories, J. Symbolic computation 8, 51-99, (1989)

Sch92 Schmidt-Schauß, M., Some results on unification in distributive equational theories, Internal report 7/92. J.W. Goethe-Universität, Frankfurt am Main, (1992)

Sch93 Schmidt-Schauß, M., Unification under one-sided distributivity with a multiplicative unit, Proc. LPAR 93, Springer LNCS 698, pp. 289-300, (1993)

Sch94 Schmidt-Schauß, M., Unification of Stratified Second-Order Terms, Internal report 12/94, Fachbereich Informatik, Johann Wolfang Goethe-Universität Frankfurt, (1994)

Su93 Schulz, K.U., Word unification and transformation of generalized equations, J. Automated reasoning 11, pp. 149-184, (1993)

Si89 Siekmann, J., Unification theory: a survey, in C. Kirchner (ed.), Special issue on unification, Journal of symbolic computation 7, (1989)

Sz82 Szabó, P., Unifikationstheorie erster Ordnung, Dissertation, Karlsruhe, (1982)

TA87 Tiden, E., Arnborg, S., Unification problems with one-sided distributivity, J. Symbolic Computation 3, pp. 183-202, (1987)