Abstract Congruence Closure*

Leo Bachmair[1], Ashish Tiwari[1], Laurent Vigneron[2]

[1] Department of Computer Science
State University of New York
Stony Brook, NY 11794-4400, U.S.A
{leo,astiwari}@cs.sunysb.edu
[2] LORIA – Université Nancy 2
Campus Scientifique, B.P. 239
54506 Vandœuvre-lès-Nancy Cedex, France
vigneron@loria.fr

Abstract. We describe the concept of an *abstract congruence closure* and provide equational inference rules for its construction. The length of any maximal derivation using these inference rules for constructing an abstract congruence closure is at most quadratic in the input size. The framework is used to describe the logical aspects of some well-known algorithms for congruence closure. It is also used to obtain an efficient implementation of congruence closure. We present experimental results that illustrate the relative differences in performance of the different algorithms. The notion is extended to handle associative and commutative function symbols, thus providing the concept of an *associative-commutative congruence closure*. Congruence closure (modulo associativity and commutativity) can be used to construct ground convergent rewrite systems corresponding to a set of ground equations (containing AC symbols).

1 Introduction

Term rewriting systems provide a simple and very general mechanism for computing with equations. The Knuth-Bendix completion method and its extensions to equational term rewriting systems can be used on a variety of problems. However, completion based methods yield semi-decision procedures usually, and in the few cases where they provide decision procedures, the time complexity is considerably worse than certain other efficient algorithms for solving the same problem. On the other hand, the specialized decision algorithms for particular problems are not very useful when considered for integration with general-purpose theorem proving systems. Moreover, the logical aspects inherent in the problem and the algorithm seem to get lost in descriptions of specific algorithms.

We are interested in developing *efficient* procedures for a large class of decidable problems using standard and general techniques from theorem proving so as

* The research described in this paper was supported in part by the National Science Foundation under grant CCR-9902031. Some of the results described in this paper also appeared in [5, 4].

to bridge the gap alluded to above. We first consider equational theories induced by systems of ground equations. Efficient algorithms for computing congruence closure can be used to decide if a ground equation is an equational consequence of a set of ground equations. All algorithms for congruence closure computation rely on the use of certain data-structures, in the process obscuring any inherent logical aspects.

In general, a system of ground equations can be completed into a convergent ground term rewriting system using a total termination ordering. However, this process can in the worst case take exponential time unless the rules are processed using a certain strategy [24]. Even under the specific strategy, the resulting completion procedure is quadratic and the $O(n\log(n))$ efficiency of congruence closure algorithms is not attained. There are known techniques [28] to construct ground convergent systems that use graph based congruence closure algorithms.

We attempt to capture the essence of some of the efficient congruence closure algorithms using standard techniques from term rewriting. We do so by introducing symbols and extending the signature to abstractly represent sharing that is inherent in the use of term directed acyclic graph data structures. We thus define a notion of abstract congruence closure and provide transition rules that can be used to construct such abstract congruence closures. A whole class of congruence closure algorithms can be obtained by choosing suitable strategies (and implementations) for the abstract transition rules. The complexity of any such congruence closure algorithm is directly related to the length of derivation (using these transition rules) required to compute an abstract congruence closure with the chosen strategy. We give bounds on the length of arbitrary maximal derivations and show its relationship with the choice of ordering used for completion.

We describe some of the specific well-known congruence closure algorithms in the framework of abstract congruence closure, and show that the abstract framework suitably captures the sources of efficiency in some of these algorithms. The description separates the logical aspects inherent in these algorithms from implementation details.

The concept of an abstract congruence closure is useful in more than one way. Many other algorithms, like those for syntactic unification and rigid E-unification, that rely either on congruence closure computation or on the use of term dag representation for efficiency, also admit simpler and more abstract descriptions using an abstract congruence closure [6, 5].

Furthermore, if certain function symbols in the signature are assumed to be associative and commutative, we can introduce standard techniques from rewriting modulo an equational theory to handle it. Thus, we obtain a notion of congruence closure modulo associativity and commutativity. As an additional application, we consider the problem of constructing ground convergent systems (in the original signature) for a set of ground equations. We show how to eliminate the new constants introduced earlier to transform all equations back to the original signature while preserving some of the nice properties of the system over the extended signature, thus generalizing the results in [28].

1.1 Preliminaries

Let Σ be a set, called a *signature*, with an associated *arity function* $\alpha : \Sigma \rightarrow 2^{\mathbb{N}}$ and let $\mathcal{V}$ be a disjoint (denumerable) set. We define $\mathcal{T}(\Sigma, \mathcal{V})$ as the smallest set containing $\mathcal{V}$ and such that $f(t_1, \ldots, t_n) \in \mathcal{T}(\Sigma, \mathcal{V})$ whenever $f \in \Sigma, n \in \alpha(f)$ and $t_1, \ldots, t_n \in \mathcal{T}(\Sigma, \mathcal{V})$. The elements of the sets Σ, $\mathcal{V}$ and $\mathcal{T}(\Sigma, \mathcal{V})$ are respectively called *function symbols*, *variables* and *terms* (over Σ and $\mathcal{V}$). Elements c in Σ for which $\alpha(c) = \{0\}$ are called *constants*. By $\mathcal{T}(\Sigma)$ we denote the set $\mathcal{T}(\Sigma, \emptyset)$ of all variable-free, or *ground* terms. The symbols $s, t, u, \ldots$ are used to denote terms; $f, g, \ldots$, function symbols; and $x, y, z, \ldots$, variables. We write $t[s]$ to indicate that a term t contains s as a subterm and (ambiguously) denote by $t[u]$ the result of replacing a particular occurrence of s by u.

An *equation* is a pair of terms, written $s \approx t$. The *replacement relation* $\rightarrow_{E^g}$ induced by a set of equations E is defined by: $u \rightarrow_{E^g} v$ if, and only if, $u = u[l]$ contains l as a subterm and $v = u[r]$ is obtained by replacing l by r in u, where $l \approx r$ is in E. The *rewrite relation* $\rightarrow_E$ induced by a set of equations E is defined by: $u \rightarrow_E v$ if, and only if, $u = u[l\sigma]$, $v = u[r\sigma]$, $l \approx r$ is in E, and σ is some substitution.

If $\rightarrow$ is a binary relation, then $\leftarrow$ denotes its inverse, $\leftrightarrow$ its symmetric closure, $\rightarrow^+$ its transitive closure and $\rightarrow^*$ its reflexive-transitive closure. Thus, $\leftrightarrow^*_{E^g}$ denotes the *congruence relation*[1] induced by E. We will mostly be interested in sets E of ground equations whence the distinction between rewrite relation and replacement relation disappears. The *equational theory* of E is defined as the relation $\leftrightarrow^*_E$. Equations are often called *rewrite rules*, and a set E a *rewrite system*, if one is interested particularly in the rewrite relation $\rightarrow^*_E$ rather than the equational theory $\leftrightarrow^*_E$.

A term t is *irreducible*, or in *normal form*, with respect to a rewrite system R, if there is no term u, such that $t \rightarrow_R u$. We write $s \rightarrow^!_R t$ to indicate that t is an R-normal form of s.

A rewrite system R is said to be (ground) *confluent* if for every pair s, s' of (ground) terms, if there exists a (ground) term t such that $s \leftarrow^*_R t \rightarrow^*_R s'$, then there exists a (ground) term t' such that $s \rightarrow^*_R t' \leftarrow^*_R s'$. Thus, if R is (ground) confluent, then every (ground) term t has at most one normal form. A rewrite system R is *terminating* if there exists no infinite reduction sequence $s_0 \rightarrow_R s_1 \rightarrow_R s_2 \cdots$ of terms. Clearly, if R is terminating, then every term t has at least one normal form. Rewrite systems that are (ground) confluent and terminating are called (ground) *convergent*.

A rewrite system R is *left-reduced* if every left-hand side term (of any rule in R) is irreducible by all other rules in R. A rewrite system R is *right-reduced* if every right-hand side term (of any rule in R) is in R-normal form. A rewrite system that is both left-reduced and right-reduced is said to be *fully reduced*.

[1] A congruence relation is a reflexive, symmetric and transitive relation on terms that is also a replacement relation.

2 Abstract Congruence Closure

We first describe the form of terms and equations that will be used in the description of an abstract congruence closure. Definitions that introduce similar concepts also appear in [15, 16, 17, 26].

Definition 1. *Let Σ be a signature and K be a set of constants disjoint from Σ. A D-*rule *(with respect to Σ and K) is a rewrite rule of the form*

$$f(c_1, \ldots, c_k) \rightarrow c$$

where $f \in \Sigma$ is a k-ary function symbol and $c_1, \ldots, c_k, c$ are constants in set K.

*A C-*rule *(with respect to K) is a rule $c \rightarrow d$, where c and d are constants in K.*

For example, if $\Sigma_0 = \{a, b, f\}$, and $E_0 = \{a \approx b, ffa \approx fb\}$[2] then

$$D_0 = \{a \rightarrow c_0,\ b \rightarrow c_1,\ fc_0 \rightarrow c_2,\ fc_2 \rightarrow c_3,\ fc_1 \rightarrow c_4\}$$

is a set of D-rules over Σ_0 and $K_0 = \{c_0, c_1, c_2, c_3, c_4\}$. Using these D-rules we can simplify the original equations in E_0. For example, the term ffa can be rewritten to c_3 as $ffa \rightarrow_{D_0} ffc_0 \rightarrow_{D_0} fc_2 \rightarrow_{D_0} c_3$. Original equations in E_0 can thus be simplified using D_0 to give $C_0 = \{c_0 \approx c_1, c_3 \approx c_4\}$. The set $D_0 \cup C_0$ may be viewed as an alternative representation of E_0 over an extended signature. The equational theory presented by $D_0 \cup C_0$ is a conservative extension of the theory E_0. This reformulation of the equations E_0 in terms of an extended signature is (implicitly) present in all congruence closure algorithms, see Section 3.

The constants in the set K can be thought of as names for equivalence classes of terms. A D-rule $f(c_1, \ldots, c_k) \rightarrow c_0$ indicates that a term with top function symbol f and arguments belonging to the equivalence classes $c_1, \ldots, c_k$ itself belongs to the equivalence class c_0. In this sense, a set of D-rules can be thought of as defining a bottom-up tree automaton [10]. Other interpretations for the constants in K are possible too, especially in the context of term directed acyclic graph (dag) representation, see Section 3 for details.

A constant c in K is said to *represent* a term t in $\mathcal{T}(\Sigma \cup K)$ (via the rewrite system R) if $t \leftrightarrow_R^* c$. A term t is *represented* by R if it is represented by some constant in K via R. For example, the constant c_3 represents the term ffa via D_0.

Definition 2 (Abstract congruence closure). *Let Σ be a signature and K be a set of constants disjoint from Σ. A ground rewrite system $R = D \cup C$ of D-rules and C-rules (with respect to Σ and K) is said to be an* (abstract) congruence closure *if*

(i) each constant $c \in K$ represents some term $t \in \mathcal{T}(\Sigma)$ via R, and

(ii) R is ground convergent.

If E is a set of ground equations over $\mathcal{T}(\Sigma \cup K)$ and in addition R is such that

(iii) for all terms s and t in $\mathcal{T}(\Sigma)$, $s \leftrightarrow_E^* t$ if, and only if, $s \rightarrow_R^* \circ \leftarrow_R^* t$,

then R will be called an (abstract) congruence closure for E.

[2] When writing a term, we remove parentheses wherever possible for clarity.

Condition (i) essentially states that K contains no superfluous constants; condition (ii) ensures that equivalent terms have the same representative (which usually also implies that congruence of terms can be tested efficiently); and condition (iii) implies that R is a conservative extension of the equational theory induced by E over $\mathcal{T}(\Sigma)$.

The rewrite system $R_0 = D_0 \cup \{c_0 \to c_1,\ c_3 \to c_4\}$ above is not a congruence closure for E_0, as it is not ground convergent. But we can transform R_0 into a suitable rewrite system, using a completion-like process described in more detail below, to obtain a congruence closure

$$R_1 = \{a \to c_1,\ b \to c_1,\ fc_1 \to c_4,\ fc_4 \to c_4,\ c_0 \to c_1,\ c_2 \to c_4,\ c_3 \to c_4\}.$$

2.1 Construction of Abstract Congruence Closures

We next present a general method for construction of an abstract congruence closure. Our description is fairly abstract, in terms of transition rules that manipulate triples (K, E, R), where K is the set of constants that extend the original fixed signature Σ, E is the set of ground equations (over $\Sigma \cup K$) yet to be processed, and R is the set of C-rules and D-rules that have been derived so far. Triples represent *states* in the process of constructing a congruence closure. Construction starts from an *initial state* $(\emptyset, E, \emptyset)$, where E is a given set of ground equations.

The transition rules can be derived from those for standard completion as described in [3], with some differences so that (i) application of the transition rules is guaranteed to terminate and (ii) a convergent system is constructed over an *extended* signature. The transition rules do *not* require a total reduction ordering[3] on terms in $\mathcal{T}(\Sigma)$, but simply an ordering on $\mathcal{T}(\Sigma \cup U)$ (that is, terms in $\mathcal{T}(\Sigma)$ need not be comparable in this ordering) where U is an infinite set disjoint from Σ from which new constants $K \subset U$ are chosen. In particular, we assume $\succ_U$ is any ordering on the set U and define $\succ$ by: $c \succ d$ if $c \succ_U d$ and $t \succ c$ if $t \to c$ is a D-rule. For simplicity, we take U to be the set $\{c_0, c_1, c_2, \ldots\}$ and assume that $c_i \succ_U c_j$ if, and only if, $i < j$.

A key transition rule introduces new constants as names for subterms.

$$\textbf{Ext}\text{ension:} \qquad \frac{(K, E[t], R)}{(K \cup \{c\}, E[c], R \cup \{t \to c\})}$$

where $t \to c$ is a D-rule, t is a term occurring in (some equation in) E, and $c \in U - K$.

The following three rules are versions of the corresponding rules for standard completion specialized to the ground case.

$$\textbf{Sim}\text{plification:} \qquad \frac{(K, E[t], R \cup \{t \to c\})}{(K, E[c], R \cup \{t \to c\})}$$

[3] By an *ordering* we mean any irreflexive and transitive relation on terms. A *reduction ordering* is an ordering that is also a well-founded replacement relation. An ordering $\succ$ is *total* if for any two distinct elements s and t, either $s \succ t$ or $t \succ s$.

where t occurs in some equation in E. (It is fairly easy to see that by repeated application of extension and simplification, any equation in E can be reduced to an equation that can be oriented by the ordering $\succ$.)

Orientation: $$\frac{(K \cup \{c\}, E \cup \{t \approx c\}, R)}{(K \cup \{c\}, E, R \cup \{t \to c\})}$$

if $t \succ c$.

Trivial equations may be deleted.

Deletion: $$\frac{(K, E \cup \{t \approx t\}, R)}{(K, E, R)}$$

In the case of completion of ground equations, deduction steps can all be replaced by suitable simplification steps, in particular by collapse. However, in order to guarantee termination, we formulate collapse by two different specialized transition rules. The usual side condition in the collapse rule, which refers to the *encompassment ordering*, can be considerably simplified in our case.

Deduction: $$\frac{(K, E, R \cup \{t \to c,\ t \to d\})}{(K, E \cup \{c \approx d\}, R \cup \{t \to d\})}$$

Collapse: $$\frac{(K, E, R \cup \{s[c] \to c',\ c \to d\})}{(K, E, R \cup \{s[d] \to c',\ c \to d\})}$$

if c is a proper subterm of s.

As in standard completion the simplification of right-hand sides of rules in R by other rules is optional and not necessary for correctness. Right-hand sides of rules in R are always constants.

Composition: $$\frac{(K, E, R \cup \{t \to c,\ c \to d\})}{(K, E, R \cup \{t \to d,\ c \to d\})}$$

Various known congruence closure algorithms can be abstractly described using different strategies over the above rules. All the above transition rules with the exception of the composition rule, constitute the *mandatory* set of transition rules.

Example 1. Consider the set of equations $E_0 = \{a \approx b,\ ffa \approx fb\}$. An abstract congruence closure for E_0 can be derived from $(K_0, E_0, R_0) = (\emptyset, E_0, \emptyset)$ as follows:

i	Constants K_i	Equations E_i	Rules R_i	Transition Rule
0	$\emptyset$	E_0	$\emptyset$	
1	$\{c_0\}$	$\{c_0 \approx b, ffa \approx fb\}$	$\{a \to c_0\}$	**Ext**
2	$\{c_0\}$	$\{ffa \approx fb\}$	$\{a \to c_0, b \to c_0\}$	**Ori**
3	$\{c_0\}$	$\{ffc_0 \approx fc_0\}$	$\{a \to c_0, b \to c_0\}$	**Sim** (twice)
4	$\{c_0, c_1\}$	$\{fc_1 \approx fc_0\}$	$R_3 \cup \{fc_0 \to c_1\}$	**Ext**
5	$\{c_0, c_1\}$	$\{fc_1 \approx c_1\}$	$R_3 \cup \{fc_0 \to c_1\}$	**Sim**
6	K_5	$\{\}$	$R_5 \cup \{fc_1 \to c_1\}$	**Ori**

The rewrite system R_6 is an abstract congruence closure for E_0.

2.2 Correctness

We use the symbol $\vdash$ to denote the one-step transformation relation on states induced by the above transformation rules. A *derivation* is a sequence of states $(K_0, E_0, R_0) \vdash (K_1, E_1, R_1) \vdash \cdots$.

Theorem 1 (Soundness). *If $(K, E, R) \vdash (K', E', R')$, then, for all terms s and t in $\mathcal{T}(\Sigma \cup K)$, we have $s \leftrightarrow^*_{E' \cup R'} t$ if, and only if, $s \leftrightarrow^*_{E \cup R} t$.*

Proof. For simplification, orientation, deletion and composition, the claim follows from correctness result for the standard completion transition rules [3]. The claim is also easily verified for the specialized collapse and deduction rules.

Now, suppose $(K', E', R' = R \cup \{u \to c\})$ is obtained from (K, E, R) using extension. For $s, t \in \mathcal{T}(\Sigma \cup K)$, if $s \leftrightarrow^*_{E \cup R} t$, then clearly $s \leftrightarrow^*_{E' \cup R'} t$. Conversely, if $s \leftrightarrow^*_{E' \cup R'} t$, then, $s\sigma \leftrightarrow^*_{E'\sigma \cup R'\sigma} t\sigma$, where σ is (homomorphic extension of) the mapping $c \mapsto u$. But $s\sigma = s$ and $t\sigma = t$ as $c \notin K$. Furthermore, $E'\sigma = E$, and $R'\sigma = R \cup \{u \to u\}$. Therefore, $s = s\sigma \leftrightarrow^*_{E \cup R} t\sigma = t$. ∎

Lemma 1. *Let K_0 be a finite set of constants (disjoint from Σ), E_0 a finite set of equations (over $\Sigma \cup K$) and R_0 a finite set of D-rules and C-rules such that for every C-rule $c \to d$ in R_0 we have $c \succ_U d$. Then each derivation starting from the state (K_0, E_0, R_0) is finite. Furthermore, if $(K_0, E_0, R_0) \vdash^* (K_m, E_m, R_m)$, then the rewrite system R_m is terminating.*

Proof. We first define the measure of a state (K, E, R) to be the number of occurrences of symbols from Σ in E. Two states are compared by comparing their measures using the usual "greater-than" ordering on natural numbers. It can be easily verified that each transformation rule either reduces this measure, or leaves it unchanged. Specifically, extension always reduces this measure.

Now, suppose there is a derivation starting from the state (K_0, E_0, R_0). Any such derivation can be written as

$$(K_0, E_0, R_0) \vdash^* (K_n, E_n, R_n) \vdash (K_{n+1}, E_{n+1}, R_{n+1}) \vdash \cdots$$

where the derivation $(K_n, E_n, R_n) \vdash (K_{n+1}, E_{n+1}, R_{n+1}) \vdash \cdots$ contains no applications of extension, and hence the set $K_n = K_{n+1} = \cdots$ is finite. Therefore, the ordering $\succ_{K_n}$ (defined as the restriction of the ordering $\succ_U$ on K_n) is well-founded.

Next we prove that the derivation $(K_n, E_n, R_n) \vdash (K_{n+1}, E_{n+1}, R_{n+1}) \vdash \cdots$ is finite. Assign a weight $w(c)$ to each symbol c in K_n so that $w(c) > w(d)$ if, and only if, $c \succ_{K_n} d$; and set $w(f) = \max\{w(c) : c \in K_n\} + 1$, for each $f \in \Sigma$. Let $\gg$ be the Knuth-Bendix ordering using these weights. Define a secondary measure of a state (K, E, R) as the set $\{\{\{s, t\}\} : s \approx t \in E\} \cup \{\{\{s\}, \{t\}\} : s \to t \in R\}$. Two states are compared by comparing their measures using a two-fold multiset extension[4] of the ordering $\gg$ on terms. It is straight forward to see that application of any transition rule (except extension) to a state reduces the secondary measure of the state. Moreover, every rule in R_j is reducing in the reduction ordering $\gg$, and hence each rewrite system R_j is terminating. ∎

The following lemma says that extension introduces no superfluous constants.

Lemma 2. *Suppose $(K, E, R) \vdash (K', E', R')$ and that for every $c \in K$, there exists a term $s \in \mathcal{T}(\Sigma)$ such that $c \leftrightarrow^*_{E \cup R} s$. Then, for every $d \in K'$, there exists a term $t \in \mathcal{T}(\Sigma)$ such that $d \leftrightarrow^*_{E' \cup R'} t$.*

Proof. If $d \in K'$ also belongs to the set K, then the claim is easily proved using Theorem 1. Otherwise let $d \in K' - K$. The only non-trivial case is the case when (K', E', R') is obtained using extension.

Let $f(c_1, \ldots, c_k) \to d$ be the rule introduced by extension. Since $c_1, \ldots, c_k \in K$, there exist terms $s_1, \ldots, s_k \in \mathcal{T}(\Sigma)$ such that $s_i \leftrightarrow^*_{E \cup R} c_i$ and hence, using Theorem 1, $s_i \leftrightarrow^*_{E' \cup R'} c_i$. The term $f(s_1, \ldots, s_k)$ is the required term t. ∎

We call a state (K, E, R) *final* if no mandatory transition rule is applicable to this state. The third component of a final state is always an abstract congruence closure.

Theorem 2. *Let Σ be a signature and K_1 a finite set of constants disjoint from Σ. Let E_1 be a finite set of equations over $\Sigma \cup K_1$ and R_1 be a finite set of D-rules and C-rules such that every $c \in K_1$ represents some term $t \in \mathcal{T}(\Sigma)$ via $E_1 \cup R_1$, and $c \succ_U d$ for every C-rule $c \to d$ in R_1. If (K_n, E_n, R_n) is a final state such that $(K_1, E_1, R_1) \vdash^* (K_n, E_n, R_n)$, then $E_n = \emptyset$ and R_n is an abstract congruence closure for $E_1 \cup R_1$ (over Σ and K_n).*

Proof. By Lemma 1, we know that K_n, E_n and R_n are all finite sets. If $E_n \neq \emptyset$, then either extension or orientation will be applicable. Since (K_n, E_n, R_n) is a final state, $E_n = \emptyset$.

In order to show that R_n is an abstract congruence closure for $E_1 \cup R_1$, we need to prove the three conditions in Definition 2.

1. Lemma 2 implies that every $c \in K_n$ represents some term $t \in \mathcal{T}(\Sigma)$ via R_n.
2. Using Lemma 1 we know that R_n is terminating. Furthermore, since (K_n, E_n, R_n) is a final state, R_n is left-reduced. By the critical pair lemma [1], therefore, R_n is confluent; and hence convergent.
3. Finally, Theorem 1 establishes that if $s \leftrightarrow^*_{E_1 \cup R_1} t$ for some $s, t \in \mathcal{T}(\Sigma)$, then $s \leftrightarrow^*_{E_n \cup R_n} t$. Since $E_n = \emptyset$ and R_n is convergent, $s \to^*_{R_n} \circ \leftarrow^*_{R_n} t$. ∎

[4] A *multiset* over a set S is a mapping M from S to the natural numbers. Any ordering $\succ$ on a set S can be extended to an ordering $\succ^m$ on multisets over S as follows: $M \succ^m N$ iff $M \neq N$ and whenever $N(x) > M(x)$ then $M(y) > N(y)$, for some $y \succ x$. The multiset ordering $\succ^m$ (on finite multisets) is well founded if the ordering $\succ$ is well founded [12].

2.3 Properties

To summarize, we have presented an abstract notion of congruence closure and given a method to construct such an abstract congruence closure for a given set of ground equations. The only parameters required by the procedure are a denumerable set U of constants (disjoint from Σ) and an ordering (irreflexive and transitive relation) on this set. It might appear that the abstract congruence closure one obtains depends on the ordering $\succ_U$ used. In this section, we first show that we can construct an abstract congruence closure that is independent of the ordering on constants.

In the process of construction of an abstract congruence closure, we may deduce an equality between two constants in K, and we require an ordering $\succ_U$ to deal with such equations. Since constants are essentially "names" for equivalence classes, it is redundant to have two different names for the same equivalence class. Hence, one such constant and the corresponding ordering dependence can be eliminated.

Definition 3. *Any constant $c \in K$ that occurs as a left-hand side of a C-rule in R is called* redundant *in R.*

Redundant constants in R can be eliminated after composition and collapse steps with C-rules in R have been applied exhaustively.

Compression: $$\frac{(K \cup \{c, d\}, E, R \cup \{c \to d\})}{(K \cup \{d\}, E\langle c \mapsto d\rangle, R\langle c \mapsto d\rangle)}$$

if c occurs only once as a left-hand side term, the notation $\langle c \mapsto d\rangle$ denotes the homomorphic extension of the mapping σ defined as $\sigma(c) = d$ and $\sigma(x) = x$ for $x \neq c$, and $E\langle c \mapsto d\rangle$ denotes the set of equations obtained by applying the mapping $\langle c \mapsto d\rangle$ to each term in the set E.

Correctness of the new enhanced set of transition rules for construction of congruence closure can be established in the same way as before.

Theorem 3. *Let Σ be a signature and E be a finite set of equations over Σ. Then, there exists an abstract congruence closure D for E (over Σ and some K) consisting only of D-rules.*

Proof. Let $(\emptyset, E, \emptyset) \vdash^* (K_n, E_n, R_n)$ such that none of the mandatory transition rules nor compression is applicable to the state (K_n, E_n, R_n).

We observe that the following version of soundness (Theorem 1) is still true: if $(K_i, E_i, R_i) \vdash (K_j, E_j, R_j)$, then, for all terms s and t in $\mathcal{T}(\Sigma)$, $s \leftrightarrow^*_{E_j \cup R_j} t$ iff $s \leftrightarrow^*_{E_i \cup R_i} t$. Additionally, Lemma 1 and Lemma 2 continue to hold with the new set of transition rules, and the proofs remain essentially unchanged. This establishes that we can use Theorem 2 in this new setting to conclude that R_n is an abstract congruence closure. Finally, since compression is not applicable to the final state, there can be no C-rules in R_n. ∎

Graph-based congruence closure algorithms can be described using D-rules, see Section 3. However, we can define a *generalized D-rule* (with respect to Σ and K) as any rule of the form $t \to c$ where $c \in K$ and $t \in \mathcal{T}(\Sigma, K) - K$, as done in [5]. The transition rules for construction of congruence closure can be suitably generalized with minimal changes. The new definition of D-rules allows for preserving as much of the original term structure as possible.

As remarked earlier, the set of transition rules presented in Section 2.1 for construction of abstract congruence closure is parameterized by a denumerable set U of constants and an ordering $\succ_U$ on this set. Since elements of U serve only as names, we can choose U to be any countable set of symbols. An ordering $\succ_U$ need not be specified a priori but can be defined on the fly as the derivation proceeds. We need to maintain irreflexivity whenever the ordering relation is extended. Observe that we only need an ordering when there is a C-equation to orient.

If we exhaustively apply simplification before trying to orient a C-equation, *any* orientation of the fully simplified C-equation can be used. Given a derivation $(K_0, E_0, D_0 \cup C_0) \vdash \cdots \vdash (K_i, E_i, D_i \cup C_i)$ using this strategy, we construct a sequence of relations $\succ_0, \succ_1, \ldots$, where each $\succ_j$ is defined by $c \succ_j d$ only if $c \to d \in \cup_{k \leq j} C_k$. We claim that each $\succ_j$ defines an ordering. To see this note that $\succ_0$ defines a trivial ordering (in which no two elements in U are comparable). Moreover, whenever the relation $\succ_j$ is extended by $c \succ d$, the constants c and d are incomparable in the transitive closure of the existing relation $\succ_j$, and hence irreflexivity of the ordering defined by $\succ_{j+1}$ is established.

The above observation establishes that there exist derivations for congruence closure construction in which we do not spend any time in comparing elements. However, we will shortly show that the length of derivations crucially depends on the chosen ordering. This reveals a tradeoff between the effort spent in choosing an ordering and the lengths of derivations obtained when using that ordering.

Definition 4. *An ordering $\succ$ on the set U is* feasible *for a state (K, E, R) if there exists an unfailing*[5] *maximal derivation starting from the state (K, E, R) that uses the ordering $\succ$.*

The depth *or* height *of an ordering $\succ$ is the length of the longest chain. More specifically, if the longest chain for ordering $\succ$ is $c_0 \succ c_1 \succ \cdots \succ c_\delta$, then the depth of $\succ$ is δ.*

Congruence closure computation using specialized data structures is known to be more efficient than naive standard completion. We next show, by proving a bound on the length of *any* maximal derivation, that our description captures the cause of this efficiency.

Lemma 3. *Any maximal derivation starting from state $(K_0 = \emptyset, E_0, R_0 = \emptyset)$ is of length $O((2k + l)\delta + n)$, where k is the number of applications of extension, l is the difference between the number of occurrences of 0-arity symbols in E_0 and number of distinct 0-arity symbols in E_0, δ is the depth of ordering $\succ_U$ used to construct the derivation, and n is the number of Σ-symbols in E_0.*

[5] By *unfailing* we mean that the set of unoriented equations in the final state is empty.

Proof. In order to simplify the argument, we first split simplification and deduction rules as follows (ignoring the K-component):

$$\textbf{Sim1}: \frac{(E[f(\ldots)], R \cup \{f(\ldots) \to c\})}{(E[c], R \cup \{f(\ldots) \to c\})} \qquad \textbf{Sim2}: \frac{(E[c], R \cup \{c \to d\})}{(E[d], R \cup \{c \to d\})}$$

$$\textbf{Ded1}: \frac{(E, R \cup \{f(\ldots) \to c,\ f(\ldots) \to d\})}{(E \cup \{c \approx d\}, R \cup \{f(\ldots) \to d\})} \quad \textbf{Ded2}: \frac{(E, R \cup \{c \to d,\ c \to d'\})}{(E \cup \{d \approx d'\}, R \cup \{c \to d\})}$$

Next, we bound the number of applications of individual rules in *any* derivation as follows:
(i) a derivation step using either sim2, ded2, collapse, or composition corresponds to *rewriting* some constant. Since the length of a rewriting sequence $c_1 \to c_2 \to \cdots$ is bounded by δ and $2k + l$ is an upper bound on the number of occurrences of constants (from K_∞) in $E_i \cup R_i$ (for any i), therefore the number of applications of sim2, ded2, collapse, and composition is $O((2k + l)\delta)$;
(ii) the number of deletion steps is at most $|E_0| + k$ as each transition rule, with the exception of extension and deletion, preserves the cardinality of $E_i \cup R_i$ and extension increases this number by one while deletion decreases it by one;
(iii) the number of sim1 and ded1 steps is at most n as each such step reduces the number of Σ-symbols (in $E \cup R$);
(iv) the number of Extension steps is k; and
(v) application of Orientation at most doubles the length of any derivation.

Thus, the total length of any derivation is $O((2k + l)\delta + n)$. ∎

The number k of extension steps used in any maximal derivation is $O(n)$ because the total number of Σ-symbols in the second component of the state is non-increasing in any derivation and an application of extension reduces this number by one.

Lemma 4. *A starting state $(K_0 = \emptyset, E_0, R_0 = \emptyset)$ can be transformed into a state (K_m, E_m, R_m) in $O(n)$ derivation steps, where n is the total number of symbols in the finite set E_0 of ground equations, such that (i) the set E_m consists of only C-equations and R_m consists of only D-rules, and (ii) the total number of symbols in $E_m \cup R_m$ is $O(n)$.*

Proof. We construct the desired derivation by an exhaustive application of extension and simplification rules. Clearly, the set E_m contains only C-equations and R_m contains only D-rules. The length of this derivation is $O(n)$ as every application of extension and simplification reduces the total number of Σ-symbols in E_i by at least one. Moreover, the total number of symbols in $E_m \cup R_m$ is $O(n)$ because every application of extension and simplification increases the total number of symbols by a constant. ∎

Informally speaking, therefore, since l is clearly $O(n)$, Lemma 3 gives us an upper bound of $O(n\delta)$ on the length of maximal derivations. Any total (linear) order on the set K_∞ of constants is feasible, but has depth equal to the cardinality of K_∞, which is $O(n)$. This gives a quadratic bound on the length of a derivation. However, we can also show that there exist feasible orderings with smaller depth.

Lemma 5. *Let (K_m, E_m, R_m) be a state such that E_m consists of only C-equations and R_m consists of only D-rules. Then, there exists a feasible ordering $\succ_U$ for this state with depth $O(\log(n))$, where n is the number of constants in K_m.*

Proof. We shall exhibit an unfailing derivation that constructs the required ordering on the fly as discussed before, i.e., during the derivation, we ensure that whenever we apply orientation as $(K_i, E_i \cup \{c \approx d\}, D_i \cup C_i) \vdash (K_i, E_i, D_i \cup C_i \cup \{c \to d\})$, the constants c and d are in C_i-normal form. Additionally, we also impose the requirement that the cardinality of the set $\{c' \in K_m : c' \leftrightarrow^*_{C_i} c\}$ is less than or equal to the cardinality of $\{c' \in K_m : c' \leftrightarrow^*_{C_i} d\}$.

As argued before, the relation thus built defines an ordering. Suppose $(K_\infty, E_\infty, D_\infty \cup C_\infty)$ is the final state of this unfailing derivation. If $c_1 \succ c_2 \succ \cdots \succ c_j$ is a maximal descending chain, then the cardinality of the set $\{c' \in K_m : c' \leftrightarrow^*_{C_\infty} c_j\}$ is at least 2^{j-1}. But, since the cardinality of K_m is $O(n)$, therefore, $j = O(\log(n))$. ∎

Combining these three lemmas leads to the following result.

Theorem 4. *There exists a maximal derivation of length $O(n \log(n))$ with starting state $(\emptyset, E_0, \emptyset)$, where n is the total number of symbols in the finite set E_0 of ground equations.*

Proof. We construct the derivation in two stages. In the first stage we use the derivation constructed in proof of Lemma 4 to obtain an intermediate state (K_m, E_m, R_m) from the starting state $(K_0 = \emptyset, E_0, R_0 = \emptyset)$. In the second stage, we start with this intermediate state and carry out the derivation in the proof of Lemma 5 to reach a final state. The claim then follows from Lemma 4 and Lemma 3. ∎

Theorem 4 establishes the possibility of obtaining short maximal derivations using (simple strategies on) the abstract transition rules. However, in order to get an efficient, say $O(n \log(n))$, algorithm for computing a congruence closure, we need to show that the ordering on constants can be efficiently computed and each individual step in the derivation can be applied in (amortized) constant time. The first of these is easily achieved by extending the state triple (K, E, R) by an additional component which is a function, `counter`, that maps each constant in K to a natural number. More precisely, `counter`(c) stores the cardinality of the set

$$[c]_C \stackrel{\text{def}}{=} \{c' \in K : c' \leftrightarrow^*_C c\}$$

where C is the set of C-equations in R. Thus, `counter`(c) is the number of constants in the current equivalence class of c (see proof of Lemma 5). The function `counter` can easily be updated when a C-equation, say $c \approx d$, is oriented into, say $c \to d$, by setting `counter`(d) = `counter`(c) + `counter`(d).

Secondly, efficient application of each transition step requires specialized data structures and/or efficient indexing mechanisms. Some such details have been described in the literature and we discuss these in the next section.

We observe here that in the special case when each congruence class modulo E_0 is finite, feasible orderings with constant depth can be constructed efficiently on the fly. During orientation, only those C-equations, which contain constants whose congruence class $[c]_{C_i}$ (w.r.t. the set C_i of C-equations in the present state) is known to *not* change in subsequent states, are oriented. That such C-equations always exist and can be efficiently identified is a simple consequence of the finiteness assumption, see [29, 14] for details. Thus, this yields a linear bound on the length of (certain) maximal derivations for construction of congruence closure in this special case.

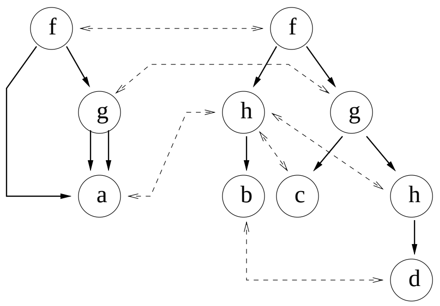
3 Congruence Closure Strategies

The literature abounds with various implementations of congruence closure algorithms. The general framework of abstract congruence closure can be used to uniformly describe the logical characteristics of such algorithms and provides a context for interpreting differences in their performance. We next describe the algorithms proposed by Downey, Sethi and Tarjan [14] and Shostak [27] in this way. That is, we provide a description of these algorithms (the description does not capture certain implementation details) using abstract congruence closure transition rules.

Directed acyclic graphs (dags) are a common data structure used to implement algorithms that work with terms. In fact, many congruence closure algorithms assume that the input is an equivalence relation on vertices of a given dag, and the desired output, the congruence closure of this equivalence, is again represented by an equivalence on the same dag.

A set of C-rules and D-rules may be interpreted as an abstraction of a dag representation. The constants in K (or U) represent nodes in a dag. The D-rules specify edges and the C-rules represent a binary relation on the nodes. More precisely, a D-rule $f(c_1, \ldots, c_k) \rightarrow c$ specifies that the node c is labelled by the symbol f and has pointers to the nodes $c_1, \ldots, c_k$. Conversely, any dag and an associated binary relation on its nodes can be represented using D-rules and C-rules. Figure 1 illustrates the representation of a set of terms (and a binary relation on them) using dags and using D-rules and C-rules. The solid lines represent *subterm* edges, and the dashed lines represent a binary relation on the vertices. We have a D-rule corresponding to each vertex, and a C-rule for each dashed edge. (We note here that generalized D-rules (with respect to Σ and K) as defined in Section 2.3 correspond to storing *contexts*, rather than just symbols from Σ, in each node of the term dag. We do not pursue this optimization in this paper.)

Traditional congruence closure algorithms employ data structures which are suitably abstracted in our presentation as follows:
(i) To obtain a representation via D-rules and C-equations for the *input dag* cor-



D-rules representing the term dag:

$$\begin{array}{rrr} a \to c_1 & gc_1c_1 \to c_2 & fc_1c_2 \to c_3 \\ b \to c_4 & hc_4 \to c_5 & c \to c_6 \\ d \to c_7 & hc_7 \to c_8 & gc_6c_8 \to c_9 \\ fc_5c_9 \to c_{10} & & \end{array}$$

C-rules representing the relation on vertices:

$$\begin{array}{lll} c_1 \approx c_5 & c_2 \approx c_9 & c_3 \approx c_{10} \\ c_4 \approx c_7 & c_6 \approx c_5 & c_5 \approx c_8 \end{array}$$

Fig. 1. A term dag and a relation on its vertices

responding to equation set E_0, we start from the state $(\emptyset, E_0, \emptyset)$, and repeatedly apply a single extension step followed by an exhaustive application of simplification (represented using the expression $(\mathbf{Ext} \cdot \mathbf{Sim}^*)^*$). In the resulting state (K_1, E_1, D_1), the set D_1 represents the input dag and the set E_1 contains only C-equations representing the input equivalence on nodes of this dag. Note that due to eager simplification, we obtain representation of a dag with maximum possible sharing. For example, if $E_0 = \{a \approx b, ffa \approx fb\}$, then $K_1 = \{c_0, c_1, c_2, c_3, c_4\}$, $E_1 = \{c_0 \approx c_1, c_3 \approx c_4\}$ and $R_1 = \{a \to c_0, b \to c_1, fc_0 \to c_2, fc_2 \to c_3, fc_1 \to c_4\}$.
(ii) The signature of a term $f(t_1, \ldots, t_k)$ is defined as $f(c_1, \ldots, c_k)$ where c_i is the name of the equivalence class containing term t_i. A *signature table* (indexed by vertices of the input dag) stores a signature for some or all vertices. A signature table specifies a set of fully left-reduced D-rules.
(iii) The *use table* (also called predecessor list) is a mapping from the constant c to the set of all nodes whose signature contains c. In our presentation this translates to a method of indexing the set of D-rules.
(iv) *Union-find* data structure is used to maintain equivalence classes on the set of nodes of the input dag. In the abstract representation, C-rules describe equivalence relations on constants in K. Operations on the union-find structure exhibit as transitions on C-rules. For instance, application of composition specifies path-compression on the union-find structure.

We note that D-rules serve a two-fold purpose: they represent a term dag, and also a signature table.

Shostak's Algorithm We show that Shostak's congruence closure procedure is a specific strategy over the general transition rules for abstract congruence closure.

Shostak's congruence closure is *dynamic* in that equations are processed one at a time. The strategy underlying Shostak's procedure can be described by the following regular expression:

$$((\mathbf{Sim}^* \cdot \mathbf{Ext}^*)^* \cdot (\mathbf{Del} \cup \mathbf{Ori}) \cdot (\mathbf{Col} \cdot \mathbf{Ded}^*)^*)^*$$

This expression should be interpreted as follows. Given a (start) state (K, E, R) (i) pick an equation $s \approx t$ from the set E, (ii) apply simplification to this state to normalize s, i.e., $s \to_R^! s'$, (iii) exhaustively apply extension to create D-rules for subterms of s' until s' reduces to a constant, say c. Perform steps (ii) and (iii) on the other term t as well to get a constant d. (iv) if c and d are identical then apply deletion (and continue with (i)), and if not, create a C-rule, say $c \to d$, using orientation. (v) Replace c by d using collapse and follow it by exhaustive application of deduction. Repeat this until there are no more possible collapse steps. Finally, the steps (i) through (v) are repeatedly applied. Shostak's procedure halts if no unoriented equations remain.

Shostak's procedure uses indexing based on the idea of the *use*() list. This *use*() based indexing helps in identifying all possible collapse applications.

It is fairly easy to observe that a maximal derivation starting from state $(\emptyset, E_0, \emptyset)$ and using the above strategy ends in a *final* state. Hence, Theorem 2 establishes that the third component of Shostak's halting state is convergent and an abstract congruence closure (for E_0).

Example 2. We use the set E_0 from Example 1 to illustrate Shostak's method, showing the essential intermediate steps in the derivation.

i	Constants K_i	Equations E_i	Rules R_i	Transition
0	$\emptyset$	E_0	$\emptyset$	
1	$\{c_0, c_1\}$	$\{ffa \approx fb\}$	$\{a \to c_0,\ b \to c_1,\ c_0 \to c_1\}$	$\mathbf{Ext} \cdot \mathbf{Ext} \cdot \mathbf{Ori}$
2	$\{c_0, c_1\}$	$\{ffc_1 \approx fb\}$	$\{a \to c_0,\ b \to c_1,\ c_0 \to c_1\}$	$\mathbf{Sim} \cdot \mathbf{Sim}$
3	$\{c_0, \ldots, c_3\}$	$\{c_3 \approx fb\}$	$R_2 \cup \{fc_1 \to c_2,\ fc_2 \to c_3\}$	$\mathbf{Ext} \cdot \mathbf{Ext}$
4	$\{c_0, \ldots, c_3\}$	$\{c_3 \approx c_2\}$	R_3	$\mathbf{Sim} \cdot \mathbf{Sim}$
5	$\{c_0, \ldots, c_3\}$	$\emptyset$	$R_4 \cup \{c_3 \to c_2\}$	$\mathbf{Ori}$

Downey, Sethi, and Tarjan's Algorithm This algorithm assumes that the input is a dag and an equivalence relation on its vertices. Thus, the starting state is a triple given by $(K_1, \emptyset, D_1 \cup C_1)$, where D_1 represents the input dag and C_1 the given equivalence. The underlying strategy of this algorithm can be described as:

$$((\mathbf{Col} \cdot (\mathbf{Ded} \cup \{\epsilon\}))^* \cdot (\mathbf{Sim}^* \cdot (\mathbf{Del} \cup \mathbf{Ori}))^*)^*$$

where ϵ is the null transition rule. This strategy is implemented by repeating the following steps: (i) Repeatedly apply the collapse rule and any resulting deduction steps until no more collapse steps are possible. (ii) if no collapse steps are possible, repeatedly select a C-equation, fully simplify it and then either delete or orient it.

In the Downey, Sethi and Tarjan procedure an equation $c \approx d$ is oriented to $c \to d$ if the equivalence class c contains fewer terms (in the set of all subterms in the input set of equations) than the equivalence class d. This is crucial in ensuring the $O(n \log(n))$ time complexity for this algorithm, c.f. Theorem 4.

If $(K_n, E_n, D_n \cup C_n)$ is the last state in a derivation from $(K_1, \emptyset, D_1 \cup C_1)$ using the above strategy, then, $(K_n, E_n, D_n \cup C_n)$ is a final state, and hence the set $D_n \cup C_n$ is convergent and an abstract congruence closure. The rewrite system D_n represents the information contained in the signature table, and C_n represents information in the union-find structure. The set C_n is usually considered the output of the Downey, Sethi and Tarjan procedure.

Example 3. We illustrate the Downey-Sethi-Tarjan algorithm by using the same set of equations E_0 as above. The start state is $(K_1, \emptyset, D_1 \cup C_1)$ where $K = \{c_0, \ldots, c_4\}$, $D_1 = \{a \to c_0,\ b \to c_1,\ fc_0 \to c_2,\ fc_2 \to c_3,\ fc_1 \to c_4\}$, and, $C_1 = \{c_0 \to c_1,\ c_3 \to c_4\}$.

i	Consts K_i	Eqns E_i	Rules R_i	Transition
1	K_1	$\emptyset$	$D_1 \cup C_1$	
2	K_1	$\emptyset$	$\{a \to c_0,\ b \to c_1,\ fc_1 \to c_2,$ $fc_2 \to c_3,\ fc_1 \to c_4\} \cup C_1$	**Col**
3	K_1	$\{c_2 \approx c_4\}$	$R_2 - \{fc_1 \to c_2\}$	**Ded**
4	K_1	$\emptyset$	$R_3 \cup \{c_4 \to c_2\}$	**Ori**

Note that $c_4 \approx c_2$ was oriented in a way that no further collapses were needed thereafter.

4 Experimental Results

We have implemented several congruence closure algorithms, including those proposed by Nelson and Oppen (NO) [22], Downey, Sethi and Tarjan (DST) [14], and Shostak (SHO) [27], and two algorithms based on completion—one with an indexing mechanism (IND) and the other without (COM). Implementation of the first three procedures is based on the representation of terms by directed acyclic graphs and the representation of equivalence classes by a union-find data structure. Union-find data structure uses path compression and the same code (with only minor variations) is used in all the three implementations.

NO is an implementation of the pseudocode given on page 358 (with some details on page 359) of [22]. In particular, the predecessor lists are kept sorted and duplicates are removed whenever two predecessor lists are merged. Furthermore, the double loop described in step 4 of the algorithm is implemented as an optimized linear search (with a "sorting" overhead) as suggested in [22]. We tested other minor variants too. The one variant in which splicing the predecessor list was done in constant time (allowing for duplicates in the process) and step 4 was implemented as a nested loop, gave the best running times on our examples, which we report here.

The DST implementation corresponds exactly to the pseudocode on page 761 of [14]. In particular, *signature table* is implemented as a hash table, equivalence classes are represented in union-find, and the sets *pending* and *combine* are implemented as singly linked lists of pointers to graph nodes and to graph edges respectively.

Implementation SHO of Shostak's algorithm is based on the specialization to the pure theory of equality of the combination method described in page 8 of [27]. The main data structures in the implementation are the union-find, *use* lists, and *sig* which stores a signature for each vertex. The manipulation of these data structures, especially the *use* lists, and the sequence of calls to *merge* is exactly as described in [27]. Description of this algorithm (with only a slight difference in the order of calls to subroutine merge) is also present in [11, 17].

The completion procedure COM uses the following strategy:

$$((\mathbf{Sim}^* \cdot \mathbf{Ext}^*)^* \cdot (\mathbf{Del} \cup \mathbf{Ori}) \cdot ((\mathbf{Com}^* \cdot \mathbf{Col}^*) \cdot \mathbf{Ded} \cdot (\mathbf{Del} \cup \mathbf{Ori}))^*)^*.$$

More specifically, we process one equation at a time, fully simplify it and if necessary use extension to generate a C-equation. The C-equation is oriented and composition and collapse are applied exhaustively, followed by a deduction step. The generated C-equation is similarly handled. When no more C-equations can be produced, we process the next equation. In short, this strategy is based on eager elimination of redundant constants.

The indexed variant IND uses a slightly different strategy

$$((\mathbf{Sim}^* \cdot \mathbf{Ext}^*)^* \cdot ((\mathbf{Del} \cup \mathbf{Ori}) \cdot (\mathbf{Col}^* \cdot \mathbf{Com}^? \cdot \mathbf{Ded}^?)^* \cdot \mathbf{Sim}^*)^*)^*.$$

As before, using $\mathbf{Sim}^* \cdot \mathbf{Ext}^*$ we convert one equation to a C-equation. This equation is oriented and individually on every D-rules, we perform all simplifications using this C-rule, viz collapse, composition, followed by any deduction step $(\mathbf{Col}^* \cdot \mathbf{Com}^? \cdot \mathbf{Ded}^?)$. Subsequently, simplification of equations using the oriented C-rule are done. All the C-equations are processed this way before we take up the next equation to process. Indexing refers to the use of suitable data structures to efficiently identify which D-rules contain specified constants, thus making the process of identifying collapse, composition and superposition efficient.

In all our implementations, input is read from a file containing equations in a specified syntax. It is parsed and represented internally as a list of tree node pairs (representing terms with no sharing). There is a preprocessing step in the NO and DST algorithms to convert this representation into a dag and to initialize the other required data structures. In DST we construct a dag in which all vertices have outdegree at most two. The other three algorithms interleave construction of a dag with deduction steps. The published descriptions of DST and NO do not address construction of a dag. Our implementation maintains the list of terms that have been represented in the dag in a hash table and creates a new node for each term not yet represented.

The input set of equations E can be classified based on: (i) the size of the input and the number of equations, (ii) the number of equivalence classes on terms and subterms of E, and, (iii) the average number of occurrences of a constant in the set of D- and C-rules, which roughly speaking corresponds to average size of *use* lists in most of the implementations. The first set of examples are relatively simple and developed by hand to highlight strengths and weaknesses of the various algorithms. Example 11 contains five equations that induce a single equivalence class[6]. Example 12 is the same as 11, except that it contains five copies of all the equations. Example 13 requires slightly larger *use* lists[7]. Finally, example 14 consists of equations that are oriented in the "wrong" way[8].

In a first set of experiments, we assume that the input is a set of equations presented as pairs of trees (representing terms). Thus, the total running time given includes time spent on preprocessing and construction of dag (for NO and DST). In Table 1 the times shown are the averages of several runs on a Sun Ultra workstation under similar load conditions. The time was computed using the *gettimeofday* system call.

	Eqns	Vert	Class	DST	NO	SHO	COM	IND
Ex.11	5	27	1	1.286	1.640	0.281	0.606	0.409
Ex.12	20	27	1	2.912	2.772	0.794	1.858	0.901
Ex.13	12	20	6	1.255	0.733	0.515	0.325	0.323
Ex.14	34	105	2	10.556	22.488	7.275	12.077	4.416

Table 1. Total running time (in milliseconds) for Examples 11–14. *Eqns* refers to the number of equations; *Vert* to the number of vertices in the initial dag; and *Class* to the number of equivalence classes induced on the dag.

Table 2 contains similar comparisons for considerably larger examples consisting of randomly generated equations over a specified signature. The equations are obtained by fixing a signature and a bound on the depth of terms and randomly picking $2n$ terms from the set of all bounded depth terms in the given signature. We generate n equations by pairing the $2n$ terms thus obtained. The choice of signatures and depth bound was governed by the need to randomly generate interesting instances (i.e. where there are a fair number of deductions). The columns Σ_i denote the number of function symbols of arity i in the signature and d denotes the maximum term depth. The total running time includes the preprocessing time[9].

In Table 3 we show the time for computing a congruence closure assuming terms are already represented by a dag. In other words, we do not include the time it takes to create a dag. Note that we include no comparison with Shostak's method, as the dynamic construction of a dag from given term equations is inherent in this procedure. However, a comparison with a suitable strategy (in which all extension steps are applied before any deduction steps) of IND is possible. We denote by IND* indexed completion based on a strategy that first constructs a dag. The examples are the same as in Table 2.

[6] The equation set is $\{f^2(a) \approx a, f^{10}(a) \approx f^{15}(b), b \approx f^5(b), a \approx f^3(a), f^5(b) \approx b\}$.

[7] The equation set is $\{g(a,a,b) \approx f(a,b),\ gabb \approx fba,\ gaab \approx gbaa,\ gbab \approx gabb,\ gbba \approx gbab,\ gaaa \approx faa,\ a \approx c,\ c \approx d,\ d \approx e,\ b \approx c1,\ c1 \approx d1,\ d1 \approx e1\}$

[8] The set is $\{g(f^i(a), h^{10}(b)) \approx g(a,b), i = \{1, \cdots, 25\}, h^{47}(b) \approx b, b \approx h^{29}(b), h(b) \approx c0, c0 \approx c1, c1 \approx c2, c2 \approx c3, c3 \approx c4, c4 \approx a, a \approx f(a)\}$.

[9] Times for COM are not included as indexing is indispensable for larger examples.

	Eqns	Vert	Σ_0	Σ_1	Σ_2	d	Class	DST	NO	SHO	IND
Ex.21	10000	17604	2	0	2	3	7472	11.087	3.187	10.206	13.037
Ex.22	5000	4163	2	1	1	3	3	2.276	306.194	3.092	0.774
Ex.23	5000	7869	3	0	1	3	2745	2.439	1.357	3.521	3.989
Ex.24	6000	8885	3	0	1	3	9	3.551	1152.652	52.353	7.069
Ex.25	7000	9818	3	0	1	3	1	4.633	1682.815	47.755	5.471
Ex.26	5000	645	4	2	0	23	77	1.224	1.580	0.371	0.363
Ex.27	5000	1438	10	2	0	23	290	1.452	3.670	0.392	0.374

Table 2. Total running time (in seconds) for randomly generated sets of equations.

	DST	NO	IND*
Ex.21	0.919	0.296	0.076
Ex.22	0.309	319.112	1.971
Ex.23	0.241	0.166	0.030
Ex.24	0.776	1117.239	7.301

	DST	NO	IND*
Ex.25	0.958	1614.961	9.770
Ex.26	0.026	0.781	0.060
Ex.27	0.048	2.470	0.176

Table 3. Running time (in seconds) when input is in a dag form.

Several observations can be drawn from these results. First, the Nelson-Oppen procedure NO is competitive only when deduction steps are few and the number of equivalence classes is large. In logical terms, this is because it uses a non-standard deduction rule (see [5]), which may force the procedure to unnecessarily repeat the same deductions many times over in a single execution. Not surprisingly, straight-forward completion without indexing is also inefficient when many deduction steps are necessary. Indexing is of course a standard technique employed in all practical implementations of completion.

The running time of the DST procedure critically depends on the size of the hash table that contains the signatures of all vertices. If the hash table size is large, enough potential deductions can be detected in (almost) constant time. If the hash table size is reduced, to say 100, then the running time increases by a factor of up to 50. A hash table with 1000 entries was sufficient for our examples (which contained fewer than 10000 vertices). Larger tables did not improve the running times substantially.

Indexed Completion, DST and Shostak's method are roughly comparable in performance, though Shostak's algorithm has some drawbacks. For instance, equations are always oriented from left to right. In contrast, Indexed Completion always orients equations in a way so as to minimize the number of applications of the collapse rule, an idea that is also implicit in Downey, Sethi and Tarjan's algorithm. Example 12 illustrates this fact. More crucially, the manipulation of the *use* lists in Shostak's method is done in a convoluted manner due to which redundant inferences may be done when searching for the correct non-redundant ones. As a consequence, Shostak's algorithm performs poorly on instances where *use* lists are large and deduction steps are many such as in Examples 13, 24 and 25.

Finally, we note that the indexing technique used in our implementation of completion is simple—with every constant c we associate a list of D-rules that contain c as a subterm. On the other hand DST maintains at least two different ways of indexing the signatures, which makes it more efficient when the examples are large and deduction steps are plenty. On small examples, the overhead to maintain the data structures dominates. This also suggests that the use of more sophisticated indexing schemes for indexed completion might improve its performance.

5 Associative-Commutative Congruence Closure

We next consider the problem of constructing a congruence closure for a set of ground equations over a signature consisting of binary function symbols that are associative and commutative. It is not obvious how the traditional dag-based algorithms can be modified to handle associativity and commutativity of certain function symbols, though commutativity alone is easily handled by simple modifications, see comments on page 767 of [14].

Let Σ be a signature with arity function α, and E a set of ground equations over Σ. Let Σ_{AC} be some subset of Σ, containing all the associative-commutative operators. We denote by P the identities

$$f(x_1, \dots, x_k, s, y_1, \dots, y_l, t, z_1, \dots, z_m) \approx f(x_1, \dots, x_k, t, y_1, \dots, y_l, s, z_1, \dots, z_m)$$

where $f \in \Sigma_{AC}$ and $k + l + m + 2 \in \alpha(f)$; and by F the set of identities

$$f(x_1, \dots, x_m, f(y_1, \dots, y_r), z_1, \dots, z_n) \approx f(x_1, \dots, x_m, y_1, \dots, y_r, z_1, \dots, z_n)$$

where $f \in \Sigma_{AC}$ and $\{m+n+1, m+n+r, r\} \subset \alpha(f)$. The congruence induced by all ground instances of P is called a *permutation congruence*. Flattening refers to normalizing a term with respect to the set F (considered as a rewrite rule). The set $AC = F \cup P$ defines an AC-theory. The symbols in Σ_{AC} are called associative-commutative operators[10]. We require that $\alpha(f)$ be singleton for all $f \in \Sigma - \Sigma_{AC}$ and $\alpha(f) = \{2, 3, 4, \dots\}$ for all $f \in \Sigma_{AC}$.

We note that apart from the D-rules and the C-rules, in the presence of AC-symbols we additionally need A-rules.

Definition 5. *Let Σ be a signature and K be a set of constants disjoint from Σ. Equations, which when fully flattened are of the form $f(c_1, \dots, c_k) \approx f(d_1, \dots, d_l)$, where $f \in \Sigma_{AC}$, and $c_1, \cdots, c_k, d_1, \cdots, d_l \in K$, will be called A-equations. Directed A-equations are called A-rules.*

[10] The equations $F \cup P$ define a conservative extension of the theory of associativity and commutativity to varyadic terms. For a fixed arity binary function symbol, the equations $f(x, y) \approx f(y, x)$ and $f(f(x, y), z) \approx f(x, f(y, z))$ define an AC-theory.

We can now generalize all definitions made in Section 2 to the case when certain function symbols are known to be associative and commutative. By $AC\backslash R$ we denote the rewrite system consisting of all rules $u \to v$ such that $u \leftrightarrow^*_{AC} u'\sigma$ and $v = v'\sigma$, for some rule $u' \to v'$ in R and some substitution σ. We say that $AC\backslash R$ is confluent modulo AC if for all terms s, t such that $s \leftrightarrow^*_{R \cup AC} t$, there exist terms w and w' such that $s \to^*_{AC\backslash R} w \leftrightarrow^*_{AC} w' \leftarrow^*_{AC\backslash R} t$. We speak of *ground confluence* if this condition is true for all ground terms s and t. The other definitions are analogous.

Part of the condition for confluence modulo AC can be satisfied by the inclusion of so-called *extensions* of rules [23]. Given an AC-operator f and a rewrite rule $\rho : f(c_1, c_2) \to c$, we consider its extension $\rho^e : f(f(c_1, c_2), x) \to f(c, x)$. Given a set of rewrite rules R, by R^e we denote the set R plus extensions of rules in R. Extensions have to be used for rewriting terms and computing critical pairs when working with AC-symbols. The key property of extended rules is that whenever a term t is reducible by $AC\backslash R^e$ and $t \leftrightarrow^*_{AC} t'$, then t' is also reducible by $AC\backslash R^e$.

Definition 6. *Let R be a set of D-rules, C-rules and A-rules (with respect to Σ and K). We say that a constant c in K* represents *a term t in $\mathcal{T}(\Sigma \cup K)$ (via the rewrite system R) if $t \leftrightarrow^*_{AC\backslash R^e} c$. A term t is also said to be* represented *by R if it is represented by some constant via R.*

Definition 7. *Let Σ be a signature and K be a set of constants disjoint from Σ. A ground rewrite system $R = A \cup D \cup C$ is said to be an* associative-commutative congruence closure *(with respect to Σ and K) if*

(i) D is a set of D-rules, C is a set of C-rules, A is a set of A-rules, and every constant $c \in K$ represents at least one term $t \in \mathcal{T}(\Sigma)$ via R, and

(ii) $AC\backslash R^e$ is a ground convergent modulo AC over $\mathcal{T}(\Sigma \cup K)$.

In addition, if E is a set of ground equations over $\mathcal{T}(\Sigma \cup K)$ such that,

(iii) If s and t are terms over $\mathcal{T}(\Sigma)$, then $s \leftrightarrow^*_{AC \cup E} t$ if, and only if, $s \to^*_{AC\backslash R^e} \circ \leftrightarrow^*_{AC} \circ \leftarrow^*_{AC\backslash R^e} t$,

then R will be called an associative-commutative congruence closure for E.

When Σ_{AC} is empty this definition specializes to that of an abstract congruence closure in Definition 2.

For example, let Σ consist of function symbols, a, b, c, f and g, (f is AC) and let E_0 be a set of three equations $f(a, c) \approx a$, $f(c, g(f(b, c))) \approx b$ and $g(f(b, c)) \approx f(b, c)$. Using extension and orientation we can obtain a representation of the equations in E_0 using D-rules and C-rules as:

$$\begin{aligned} R_1 = \{&a \to c_1,\ b \to c_2,\ c \to c_3,\ f(c_2, c_3) \to c_4, \\ &g(c_4) \to c_5,\ f(c_1, c_3) \to c_1,\ f(c_3, c_5) \to c_2,\ c_5 \to c_4\}. \end{aligned}$$

However, the rewrite system R_1 above is not a congruence closure for E_0, as it is not a ground convergent rewrite system. But we can transform R_1 into a suitable rewrite system, using a completion-like (modulo an equational theory) process described in more detail in the next section, to obtain a congruence closure (Example 4),

$$R' = \{a \to c_1,\ b \to c_2,\ c \to c_3,\ fc_2c_3 \to c_4,\ fc_3c_4 \to c_2,\ fc_1c_3 \to c_1,$$
$$fc_2c_2 \to fc_4c_4,\ fc_1c_2 \to fc_1c_4,\ gc_4 \to c_4\}$$

that provides a more compact representation of E_0. Attempts to replace every A-rule by two D-rules (introducing a new constant in the process) leads to non-terminating derivations.

5.1 Construction of Associative-Commutative Congruence Closure

Let U be a set of symbols from which new names (constants) are chosen. We need a (partial) AC-compatible reduction ordering which orients the D-rules in the right way, and orients all the C- and A-equations. The precedence-based AC-compatible ordering $\succ$ of [25], with any precedence in which $f \succ_{\Sigma \cup U} c$, whenever $f \in \Sigma$ and $c \in U$, serves the purpose. However, much simpler partial orderings would suffice too, but for convenience we use the ordering in [25]. In our case, this simply means that, orientation of D-rules is from left to right, and the orientation of an A-rule is given by comparing the fully flattened terms as follows: $f(c_1, \ldots, c_i) \succ f(c'_1, \ldots, c'_j)$ iff either $i > j$, or $i = j$ and $\{c_1, \ldots, c_i\} \succ^{mult} \{c'_1, \ldots, c'_j\}$, i.e., if the two terms have the same number of arguments, we compare the multisets of constants using a multiset extension $\succ^{mult}$ of the precedence $\succ_{\Sigma \cup U}$, see [12].

We next present a general method for construction of associative-commutative congruence closures. Our description is fairly abstract, in terms of transition rules that operate on triples (K, E, R), where K is a set of new constants that are introduced (the original signature Σ is fixed); E is a set of ground equations (over $\Sigma \cup K$) yet to be processed; and R is a set of C-rules, D-rules and A-rules. Triples represent possible *states* in the process of constructing a closure. The initial state is $(\emptyset, E, \emptyset)$, where E is the input set of ground equations.

New constants are introduced by the following transition.

Extension: $$\frac{(K, E[t], R)}{(K \cup \{c\}, E[c], R \cup \{t \to c\})}$$

if $t \to c$ is a D-rule, $c \in U - K$, and t occurs in some equation in E that is neither an A-equation nor a D-equation.

Once a D-rule has been introduced by extension, it can be used to simplify equations.

Simplification: $$\frac{(K, E[s], R)}{(K, E[t], R)}$$

where s occurs in some equation in E, and, $s \to_{AC \backslash R^e} t$.

It is fairly easy to see that any equation in E can be transformed to a C- or an A-equation by suitable extension and simplification[11].

Equations are moved from the second to the third component of the state by orientation. All rules added to the third component are either C-rules, D-rules or A-rules.

$$\textbf{Ori}\text{entation:} \qquad \frac{(K, E \cup \{s \approx t\}, R)}{(K, E, R \cup \{s \to t\})}$$

if $s \succ t$, and, $s \to t$ is either a D-rule, or a C-rule, or a A-rule.

Deletion allows us to delete trivial equations.

$$\textbf{Del}\text{etion:} \qquad \frac{(K, E \cup \{s \approx t\}, R)}{(K, E, R)}$$

if $s \leftrightarrow^*_{AC} t$.

We consider overlaps between extensions of A-rules in ACSuperposition.

$$\textbf{ACSup}\text{erposition:} \qquad \frac{(K, E, R)}{(K, E \cup \{f(s, x\sigma) \approx f(t, y\sigma)\}, R)}$$

if $f \in \Sigma_{AC}$, there exist D- or A-rules (fully flattened as) $f(c_1, \ldots, c_k) \to s$ and $f(d_1, \ldots, d_l) \to t$ in R, substitution σ is a most-general unifier modulo AC of $f(c_1, \ldots, c_k, x)$ and $f(d_1, \ldots, d_l, y)$[12], and $\{c_1, \ldots, c_k\} \cap \{d_1, \ldots, d_l\} \neq \emptyset$.

In the special case when one multiset is contained in the other, we obtain the ACCollapse rule.

$$\textbf{ACCol}\text{lapse:} \qquad \frac{(K, E, R \cup \{t \to s\})}{(K, E \cup \{t' \approx s\}, R)}$$

if for some $u \to v \in R$, $t \to_{AC\backslash\{u \to v\}^e} t'$, and if $t \leftrightarrow^*_{AC} u$ then $s \succ v$.

In case of rules that do not contain AC-symbols at the top, superposition reduces to simplification, and hence ACCollapse also captures such superpositions. Note that we do not explicitly add AC extensions of rules to the set R. Consequently, any rule in R is either a C-rule, or a D-rule, or an A-rule, and *not* its extension. We implicitly work with extensions in ACSuperposition.

We need additional transition rules to perform simplifications on the left- and right-hand sides of other rules. The use of C-rules to simplify left-hand sides of rules is captured by ACCollapse. The simplification on the right-hand sides is subsumed by the following generalized composition rule.

$$\textbf{Com}\text{position:} \qquad \frac{(K, E, R \cup \{t \to s\})}{(K, E, R \cup \{t \to s'\})}$$

if $s \to_{AC\backslash R^e} s'$.

[11] We do not need an explicit rule for flattening as Definition 5 allows for non-flattened terms to occur in A-rules.

[12] For the special case in hand, there is exactly one most general unifier modulo AC, and it is easy to compute.

Example 4. Let $E_0 = \{f(a,c) \approx a,\ f(c, g(f(b,c))) \approx b,\ g(f(b,c)) \approx f(b,c)\}$. We show some intermediate states of a derivation below (superscripts in the last column indicate the number of applications of the respective rules). We assume that f is AC and $c_i \succ c_j$ if $i < j$.

i	Constants K_i	Equations E_i	Rules R_i	Transitions
0	$\emptyset$	E_0	$\emptyset$	
1	$\{c_1, c_3\}$	$\{fcgfbc \approx b,$ $gfbc \approx fbc\}$	$\{a \to c_1, c \to c_3,$ $fc_1c_3 \to c_1\}$	$\mathbf{Ext}^2 \cdot \mathbf{Sim} \cdot$ $\mathbf{Ori}$
2	$K_1 \cup \{c_2, c_4\}$	$\{fcgfbc \approx b\}$	$R_1 \cup \{b \to c_2,$ $fc_2c_3 \to c_4, gc_4 \to c_4\}$	$\mathbf{Sim}^2 \cdot \mathbf{Ext}^2 \cdot$ $\mathbf{Sim} \cdot \mathbf{Ori}$
3	K_2	$\emptyset$	$R_2 \cup \{fc_3c_4 \to c_2\}$	$\mathbf{Sim}^6 \cdot \mathbf{Ori}$
4	K_2	$\emptyset$	$R_3 \cup \{fc_1c_2 \to fc_1c_4\}$	$\mathbf{ACSup} \cdot \mathbf{Ori}$
5	K_2	$\emptyset$	$R_4 \cup \{fc_2c_2 \to fc_4c_4\}$	$\mathbf{ACSup} \cdot \mathbf{Ori}$

The derivation moves equations, one by one, from the second component of the state to the third component through simplification, extension and orientation. It can be verified that the set R_5 is an AC congruence closure for E_0. Note that the side-condition in extension disallows breaking of an A-rule into two D-rules, which is crucial for termination.

5.2 Termination and Correctness

Definition 8. *We use the symbol $\vdash$ to denote the one-step transition relation on states induced by the above transition rules. A derivation is a sequence of states $(K_0, E_0, R_0) \vdash (K_1, E_1, R_1) \vdash \cdots$. A derivation is said to be fair if any transition rule which is continuously enabled is eventually applied. The set R_∞ of persisting rules is defined as $\cup_i \cap_{j>i} R_j$; and similarly, $K_\infty = \cup_i \cap_{j>i} K_j$.*

We shall prove that any fair derivation will only generate finitely many persisting rewrite rules (in the third component) using Dickson's lemma [8]. Multisets over K_∞ can be compared using the multiset inclusion relation. If K_∞ is finite, this relation defines a Dickson partial order.

Lemma 6. *Let E be a finite set of ground equations. The set of persisting rules R_∞ in any fair derivation starting from state $(\emptyset, E, \emptyset)$ is finite.*

Proof. We first claim that K_∞ is finite. To see this, note that new constants are created by extension. Using finitely many applications of extension, simplification, and orientation, we can move all rules from the initial second component E of the state tuple to the third component R. Fairness ensures that this will eventually happen. Thereafter, any equations ever added to E can be oriented using orientation, hence we never apply extension subsequently (see the side condition of the extension rule). Let $K_\infty = \{c_1, \ldots, c_n\}$.

Next we claim that R_∞ is finite. Since non Σ_{AC}-symbols have fixed arities, therefore, R_∞ contains infinitely many rules with top symbol from Σ_{AC}. Since

Σ_{AC} is finite, one AC-operator, say $f \in \Sigma_{AC}$, must occur infinitely often as the top symbol in the left-hand sides of R_∞. By Dickson's lemma, there exists an infinite chain of rules (written as fully flattened for simplicity), $f(c_{11}, \ldots, c_{1k_1}) \to s_0, f(c_{21}, \ldots, c_{2k_2}) \to s_1, \ldots$, such that $\{c_{11}, \ldots, c_{1k_1}\} \subseteq \{c_{21}, \ldots, c_{2k_2}\} \subseteq \cdots$, where $\{c_{i1}, \ldots, c_{ik_i}\}$ denotes a multiset and $\subseteq$ denotes multiset inclusion. But, this contradicts fairness (in application of ACCollapse). ∎

5.3 Proof Ordering

The correctness of the procedure will be established using proof simplification techniques for associative-commutative completion, as described by Bachmair [1] and Bachmair and Dershowitz [2]. In fact, we can directly use the results and the proof measure from [2]. However, since all rules in R have a special form, we can choose a simpler proof ordering. One other difference is that we do not have explicit transition rules to create extensions of rules in the third component. Instead we use extensions of rules for simplification and computation of superpositions.

Let $s = s[u\sigma] \leftrightarrow s[v\sigma] = t$ be a proof step using the equation (rule) $u \approx v \in AC \cup E \cup R$. The complexity of this proof step is defined by

$$\begin{array}{rlrl} (\{s,t\},\bot,\bot) & \text{if } u \approx v \in E & (\{s\},\bot,t) & \text{if } u \approx v \in AC \\ (\{s\},u,t) & \text{if } u \to v \in R & (\{t\},v,s) & \text{if } v \to u \in R \end{array}$$

where $\bot$ is a new symbol. Tuples are compared lexicographically using the multiset extension of the reduction ordering $\succ$ on terms over $\Sigma \cup K_\infty$ in the first component, and the ordering $\succ$ in the second and third component. The constant $\bot$ is assumed to be minimum. The complexity of a proof is the multiset of complexities of its proof steps. The multiset extension of the ordering on tuples yields a proof ordering, denoted by the symbol $\succ_{\mathcal{P}}$. The ordering $\succ_{\mathcal{P}}$ on proofs is well founded as it is a lexicographic combination of well founded orderings.

Lemma 7. *Suppose $(K, E, R) \vdash (K', E', R')$. Then, for any two terms $s, t \in \mathcal{T}(\Sigma)$, it is the case that $s \leftrightarrow^*_{AC\cup E'\cup R'} t$ iff $s \leftrightarrow^*_{AC\cup E\cup R} t$. Further, for any $s_0, s_k \in \mathcal{T}(\Sigma \cup K)$, if π is a ground proof $s_0 \leftrightarrow s_1 \leftrightarrow \cdots \leftrightarrow s_k$ in $AC \cup E \cup R$, then there is a proof π' $s_0 = s'_0 \leftrightarrow s'_1 \leftrightarrow \cdots \leftrightarrow s'_l = s_k$ in $AC \cup E' \cup R'$ such that $\pi \succeq_{\mathcal{P}} \pi'$.*

Proof. The first part of the lemma, which states that the congruence on $\mathcal{T}(\Sigma)$ remains unchanged, is easily verified by exhaustively checking it for each transition rule. In fact, except for extension, all the other transition rules are standard rules for completion modulo a congruence, and hence the result follows. Consider the case when the state $(K' = K \cup \{c\}, E', R' = R \cup \{t \to c\})$ is obtained from the state (K, E, R) by extension. Now, if $s \leftrightarrow^*_{AC\cup E\cup R} t$, then clearly $s \leftrightarrow^*_{AC\cup E'\cup R'} t$. Conversely, if $s \leftrightarrow^*_{AC\cup E'\cup R'} t$, then we replace all occurrences of c in this proof by t to get a proof in $AC \cup E \cup R$.

For the second part, one needs to check that each equation in $(E - E') \cup (R - R')$ has a simpler proof in $E' \cup R' \cup AC$ for each transition rule application,

see [2]. In detail, we have the following cases:
(i) Extension. The proof $s[t] \leftrightarrow_E u$ is replaced by a proof $s[t] \rightarrow_{R'} s[c] \leftrightarrow_{E'} u$ and the new proof is smaller as $\{s[t], u\} \succ^m \{s[t]\}$, and $\{s[t], u\} \succ^m \{s[c], u\}$.
(ii) Simplification. The proof $r[s] \leftrightarrow_E u$ is replaced by the new proof $r[s] \leftrightarrow^*_{AC} r' \rightarrow_{R'} r[t] \leftrightarrow_{E'} u$[13]. Now, $\{r[s], u\} \succ^m \{r''\}$ for every r'' in $r[s] \leftrightarrow^*_{AC} r'$, and $\{r[s], u\} \succ^m \{r[t], u\}$.
(iii) ACCollapse. The proof $t \rightarrow_R s$ is transformed to the smaller proof $t \leftrightarrow^*_{AC} t' \rightarrow_{\{u \rightarrow v\}} t'' \leftrightarrow_{E'} s$. This new proof is smaller because the rewrite step $t \rightarrow_R s$ is more complex than (a) all proof steps in $t \leftrightarrow^*_{AC} t'$ (in the second component), (b) the proof step $t' \rightarrow_{\{u \rightarrow v\}} t''$ in the second component if $t \not\leftrightarrow^*_{AC} u$, and in the third component if $t \leftrightarrow^*_{AC} u$ (see side condition in ACCollapse); and, (c) the proof step $t'' \leftrightarrow_{E'} s$ (in the first component).
(iv) Orientation. In this case, $s \leftrightarrow_E t$ is more complex than the new proof $s \rightarrow_{R'} t$ follows from $\{s, t\} \succ^m \{s\}$.
(v) Deletion. We have $s \leftrightarrow_E t$ more complex than $s \leftrightarrow^*_{AC} t$ because $\{s, t\} \succ^m \{s'\}$ for every s' in $s \leftrightarrow^*_{AC} t$.
(vi) Composition. We have the proof $t \rightarrow_R s$ transformed to the smaller proof $t \rightarrow_R s' \leftarrow_{R'} s'' \leftrightarrow^*_{AC} s$. This new proof is smaller because the rewrite step $t \rightarrow_R s$ is more complex than (a) the rewrite step $t \rightarrow_{R'} s'$ in the third component, (b) all proof steps in $s'' \leftrightarrow^*_{AC} s$ in the first component, and (c) the rewrite step $s'' \rightarrow_{R'} s'$ in the first component.

The ACSuperposition transition rule does not delete any equation. This completes the proof of the lemma. ∎

Note that in any derivation, extensions of rules are not added explicitly, and hence, they are never deleted either. Once we converge to R_∞, we introduce extensions to take care of cliffs in proofs.

Lemma 8. *If R_∞ is a set of persisting rules of a fair derivation starting from the state $(\emptyset, E, \emptyset)$, then, R^e_∞ is a ground convergent (modulo AC) rewrite system. Furthermore, $E_\infty = \emptyset$.*

Proof. Fairness implies that all critical pairs (modulo AC) between rules in R^e_∞ are contained in the set $\cup_i E_i$. Since a fair derivation is non failing, $E_\infty = \emptyset$. Since the proof ordering is well-founded, for every proof in $E_i \cup R_i \cup AC$, there exists a minimal proof π in $E_\infty \cup R_\infty \cup AC$. We argue by contradiction that certain proof patterns can not occur in the minimal proof π: specifically, there can be no peaks $s \leftarrow_{R^e_\infty} u \rightarrow_{AC \setminus R^e_\infty}$, non-overlap cliffs or variable overlap cliffs.

(i) Peaks. A peak caused by a non-overlap or a variable overlap $s \leftarrow_{R^e_\infty} u \rightarrow_{AC \setminus R^e_\infty} t$ can be transformed to a simpler proof $s \rightarrow^*_{AC \setminus R^e_\infty} v \leftarrow^*_{AC \setminus R^e_\infty} t$. The new proof is simpler because u is bigger than each term in the new proof. Next suppose that the above pattern is caused by a proper overlap. By the Extended Critical Pair Lemma (see [2]), $s \leftrightarrow^*_{AC} s' \leftrightarrow_{CP_{AC}(R^e_\infty)} t' \leftrightarrow^*_{AC} t$, where

[13] Note that we used extended rule in the specifying simplification, but for purposes of proof transformations, we only consider the original (non-extended) rules as being present in the third component.

$CP_{AC}(R^e_\infty)$ denotes the set of all critical pairs between rules in R^e_∞ modulo AC. Call this proof π. Since by fairness $CP_{AC}(R^e_\infty) \subseteq \cup_k E_k$, there is a proof $s \leftrightarrow^*_{AC} s' \leftrightarrow_{E_k} t' \leftrightarrow^*_{AC} t$ for some $k \geq 0$. Using Lemma 7, we may infer that there is a proof π' in $AC \cup R_\infty$ such that π' is strictly smaller than π, a contradiction.

(ii) Cliffs. A non-overlap cliff $w[v,s] \leftrightarrow_{AC} w[u,s] \rightarrow_{AC\backslash R^e_\infty} w[u,t]$ can be transformed to the following less complex proof: $w[v,s] \rightarrow_{AC\backslash R^e_\infty} w[v,t] \leftrightarrow_{AC} w[u,t]$. Clearly, $w[v,s] \succ w[v,t]$ and hence the proof $w[v,t] \leftrightarrow_{AC} w[u,t]$ is smaller than the proof $w[v,s] \leftrightarrow_{AC} w[u,s]$ (in the first component). The complexity of the proof $w[u,s] \rightarrow_{AC\backslash R^e_\infty} w[u,t]$ is identical to the complexity of the proof $w[v,s] \rightarrow_{AC\backslash R^e_\infty} w[v,t]$.

In case of AC, a variable overlap cliff $s \leftrightarrow_{AC} u \rightarrow_{AC\backslash R^e_\infty} t$ can be eliminated in favour of the proof $s \rightarrow_{AC\backslash R^e_\infty} t' \leftrightarrow_{AC} t$. Note that the proof $u \rightarrow_{AC\backslash R^e_\infty} t$ and the proof $s \rightarrow_{AC\backslash R^e_\infty} t'$ are of the same complexity, and additionally the proof $s \leftrightarrow_{AC} u$ is larger than the proof $t' \leftrightarrow_{AC} t$ as all terms in the latter proof are smaller than u.

In summary, the proof π can not contain peaks $s \leftarrow_{R^e_\infty} u \rightarrow_{AC\backslash R^e_\infty}$, or, non-overlap or variable overlap cliffs $s \leftrightarrow_{AC} u \rightarrow_{AC\backslash R^e_\infty} t$. The cliffs arising from proper overlaps can be replaced by extended rules, as $(R^e_\infty)^e = R^e_\infty$. The minimal proof π in $R_\infty \cup AC$ can, therefore, only be of the form $s \rightarrow^*_{AC\backslash R^e_\infty} s' \leftrightarrow^*_{AC} t' \leftarrow^*_{AC\backslash R^e_\infty} t$; which is a rewrite proof. ∎

Note that we did not define the proof complexities for the extended rules as the extended rules are introduced only in the end. Hence, the argument given here is not identical to the one in [2], though it is similar. Using Lemmas 7 and 8, we can easily prove the following.

Theorem 5. *Let R_∞ be the set of persisting rules of a fair derivation starting from state $(\emptyset, E, \emptyset)$. Then, the set R^e_∞ is an associative-commutative congruence closure for E.*

Proof. In order to show that R_∞ is an associative-commutative congruence closure for E_0, we need to prove the three conditions in Definition 7.

1. Orientation rule ensures that R_∞ consists of only D-rules, C-rules, and A-rules. Let c be any constant in K_∞. Since all constants are added by extension, let $f(c_1, \ldots, c_k) \rightarrow c$ be the rule introduced by extension when c was added. By induction hypothesis we can assume that all constants added before c represent a term in $\mathcal{T}(\Sigma)$ via R_∞. Therefore, there exist terms $s_1, \ldots, s_k \in \mathcal{T}(\Sigma)$ such that $s_i \leftrightarrow^*_{AC\backslash R^e_\infty} c_i$ and hence,
$$f(s_1, \ldots, s_k) \leftrightarrow^*_{AC\backslash R^e_\infty} f(c_1, \ldots, c_k) \rightarrow_{\cup_i R_i} c.$$
Using Lemma 7, $f(s_1, \ldots, s_k) \leftrightarrow^*_{R^e_\infty \cup E_\infty \cup AC} c$. Lemma 8 shows that $E_\infty = \emptyset$, and $f(s_1, \ldots, s_k) \leftrightarrow^*_{AC\backslash R^e_\infty} c$.
2. Lemma 8 shows that $AC\backslash R^e_\infty$ is convergent.
3. Let $s, t \in \mathcal{T}(\Sigma)$. Using Lemma 7, we know $s \leftrightarrow^*_{E\cup AC} t$ iff $s \leftrightarrow^*_{E_\infty \cup R_\infty \cup AC} t$. Since $E_\infty = \emptyset$, Lemma 8 implies that $s \rightarrow^*_{AC\backslash R^e_\infty} \circ \leftrightarrow^*_{AC} \circ \leftarrow^*_{AC\backslash R^e_\infty} t$. ∎

Since R_∞ is finite, there exists a k such that $R_\infty \subset R_k$. Though there exist non-terminating fair derivations, we can still identify the finite set of persisting rules if we eagerly apply ACCollapse before any application of ACSuperposition at any point after stage k.

5.4 Optimizations

The set of transition rules for computing an AC congruence closure can be further enhanced by additional simplifications and optimizations. First, we can flatten terms in E.

$$\textbf{Fla}\text{ttening:} \qquad \frac{(K, E \cup \{s \approx t\}, R)}{(K, E \cup \{u \approx t\}, R)}$$

where $s \to_F u$.

However, now the correctness proof given above, in particular Lemma 7, fails as the new proof $s \leftrightarrow_{AC} u \leftrightarrow_{E'} t$ of the deleted equation $s \approx t$ is larger than the old proof $s \leftrightarrow_{E'} t$. But we can still establish the correctness of the extended set of inference rules as follows: Assume that flattening does not delete the equation $s \approx t$ from E but only marks it. All subsequent derivation steps do not work on the marked equations. Once the derivation converges (ignoring the marked equations), we can delete the marked equations as any such equation, say $s \approx t$, would have a proof $s \leftrightarrow_{AC} u \leftrightarrow_{AC \cup R_\infty} t$, and hence also a desired rewrite proof (using the persisting set of rewrite rules).

As a consequence of the flattening rule, we can construct fully flattened AC congruence closures, i.e., where each term in the congruence closure is fully flattened.

Secondly, a lot of inferences can be avoided if we note that we do not need to consider extensions of all rules that have an AC symbol on top of their left-hand sides (for the purpose of critical pair computations). For example, we need not consider extensions of those D-rules that are created by extension to name a *proper subterm* in E. This fact can be easily incorporated using constraints, but we chose not to show it here for the sake of simplicity and clarity. Rather than formalizing this completely here, we illustrate this with an example below.

Example 5. Suppose that the initial set of equations E_0 is: $\{f(a,b) \approx b,\ g(b) \approx f(a,a),\ f(b, g(f(b, g(f(a,a,b))))) \approx c\}$. Applying extension and simplification until all the initial equations can be oriented using orientation, we generate the following set of eleven D-rules:

$$\begin{array}{rrrr}
a \to c_1 & b \to c_2 & fc_1c_2 \to c_2 & fc_1c_1 \to c_3 \\
gc_2 \to c_3 & fc_3c_2 \to c_4 & gc_4 \to c_5 & fc_2c_5 \to c_6 \\
gc_6 \to c_7 & fc_2c_7 \to c_8 & c \to c_8 &
\end{array}$$

Now, note the huge number of possible ACSuperpositions. However, we need not consider all of them. In particular, we need to consider extensions only of three rules: $fc_1c_2 \to c_2$, $fc_1c_1 \to c_3$, and $fc_2c_7 \to c_8$, and hence, AC-Superpositions between only these.

In order to express this optimization, we constrain the extension variable to disallow certain instantiations. These additional constraints can be carried to new equations generated in a deduction step. Moreover, additional constraints can also be introduced during deduction. For example, after deducing the critical pair $fc_1c_2 \approx fc_2c_3$ that arises by overlapping the (extensions of) rules $fc_1c_2 \to c_2$ and $fc_1c_1 \to c_3$, we can delete the instance $fc_1c_1c_2 \to fc_2c_3$ of the latter rule (by adding a suitable constraint) as it has a smaller proof $fc_1c_1c_2 \to fc_1c_2 \approx fc_2c_3$ using the deduced equation.

Finally, note that, as in the case of congruence closure discussed before, we can choose the ordering between two constants in K on the fly. As an optimization we could always choose it in a way so as to minimize the applications of ACCollapse and composition later. In other words, when we need to choose the orientation for $c \approx d$, we can count the number of occurrences of c and d in the set of D- and A-rules (in the R-component of the state), and the constant with fewer occurrences is made larger.

5.5 Properties

Results in the previous sections lead to decidability of the word problem for ground theories presented over a signature containing finitely many associative-commutative symbols. This is done by decomposing the equations (over a signature consisting of several symbols) into equations over exactly one function symbol and a set of new constants. A set of equations over exactly one AC symbol and finitely many constants defines a finitely presented commutative semigroup.

The word problem for commutative semigroups is known to be complete for deterministic *EXP* space [9]. It is a simple observation that the word problem for commutative semigroups can be reduced to the ideal membership problem for binomial ideals. In fact, an optimal exponential space algorithm for generating the reduced Gröbner basis of binomial ideals was presented in [18], but that algorithm was not based on critical pair completion.

Thus, using the approach proposed in this paper, we can construct AC congruence closure in time $O(n|\Sigma|T(n))$ and space $O(n^2+S(n))$ using an algorithm for constructing Gröbner basis for binomial ideals that runs in time $O(T(n))$ and space $S(n)$. We have not worked out the time complexity of the critical pair completion based algorithm (as presented in this paper) for constructing Gröbner bases for binomial ideals and that remains as future work.

6 Construction of Ground Convergent Rewrite Systems

We have presented transition rules for constructing a convergent presentation in an extended signature for a set of ground equations. We next discuss the problem of obtaining a ground convergent (AC) rewrite system for the given ground (AC-) theory *in the original signature*. Hence, now we focus our attention to the problem of transforming a convergent system over an extended signature to a convergent system in the original signature.

The basic idea of transforming back is elimination of constants from the presentation R as follows: (i) if a constant c is not redundant (Definition 3), then we pick a term $t \in \mathcal{T}(\Sigma)$ that is represented by c, and replace all occurrences of c by t in R; (ii) if a constant c is redundant (and say $c \to d$ is a C-rule in which c occurs as the left-hand side term), then all occurrences of c can be replaced by d in R.

In the case when there are no AC-symbols in the signature, the above method generates a ground convergent system from any given abstract congruence closure. This gives an indirect way to construct ground convergent systems equivalent to a given set of ground equations. However, we run into problems when we use the same method for translation in presence of AC-symbols. Typically, after translating back, the set of rules obtained is usually non-terminating modulo AC (Example 6). But if we suitably define the notion of AC-rewriting, the rules are seen to be convergent in the new definition. This is useful in two ways: (i) the new notion of AC-rewriting seems to be more practical, in the sense that it involves strictly less work than a usual $AC\backslash R^e$ reduction; and, (ii) it helps to clarify the advantage offered by the use of extended signatures when dealing with a set of ground equations over a signature containing associative and commutative symbols.

6.1 Transition Rules

We describe the process of transforming a rewrite system over an extended signature $\Sigma \cup K$ to a rewrite system over the original signature Σ by transformation rules on states (K, R), where K is the set of constants to be eliminated, and R is a set of rewrite rules over $\Sigma \cup K$ to be transformed.

Redundant constants can be easily eliminated by the compression rule.

Compression: $$\frac{(K \cup \{c\}, R \cup \{c \to t\})}{(K, R\langle c \mapsto t\rangle)}$$

where $\langle c \mapsto t\rangle$ denotes the (homomorphic extension of the) mapping $c \mapsto t$, and $R\langle c \mapsto t\rangle$ denotes the application of this homomorphism to each term in the set R.

The basic idea for eliminating a constant c that is not redundant in R involves picking a representative term t (over the signature Σ) in the equivalence class of c, and replacing c by t everywhere in R.

Selection: $$\frac{(K \cup \{c\}, R \cup \{t \to c\})}{(K, R\langle c \mapsto t\rangle \cup R')}$$

if (i) c is not redundant in R, (ii) $t \in \mathcal{T}(\Sigma)$, and (iii) if $t \equiv f(t_1, \ldots, t_k)$ with $f \in \Sigma_{AC}$ then $R' = \{f(t_1, \ldots, t_k, X) \to f(f(t_1, \ldots, t_k), X)\}$, otherwise $R' = \emptyset$.

In case $\Sigma_{AC} = \emptyset$, we note that R' will always be empty. We also require that terms are *not* flattened after application of mapping $R\langle c \mapsto t\rangle$. The variable X is a special sequence variable which can only be instantiated by non-empty sequences. We shall formally define its role later.

Example 6. Consider the problem of constructing a ground convergent system for the set E_0 from Example 4. A fully-reduced congruence closure for E_0 is given by the set R_0

$$\begin{array}{cccc} a \to c_1 & b \to c_2 & c \to c_3 & fc_2c_3 \to c_4 \\ fc_3c_4 \to c_2 & fc_1c_3 \to c_1 & fc_2c_2 \to fc_4c_4 & fc_1c_2 \to fc_1c_4 \\ gc_4 \to c_4 & & & \end{array}$$

under the ordering $c_2 \succ c_4$ between constants. For the constants c_1, c_2 and c_3 we have no choice but to choose a, b and c as representatives respectively. Thus after three applications of selection, we get

$$\begin{array}{cccc} fcc_4 \to b & fac \to a & fbb \to fc_4c_4 & fab \to fac_4 \\ fbc \to c_4 & gc_4 \to c_4 & & \end{array}$$

Next we are forced to choose fbc as the representative for the class c_4. This gives us the transformed set R_1,

$$\begin{array}{cccc} fc(fbc) \to b & fac \to a & fbb \to f(fbc)(fbc) & fab \to fa(fbc) \\ fbcX \to f(fbc)X & gfbc \to fbc & & \end{array}$$

The relation $\to_{AC\backslash R_1^e}$ is clearly non-terminating (with the variable X considered as a regular term variable).

6.2 Rewriting with Sequence Extensions modulo Permutation Congruence

Let X denote a variable ranging over non-empty sequences of terms. A sequence substitution σ is a substitution that maps variables to the sequences. If σ is a sequence substitution that maps X to the sequence $\langle s'_1, \ldots, s'_m\rangle$, then $f(s_1, \ldots, s_k, X)\sigma$ is the term $f(s_1, \ldots, s_k, s'_1, \ldots, s'_m)$.

Definition 9. *Let ρ be a ground rule of the form $f(t_1, \ldots, t_k) \to g(s_1, \ldots, s_m)$ where $f \in \Sigma_{AC}$. We define the* sequence extension *ρ^s of ρ as $f(t_1, \ldots, t_k, X) \to f(s_1, \ldots, s_m, X)$ if $f = g$, and as $f(t_1, \ldots, t_k, X) \to f(g(s_1, \ldots, s_m), X)$ if $f \neq g$.*

Now we are ready to define the notion of rewriting we use. Recall that P denotes the equations defining the permutation congruence, and that $AC = F \cup P$. Given a set R, we denote by R^s the set R plus sequence extensions of all ground rules in R.

Definition 10. *Let R be a set of rewrite rules. For ground terms $s, t \in \mathcal{T}(\Sigma)$, we say that $s \to_{P\backslash R^s} t$ if there exists a rule $l \to r \in R^s$ and a sequence substitution σ such that $s = C[l']$, $l' \leftrightarrow_P^* l\sigma$, $r' = r\sigma$, and $t = C[r']$.*

Note that the difference with standard rewriting modulo AC is that instead of performing matching modulo AC, we do matching modulo P. For example, if ρ is $fac \to a$, then the term $f(f(a,b),c)$ is not reducible by $\to_{P\backslash\rho^s}$, although it is reducible by $\to_{AC\backslash\rho^e}$. The term $f(f(a,b),c,a)$ can be rewritten by $\to_{P\backslash\rho^s}$ to $f(f(a,b),a)$.

Example 7. Following up on Example 6, we note that the relation $P\backslash R_1^s$ is convergent. For instance, a normalizing rewrite derivation for the term $fabc$ is,

$$fabc \to_{P\backslash R_1^s} fa(fbc)c \to_{P\backslash R_1^s} fab \to_{P\backslash R_1^s} fa(fbc).$$

On closer inspection, we find that we are essentially doing a derivation in the original rewrite system R_0 (over the extended signature),

$$fc_1c_2c_3 \to_{P\backslash R_0^s} fc_1c_4c_3 \to_{P\backslash R_0^s} fc_1c_2 \to_{P\backslash R_0^s} fc_1c_4.$$

A $P\backslash R_0^s$ proof step can be projected onto a $P\backslash R_1^s$ proof step, see Lemma 9(a) and Lemma 10(a). This is at the core of the proof of correctness, see Theorem 6.

6.3 Correctness

We shall prove that compression and selection transform a fully flattened AC congruence closure over $\Sigma \cup K$ into a rewrite system R over Σ which is convergent modulo P and which defines the same equational theory over fully flattened terms over Σ. First note that any derivation starting from the state (K, R), where R is an AC congruence closure over Σ and K, is finite. This is because K is finite, and each application of compression and selection reduces the cardinality of K by one. Furthermore, in any intermediate state (K, R), R is always a rewrite system over $\Sigma \cup K$. Hence, in the final state (K_∞, R_∞), if $K_\infty = \emptyset$, then, R_∞ is a rewrite system over Σ, the original signature. We will show that K_∞ is actually empty, and that the reduction relation $P\backslash R_\infty^s$ is terminating on $\mathcal{T}(\Sigma)$ and confluent on fully flattened terms in $\mathcal{T}(\Sigma)$.

In this section, we say R is left-reduced (modulo P) if every left-hand side of any rule in R is irreducible by $P\backslash\rho$ and $P\backslash\rho^s$ for every other rule ρ in R; and, R is terminating (modulo P) if $P\backslash R^s$ is.

Lemma 9. *Let $(K_1, R_1 = R_0'\sigma)$ be obtained from $(K_0 = K_1 \cup \{c\}, R_0 = R_0' \cup \{c \to u\})$ using compression, where $\sigma = \langle c \mapsto u\rangle$. Assume that the rewrite system R_0 is left-reduced and terminating. Then,*

(a) For any two terms $s, t \in \mathcal{T}(\Sigma \cup K_0)$, if $s \to_{P\backslash R_0^s} t$, then $s\sigma \to_{P\backslash R_1^s}^{0,1} t\sigma$.
(b) For any two terms $s, t \in \mathcal{T}(\Sigma \cup K_1)$, if $s \to_{P\backslash R_1^s} t$, then $s\theta \leftrightarrow_{P\backslash R_0^s}^{+} t\theta$, where $\theta = \langle u \mapsto c\rangle$[14].
(c) R_1 is left-reduced and terminating.

[14] Note that if θ is defined by $\langle fab \mapsto c_0\rangle$, then $fabc\theta = fabc$, but $f(fab)c\theta = fc_0c$.

Proof. To prove (a), let s, t be two terms over $\Sigma \cup K_0$ such that $s = C[l_0\sigma^s]$ and $t = C[r_0\sigma^s]$, where $l_0 \to r_0$ is (a sequence extension of) some rule in R_0. Clearly, $s\sigma = (C[l_0\sigma^s])\sigma = (C\sigma)[(l_0\sigma)(\sigma^s\sigma)] = (C\sigma)[l_1\sigma^s\sigma]$, and similarly $t\sigma = (C[r_0\sigma^s])\sigma = (C\sigma)[(r_0\sigma)(\sigma^s\sigma)] = (C\sigma)[r_1\sigma^s\sigma]$, where either $l_1 = r_1$, or, $l_1 \to r_1$ is (a sequence extension of) some rule in R_1. In the first case, $s\sigma = t\sigma$ and in the second case, $s\sigma \to_{P\backslash R_1^s} t\sigma$.

To prove (b), note that since R_0 is left-reduced, a compression step has the same effect as a sequence of composition steps followed by deletion of a rule. Hence, if $s \to_{R_1^s} t$, then $s \leftrightarrow^+_{R_0^s} t$. Therefore, $s\theta \to^*_{\{c\to u\}} s \leftrightarrow^+_{R_0} t \leftarrow^*_{\{c\to u\}} t\theta$.

To prove (c), note that termination is preserved by composition and deletion. Furthermore, the left-hand side terms do not change, and hence the system continues to remain left-reduced. ∎

Lemma 10. *Let $(K_1, R_1 = R_0'\sigma \cup R')$ be obtained from $(K_0 = K_1 \cup \{c\}, R_0 = R_0' \cup \{u \to c\})$ using selection, where $\sigma = \langle c \mapsto u\rangle$. Assume that the rewrite system R_0 is left-reduced and terminating. Then,*

(a) For any two terms $s, t \in \mathcal{T}(\Sigma \cup K_0)$, if $s \to_{P\backslash R_0^s} t$, then $s\sigma \to^{0,1}_{P\backslash R_1^s} t\sigma$.
(b) For any two terms $s, t \in \mathcal{T}(\Sigma \cup K_1)$, if $s \to_{P\backslash R_1^s} t$, then $s\theta \to^+_{P\backslash R_0^s} t\theta$, where $\theta = \langle u \mapsto c\rangle$.
(c) R_1 is left-reduced and terminating.

Proof. The proof of (a) is identical to the proof of Lemma 9(a). Note that in the case when $u = f(u_1, \ldots, u_k)$, where $f \in \Sigma_{AC}$, $s = C[f(u_1, \ldots, u_k, X\sigma^s)]$, and $t = C[f(c, X\sigma^s)]$, the proof

$$s\sigma = (C\sigma)[f(u_1, \ldots, u_k, X\sigma^s\sigma)] \to_{P\backslash R_1^s} (C\sigma)[f(f(u_1, \ldots, u_k), X\sigma^s\sigma)] = t\sigma$$

uses the rule in the set R'.

To prove (b), let s, t be two terms over $\Sigma \cup K_1$ such that $s = C[l_1\sigma^s]$ and $t = C[r_1\sigma^s]$, where $l_1 \to r_1$ is (a sequence extension of) some rule in R_1. First consider the case when $l_1 = f(u_1, \ldots, u_k, X) \to f(f(u_1, \ldots, u_k), X) = r_1$ is the rule in R', and since $X\sigma^s$ is non-empty,

$$s\theta = (C\theta)[f(u_1, \ldots, u_k, X\sigma^s\theta)] \to_{P\backslash R_0^s} (C\theta)[f(c, X\sigma^s\theta)] = t\theta.$$

In the other case, assume $l_1 = l_0\sigma$ and $r_1 = r_0\sigma$, where $l_0 \to r_0$ is (an extension of) some rule different from $u \to c$ in R_0. Since R_0 is left-reduced modulo P, $s\theta = (C[(l_0\sigma)\sigma^s])\theta = (C\theta)[l_0(\sigma^s\theta)]$, and therefore we have,

$$s\theta = (C\theta)[l_0(\sigma^s\theta)] \to_{R_0} (C\theta)[r_0(\sigma^s\theta)] \to^*_{\{u\to c\}} (C[(r_0\sigma)\sigma^s])\theta = t\theta.$$

Since R_0 is terminating, it follows from (b) that R_1 is also terminating. Finally, to prove that $R_1 = R_0'\sigma \cup R'$ is left-reduced, note that $R_0'\sigma$ is left-reduced because R_0 is. Furthermore, Condition (i) in selection and the fact that R_0 is left-reduced together imply that $R_0'\sigma \cup R'$ is left-reduced too. ∎

The second step in the correctness argument involves showing that if $K_i \neq \emptyset$, then we can always apply either selection or compression to get to a new state.

Lemma 11. *Let (K_i, R_i) be a state in the derivation starting from (K_0, R_0), where $R_0 = D_0 \cup C_0 \cup A_0$ is a left-reduced (modulo AC) associative-commutative congruence closure over the signature $\Sigma \cup K_0$. Assume that for every constant c in K_0, there exists a term t in $\mathcal{T}(\Sigma)$ such that*[15] $t \rightarrow^*_{D_0/C_0} c$[16]*. If $K_i \neq \emptyset$, then either selection or compression is applicable to the state (K_i, R_i).*

Proof. Since $K_i \neq \emptyset$, let c be some constant in K_i. By assumption c represents some term $t \in \mathcal{T}(\Sigma)$ such that $t \rightarrow^*_{D_0/C_0} c$ [17]. It follows from convergence of $AC\backslash R_0$ that,

$$t \rightarrow^*_{D_0 \cup C_0} c' \leftarrow^*_{C_0} c.$$

Since R_0 is a left-reduced (modulo AC) congruence closure, therefore R_0 is left-reduced and terminating modulo P, and hence Lemma 9 and Lemma 10 are applicable. As none of the constants in $K_0 - K_i$ occur in the terms t and c, using Lemma 9(a) and Lemma 10(a), we have

$$t \rightarrow^*_{P\backslash R_i^s} \circ \leftarrow^*_{P\backslash R_i^s} c,$$

where the right-hand side of each rule used in the above proof is either a constant or a term in $\mathcal{T}(\Sigma)$. If it is the case that c is reducible by R_i, then c is a redundant constant which can be eliminated by compression. If there are no redundant constants, then the above proof is of the form $t \rightarrow^*_{P\backslash R_i^s} c$. If $l \rightarrow d \in R_i^s$ is the first rule used in the above proof that has a constant as a right-hand side, then we can choose l as the representative for d and hence selection is applicable. ∎

Theorem 6. *If (K_∞, R_∞) is the final state of a maximal derivation starting from state (K, R), where R is a left-reduced fully flattened AC congruence closure such that for every constant c in K_0, there exists a term t in $\mathcal{T}(\Sigma)$ such that $t \rightarrow^*_{D_0/C_0} c$, then (i) $K_\infty = \emptyset$, (ii) $\rightarrow_{P\backslash R^s_\infty}$ is ground convergent on all fully flattened terms over Σ, and (iii) the equivalence over flattened $\mathcal{T}(\Sigma)$ terms defined by this relation is the same as the equational theory induced by $R \cup AC$ over flattened $\mathcal{T}(\Sigma)$ terms.*

Proof. Statement (i) is a consequence of Lemma 11. It follows from Lemma 9(c) and Lemma 10(c) that $\rightarrow_{P\backslash R^s_\infty}$ is terminating. Let s, t be fully flattened terms over $\mathcal{T}(\Sigma)$ such that $s \leftrightarrow^*_{P\cup R^s_\infty} t$. Using Lemma 9($b$) and Lemma 10($b$), it follows that $s \leftrightarrow^*_{AC\cup R} t$. This implies that $s \rightarrow^*_{AC\backslash R^e} \circ \leftrightarrow^*_{AC} \circ \leftarrow^*_{AC\backslash R^e} t$, and hence, by projecting this proof onto fully flattened terms (normalize each term in the proof by F), we obtain a proof $s \rightarrow^*_{P\backslash R^s} \circ \leftrightarrow^*_P \circ \leftarrow^*_{P\backslash R^s} t$, as R is assumed to be fully flattened. Using Lemma 9(a) and Lemma 10(a), this normal form proof can be projected onto a proof $s \rightarrow^*_{P\backslash R^s_\infty} \circ \leftrightarrow^*_P \circ \leftarrow^*_{P\backslash R^s_\infty} t$. This establishes claims (ii) and (iii). ∎

[15] $\rightarrow_{D/C} = (\leftrightarrow^*_C \circ \rightarrow_D \circ \leftrightarrow^*_C)$.

[16] If $\Sigma_{AC} = \emptyset$, then this condition is satisfied by any abstract congruence closure.

[17] Note that if the non-extended form of a A-rule is a D-rule, it is included in the set D_0.

Note that in the special case when Σ_{AC} is empty, the notion of rewriting corresponds to the standard notion, and hence R_∞ is convergent in the standard sense by this theorem.

7 Conclusion

Abstract Congruence Closure

Kapur [17] considered the problem of casting Shostak's congruence closure [27] algorithm in the framework of ground completion on rewrite rules. Our work has been motivated by the goal of formalizing not just one, but several congruence closure algorithms, so as to be able to better compare and analyze them.

We have suggested that, abstractly, congruence closure can be *defined* as a ground convergent system; and that this definition does not restrict the applicability of congruence closure. We give strong bounds on the length of derivations used to construct an abstract congruence closure. This brings out a relationship between derivation lengths and term orderings used in the derivation. The rule-based abstract description of the logical aspects of the various published congruence closure algorithms leads to a better understanding of these methods. It explains the observed behaviour of implementations and also allows one to identify weaknesses in specific algorithms.

The paper also illustrates the use of an extended signature as a formalism to model and subsequently reason about data structures like the term dags, which are based on the idea of structure sharing. This insight is more generally applicable to other algorithms as well [6].

Efficient Construction of Ground Convergent Systems

Graph-based congruence closure algorithms have also been used to construct a convergent set of ground rewrite rules in polynomial time by Snyder [28]. Plaisted et. al. [24] gave a *direct* method, not based on using congruence closure, for completing a ground rewrite system in polynomial time. Hence our work completes the missing link, by showing that congruence closure is nothing but ground completion.

Snyder [28] uses a *particular implementation* of congruence closure due to which some postprocessing followed by a *second* run of congruence closure is required. We, on the other hand, work with abstract congruence closure and are free to choose any implementation. All the steps in the algorithm in [28] can be described using our *construction of abstract congruence closure* steps, and the final output of Snyder's algorithm corresponds to an abstract congruence closure. The compression and selection rules for *translating back* in our work, actually correspond to what Snyder calls *printing-out the reduced system* and this is not included in the algorithms time complexity of $O(n \log(n))$ as computed in [28]. Finally, the approach in [28] is to solve the problem "by abandoning rewriting techniques altogether and recasting problem in graph theoretic terms." On the other hand, we stick to rewriting over extensions.

Plaisted and Sattler-Klein [24] show that ground term-rewriting systems can be completed in a polynomial number of rewriting steps by using an appropriate data structure for terms and processing the rules in a certain way. Our work describes the construction of ground convergent systems using congruence closure as completion with *extensions*, followed by a translating back phase. Plaisted and Sattler-Klein prove a quadratic time complexity of their completion procedure.

AC Congruence Closure

The fact that we can construct an AC congruence closure implies that the word problem for finitely presented ground AC-theories is decidable, see [19], [21] and [13]. Note that we arrive at this result *without* assuming the existence of an AC-simplification ordering that is total on ground terms. The existence of such AC-simplification orderings was established in [21], but required a non-trivial proof.

Since we construct a convergent rewrite system, even the problem of determining whether two finitely presented ground AC-theories are equivalent, is decidable. Since commutative semigroups are special kinds of AC-theories, where the signature consists of a single AC-symbol and a finite set of constants, these results carry over to this special case [20, 18].

The idea of using variable abstraction to transform a set of equations over several AC-symbols into a set of equations in which each equation contains exactly one AC-symbol appears in [13]. All equations containing the same AC-symbol are separated out, and completed into a canonical rewriting system (modulo AC) using the method proposed in [7]. However, the combination of ground AC-theories with other ground theories is done differently here. In [13], the ground theory (non-AC part) is handled using ground completion (and uses a recursive path ordering during completion). We, on the other hand, use a congruence closure. The usefulness of our approach can also be seen from the simplicity of the correctness proof and the results we obtain for transforming a convergent system over an extended signature to one over the original signature.

The method for completing a finitely presented commutative semigroup (using what we call A-rules here) has been described in various forms in the literature, e.g. [7])[18]. It is essentially a specialization of Buchberger's algorithm for polynomial ideals to the case of binomial ideals (i.e. when the ideal is defined by polynomials consisting of exactly two monomials with coefficients $+1$ and -1).

The basic idea behind our construction of associative-commutative congruence closure is that we consider only certain ground instantiations of the non-ground AC axioms. If we are interested in the $\mathcal{E}$-algebra presented by E (where $\mathcal{E}$ consists of only AC axioms for some function symbols in the signature Σ in our

[18] Actually there is a subtle difference between the proposed method here and the various other algorithms for deciding the word problem for commutative semigroups too. For example, working with rule extensions is not the same as working with rules on equivalence classes (under AC) of terms. Hence, in our method, we can apply certain optimizations as mentioned in Section 5.4.

case, and E is a set of ground equations), then since $\mathcal{E}$ consists of non-ground axioms, one needs to worry about what instantiations of these axioms to consider. For the case when $\mathcal{E}$ is a set of AC axioms, we show that we need to consider ground instances in which every variable is replaced by some subterm occurring in E. This observation can be generalized and one can ask for what choices of $\mathcal{E}$ axioms does considering such restricted instantiations suffice to decide the word problem in $\mathcal{E}$-algebras? Evans [15, 16] gives a characterization in terms of *embeddability of partial $\mathcal{E}$-algebras*. Apart from commutative semigroups, this method works for lattices, groupoids, quasigroups, loops, etc.

Acknowledgements. We would like to thank Deepak Kapur and Rakesh Verma for their helpful comments.

References

[1] L. Bachmair. *Canonical Equational Proofs*. Birkhäuser, Boston, 1991.

[2] L. Bachmair and N. Dershowitz. Completion for rewriting modulo a congruence. *Theoretical Computer Science*, 67(2 & 3):173–201, Oct. 1989.

[3] L. Bachmair and N. Dershowitz. Equational inference, canonical proofs, and proof orderings. *J. ACM*, 41:236–276, 1994.

[4] L. Bachmair, I. Ramakrishnan, A. Tiwari, and L. Vigneron. Congruence closure modulo Associativity-Commutativity. In H. Kirchner and C. Ringeissen, editors, *Frontiers of Combining Systems, Third International Workshop, FroCoS 2000*, volume 1794 of *Lecture Notes in Artificial Intelligence*, pages 245–259, Nancy, France, Mar. 2000. Springer-Verlag.

[5] L. Bachmair and A. Tiwari. Abstract congruence closure and specializations. In D. McAllester, editor, *Conference on Automated Deduction, CADE 2000*, volume 1831 of *Lecture Notes in Artificial Intelligence*, pages 64–78, Pittsburgh, PA, June 2000. Springer-Verlag.

[6] L. Bachmair and A. Tiwari. Congruence closure and syntactic unification. In C. Lynch and D. Narendran, editors, *14th International Workshop on Unification*, 2000.

[7] A. M. Ballantyne and D. S. Lankford. New decision algorithms for finitely presented commutative semigroups. *Comp. and Maths. with Appls.*, 7:159–165, 1981.

[8] T. Becker and V. Weispfenning. *Gröbner bases: a computational approach to commutative algebra*. Springer-Verlag, Berlin, 1993.

[9] E. Cardozo, R. Lipton, and A. Meyer. Exponential space complete problems for petri nets and commutative semigroups. In *Proc. 8th Ann. ACM Symp on Theory of Computing*, pages 50–54, 1976.

[10] H. Comon, M. Dauchet, R. Gilleron, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. Tree automata techniques and applications. Available on: `http://www.grappa.univ-lille3.fr/tata`, 1997.

[11] D. Cyrluk, P. Lincoln, and N. Shankar. On Shostak's decision procedure for combination of theories. In M. A. McRobbie and J. Slaney, editors, *Proceedings of the 13th Int. Conference on Automated Deduction*, pages 463–477, 1996. Vol. 1104 of *Lecture Notes in Computer Science*, Springer, Berlin.

[12] N. Dershowitz and Z. Manna. Proving termination with multiset orderings. *Communications of the ACM*, 22(8):465–476, 1979.

[13] E. Domenjoud and F. Klay. Shallow AC theories. In *Proceedings of the 2nd CCL Workshop*, La Escala, Spain, Sept. 1993.

[14] P. J. Downey, R. Sethi, and R. E. Tarjan. Variations on the common subexpressions problem. *J. ACM*, 27(4):758–771, 1980.

[15] T. Evans. The word problem for abstract algebras. *Journal of London Mathematical Society*, 26:64–71, 1951.

[16] T. Evans. Word problems. *Bulletin of the American Mathematical Society*, 84(5):789–802, 1978.

[17] D. Kapur. Shostak's congruence closure as completion. In H. Comon, editor, *Rewriting Techniques and Applications, RTA 1997*, volume 1103 of *Lecture Notes in Computer Science*, pages 23–37, Sitges, Spain, July 1997. Springer-Verlag.

[18] U. Koppenhagen and E. W. Mayr. An optimal algorithm for constructing the reduced Gröbner basis of binomial ideals. In Y. D. Lakshman, editor, *Proceedings of the International Symposium on Symbolic and Algebraic Computation*, pages 55–62, 1996.

[19] C. Marche. On ground AC-completion. In R. V. Book, editor, *4th International Conference on Rewriting Techniques and Applications*, pages 411–422, 1991. Vol. 488 of *Lecture Notes in Computer Science*, Springer, Berlin.

[20] E. W. Mayr and A. R. Meyer. The complexity of the word problems for commutative semigroups and polynomial ideals. *Advances in Mathematics*, 46:305–329, 1982.

[21] P. Narendran and M. Rusinowitch. Any ground associative-commutative theory has a finite canonical system. In R. V. Book, editor, *4th International Conference on Rewriting Techniques and Applications*, pages 423–434, 1991. Vol. 488 of *Lecture Notes in Computer Science*, Springer, Berlin.

[22] G. Nelson and D. Oppen. Fast decision procedures based on congruence closure. *Journal of the Association for Computing Machinery*, 27(2):356–364, Apr. 1980.

[23] G. E. Peterson and M. E. Stickel. Complete sets of reductions for some equational theories. *J. ACM*, 28(2):233–264, Apr. 1981.

[24] D. Plaisted and A. Sattler-Klein. Proof lengths for equational completion. *Information and Computation*, 125:154–170, 1996.

[25] A. Rubio and R. Nieuwenhuis. A precedence-based total AC-compatible ordering. In C. Kirchner, editor, *Proceedings of the 5 Intl. Conference on Rewriting Techniques and Applications*, pages 374–388, 1993. Vol. 690 of *Lecture Notes in Computer Science*, Springer.

[26] D. J. Sherman and N. Magnier. Factotum: Automatic and systematic sharing support for systems analyzers. In *Proc. TACAS, LNCS 1384*, 1998.

[27] R. E. Shostak. Deciding combinations of theories. *Journal of the ACM*, 31(1):1–12, 1984.

[28] W. Snyder. A fast algorithm for generating reduced ground rewriting systems from a set of ground equations. *Journal of Symbolic Computation*, 15(7), 1993.

[29] A. Tiwari. *Decision Procedures in Automated Deduction.* PhD thesis, State University of New York at Stony Brook, NY, 2000.