Abstract Congruence Closure

Leo Bachmair  
Department of Computer Science, State University of New York, Stony Brook, NY 11794-4400, U.S.A. (leo@cs.sunysb.edu)

Ashish Tiwari  
SRI International, 333 Ravenswood Ave, Menlo Park, CA 94025, U.S.A.  
(tiwari@cs1.sri.com)

Laurent Vigneron  
LORIA – Université Nancy 2, Campus Scientifique, B.P. 239, 54506 Vandœuvre-lés-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).

Keywords: Term Rewriting, Congruence Closure, Associative-Commutative Theories

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 as-

pects 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 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 [25]. 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 [29] 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 [29].

1.1. Preliminaries

Let \( \Sigma \) 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 \( \Sigma, \mathcal{V} \) and \( \mathcal{T}(\Sigma, \mathcal{V}) \) are respectively called function symbols, variables and terms (over \( \Sigma \) and \( \mathcal{V} \)). Elements \( c \) in \( \Sigma \) 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 \( \rightarrow_{E} \) induced by a set of equations \( E \) is defined by: \( u \rightarrow_{E} v \) 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 \( \sigma \) 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, \( \leftrightarrow_{E}^{*} \) 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

\(^1\) A congruence relation is a reflexive, symmetric and transitive relation on terms that is also a replacement relation.
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 $s \rightarrow^*_R t \rightarrow^*_R s'$, then there exists a (ground) term $t'$ such that $s \rightarrow^*_R t' \rightarrow^*_R s'$. 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.

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 [16, 17, 18, 27].

DEFINITION 1. Let $\Sigma$ be a signature and $K$ be a set of constants disjoint from $\Sigma$. A $D$-rule (with respect to $\Sigma$ 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 $E_0 = \{a \approx b, f f a \approx f b\}$\footnote{When writing a term, we remove parentheses wherever possible for clarity.} then

$$D_0 = \{a \rightarrow c_0, \ b \rightarrow c_1, \ f c_0 \rightarrow c_2, \ f c_2 \rightarrow c_3, \ f c_1 \rightarrow c_4\}$$

is a set of $D$-rules over $\Sigma_0$ 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 \( f^2 a \) can be rewritten to \( c_3 \) as \( f^2 a \rightarrow \alpha_0 \) \( f^a \rightarrow \alpha_2 \rightarrow \alpha_0 \) \( c_3 \). Original equations in \( E_0 \) can thus be simplified using \( D_0 \) to give \( C_0 = \{ c_0 \approx c_1, c_2 \approx c_3 \} \). The set \( \alpha_0 \cup C_0 \) may be viewed as an alternative representation of \( E_0 \) over an extended signature. The equational theory presented by \( \alpha_0 \cup C_0 \) 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 \( t \rightarrow^*_R c \). A term \( t \) is represented by \( R \) if it is represented by some constant in \( K \) via \( R \). For example, the constant \( c_0 \) represents the term \( f^2 a \) via \( D_0 \).

**DEFINITION 2 (Abstract congruence closure).** Let \( \Sigma \) be a signature and \( K \) be a set of constants disjoint from \( \Sigma \). A ground rewrite system \( R = D \cup C \) of \( D \)-rules and \( C \)-rules (with respect to \( \Sigma \) 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 \( \mathcal{T}(\Sigma) \), \( s \rightarrow^*_E t \) if and only if, \( s \rightarrow^*_R t \),

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

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 \( R_0 = D_0 \cup \{ c_0 \rightarrow c_1, c_3 \rightarrow c_4 \} \) 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

\[
R_1 = \{ a \rightarrow c_1, b \rightarrow c_1, fc_1 \rightarrow c_4, fc_4 \rightarrow c_4, \\
c_0 \rightarrow c_1, c_2 \rightarrow c_4, c_3 \rightarrow c_4 \}.
\]
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 \(\Sigma\), \(E\) 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 \(T(\Sigma)\), but simply an ordering on \(T(\Sigma \cup U)\) (that is, terms in \(T(\Sigma)\) need not be comparable in this ordering) where \(U\) is an infinite set disjoint from \(\Sigma\) 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 \(t \rightarrow c\) 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.

\[
\text{Extension:} \quad \frac{(K, E[t], R)}{(K \cup \{c\}, E[c], R \cup \{t \rightarrow c\})}
\]

where \(t \rightarrow c\) 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.

\[
\text{Simplification:} \quad \frac{(K, E[t], R \cup \{t \rightarrow c\})}{(K, E[c], R \cup \{t \rightarrow c\})}
\]

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 \(\succ\).)

\[
\text{Orientation:} \quad \frac{(K \cup \{c\}, E \cup \{t \equiv c\}, R)}{(K \cup \{c\}, E, R \cup \{t \rightarrow c\})}
\]

---

\(^3\) 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 \(t \succ s\).
Abstract Congruence Closure

if \( t > c \).

Trivial equations may be deleted.

\[
\text{Deletion:} \quad \frac{(K, E \cup \{t \approx t\}, R)}{(K, E, R)}
\]

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.

\[
\text{Deduction:} \quad \frac{(K, E, R \cup \{t \rightarrow c, t \rightarrow d\})}{(K, E \cup \{c \approx d\}, R \cup \{t \rightarrow d\})}
\]

\[
\text{Collapse:} \quad \frac{(K, E, R \cup \{s[c] \rightarrow d', c \rightarrow d\})}{(K, E, R \cup \{s[d] \rightarrow d', c \rightarrow d\})}
\]

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.

\[
\text{Composition:} \quad \frac{(K, E, R \cup \{t \rightarrow d, c \rightarrow d\})}{(K, E, R \cup \{t \rightarrow c, c \rightarrow d\})}
\]

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 the initial state \((K_0, E_0, R_0) = (\emptyset, E_0, \emptyset)\) as follows:

<table>
<thead>
<tr>
<th>( i )</th>
<th>Constants ( K_i )</th>
<th>Equations ( E_i )</th>
<th>Rules ( R_i )</th>
<th>Transition</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>( \emptyset )</td>
<td>( E_0 )</td>
<td>( \emptyset )</td>
<td>Ext</td>
</tr>
<tr>
<td>1</td>
<td>( {c_0} )</td>
<td>( c_0 \approx b, ffa \approx fb )</td>
<td>( { a \rightarrow c_0 } )</td>
<td>Ori</td>
</tr>
<tr>
<td>2</td>
<td>( {c_0} )</td>
<td>( ffa \approx fb )</td>
<td>( { a \rightarrow c_0, b \rightarrow c_0 } )</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>( {c_0} )</td>
<td>( ffc_0 \approx ffc_0 )</td>
<td>( { a \rightarrow c_0, b \rightarrow c_0 } )</td>
<td>Sim (twice)</td>
</tr>
<tr>
<td>4</td>
<td>( {c_0, c_1} )</td>
<td>( ffc_1 \approx ffc_0 )</td>
<td>( R_3 \cup { ffc_0 \rightarrow c_1 } )</td>
<td>Ext</td>
</tr>
<tr>
<td>5</td>
<td>( {c_0, c_1} )</td>
<td>( ffc_1 \approx c_1 )</td>
<td>( R_3 \cup { ffc_0 \rightarrow c_1 } )</td>
<td>Sim</td>
</tr>
<tr>
<td>6</td>
<td>( K_5 )</td>
<td>( {} )</td>
<td>( R_5 \cup { ffc_1 \rightarrow c_1 } )</td>
<td>Ori</td>
</tr>
</tbody>
</table>

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 $T(\Sigma \cup K)$, we have $s \leftrightarrow^*_E t$ if, and only if, $s \leftrightarrow^*_R t$.

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

**Lemma 1.** Let $K_0$ be a finite set of constants (disjoint from $\Sigma$), $E_0$ 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 $c \mapsto d$ in $R_0$ we have $c \nless_U d$. 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 $\Sigma$ 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, consider 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 $\nless_{K_n}$ (defined as the restriction of the ordering $\nless_U$ 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, $c \nless_{K_n} d$; and set $w(f) = \max\{w(c) : c \in K_n\} + 1$, for each $f \in \Sigma$. Let $\not\gg$ be the Knuth-Bendix ordering
using these weights. Define a secondary measure of a state \((K, E, R)\) as the set \(\{ \{ \{ s, t \} \} : s \approx t \in E \} \cup \{ \{ \{ s \}, \{ t \} \} : s \rightarrow t \in R \} \). Two states are compared by comparing their measures using a two-fold multiset extension\(^4\) of the ordering \(\succ\) on terms. It is straightforward 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 \(\succ\), 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 \(s \in T(\Sigma)\) such that \(c \leftrightarrow^\oplus \_R s\). Then, for every \(d \in K'\), there exists a term \(t \in T(\Sigma)\) such that \(d \leftrightarrow^\oplus \_R t\).

**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 \(f(c_1, \ldots, c_k) \rightarrow d\) be the rule introduced by extension. Since \(c_1, \ldots, c_k \in K\), there exist terms \(s_1, \ldots, s_k \in T(\Sigma)\) such that \(s_i \leftrightarrow^\oplus \_R c_i\) and hence, using Theorem 1, \(s_i \leftrightarrow^\oplus \_R c_i\). 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. It follows from Lemma 1 that final states can be finitely derived. The third component of a final state is always an abstract congruence closure.

**Theorem 2.** Let \(\Sigma\) be a signature and \(K_1\) a finite set of constants disjoint from \(\Sigma\). 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 \(t \in T(\Sigma)\) via \(E_1 \cup R_1\), and \(c \succ \_ R d\) for every \(C\)-rule \(c \rightarrow d\) 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 \(\Sigma\) and \(K_n\)).

**Proof.** Since the sets \(K_1, E_1,\) and \(R_1\) are finite and the state \((K_n, E_n, R_n)\) is obtained from \((K_1, E_1, R_1)\) using a finite derivation, it follows that \(K_n, E_n,\) and \(R_n\) are all finite sets. If \(E_n \neq \emptyset\), then either

---

\(^4\) 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 [13].
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 \(s \leftrightarrow_{E_1 \cup R_1}^t t\) for some \(s, t \in \mathcal{T}(\Sigma)\), then \(s \leftrightarrow_{E_n \cup R_n}^t t\). Since \(E_n = \emptyset\) and \(R_n\) is convergent, \(s \not\rightarrow_{R_n}^t 0\).

\[ \square \]

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 \(\Sigma\)) and an ordering (irreflexive and transitive relation) on this set. It might appear that the abstract congruence closure one obtains depends on the ordering \(\triangleright_U\) 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 \(\triangleright_U\) 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.

\[
\text{Compression: } \quad \frac{(K \cup \{c, d\}, E, R \cup \{c \rightarrow d\})}{(K \cup \{d\}, E \langle c \mapsto d \rangle, R(c \mapsto d))}
\]

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 \(\sigma\) defined as \(\sigma(c) = \)
$d$ and $\sigma(x) = x$ for $x \neq c$, and $E(c \mapsto d)$ 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 $\Sigma$ be a signature and $E$ be a finite set of equations over $\Sigma$. Then, there exists an abstract congruence closure $D$ for $E$ (over $\Sigma$ 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 $T(\Sigma \cup (K_i \cap K_j))$, $s \rightarrow^+_{E_i \cup R_i} t$ if and only if $s \rightarrow^+_{E_j \cup R_j} t$. 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 $\Sigma$ and $K$) as any rule of the form $t \rightarrow c$ where $c \in K$ and $t \in T(\Sigma, K) - K$, 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.

*Choosing an ordering $\succ_U$ on the fly:* As remarked earlier, the set of transition rules presented in Section 2.1\(^5\) 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 $c \succ_j d$ if $c \rightarrow d \in \cup_{k \leq j} C_k$. We claim that each $\succ_j$ defines

\(^5\) We exclude Compression for rest of the discussion.
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.

**Bounding the maximal derivation length:** 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 unflailing\(^6\) 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 \( \delta \).

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 the 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 \( E_0 \), \( \delta \) is the depth of ordering \( \succ_U \) used to construct the derivation, and \( n \) is the number of \( \Sigma \)-symbols in \( E_0 \).

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

\[
\begin{align*}
\text{Sim1 : } & \frac{(E[f(\ldots)], R \cup \{ f(\ldots) \to c \})}{(E[c], R \cup \{ f(\ldots) \to c \})} \\
\text{Sim2 : } & \frac{(E[c], R \cup \{ c \to d \})}{(E[d], R \cup \{ c \to d \})} \\
\text{Ded1 : } & \frac{(E, R \cup \{ f(\ldots) \to c, f(\ldots) \to d \})}{(E \cup \{ c \approx d \}, R \cup \{ f(\ldots) \to d \})} \\
\text{Ded2 : } & \frac{(E, R \cup \{ c \to d, c \to d \})}{(E \cup \{ d \approx d \}, R \cup \{ c \to d \})}
\end{align*}
\]

\(^6\) By *unflailing* we mean that the set of unoriented equations in the final state is empty.
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 $c_1 \rightarrow c_2 \rightarrow \cdots$ is bounded by $\delta$ and $2k + l$ is an upper bound on the number of occurrences of constants (from $K_\infty$) 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 simplified steps is at most $n$ as each such step reduces the number of $\Sigma$-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 $\Sigma$-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 $\Sigma$-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 $K_\infty$ of constants is feasible, but has depth equal to the cardinality of $K_\infty$, 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 $\succeq_U$ for this state with depth $O(\log(n))$, where $n$ is the number of constants in $K_m$.

Proof. We shall exhibit an unflailing 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 $(K_i, E_i \cup \{c \approx d\}, D_i \cup C_i) \vdash (K_i, E_i, D_i \cup C_i \cup \{c \rightarrow d\})$, the constants $c$ and $d$ are in $C_i$-normal form. Additionally, we also impose the requirement that the cardinality of the set \( \{c' \in K_m : c' \triangleright \triangleright_{C_i} c\} \) is less than or equal to the cardinality of \( \{c' \in K_m : c' \triangleright \triangleright_{C_i} d\} \).

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 unflailing derivation. If $c_1 \triangleright c_2 \triangleright \cdots \triangleright c_j$ is a maximal descending chain, then the cardinality of the set \( \{c' \in K_m : c' \triangleright \triangleright_{C_\infty} c_j\} \) is at least $2^{j-1}$. But, since the cardinality of $K_m$ is $O(n)$, therefore, $j = O(\log(n))$. \hfill \Box

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 the 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. \hfill \Box

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

\[
[C]_C \overset{\text{def}}{=} \{c' \in K : c' \triangleright \triangleright_{C} c\}
\]

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 \texttt{counter} can easily be updated when a $C$-equation, say $c \approx d$, is oriented into, say $c \rightarrow d$, by setting $\text{counter}(d) = \text{counter}(c) + \text{counter}(d)$.

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 (in fact, depth 1) 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. For example, if $c$ is one such constant and $[c]_{C_i} = \{c, c_1, \ldots, c_k\}$, then we orient so that we add rules $\{c_i \rightarrow c : i = 1, \ldots, k\}$ to the third component. That such $C$-equations always exist and can be efficiently identified is a simple consequence of the finiteness assumption, see [30, 15] 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 [15], Nelson and Oppen [23], and Shostak [28] 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 labeled by the symbol $f$ and has pointers
D-rules representing the term dag:

\[
\begin{align*}
  a &\rightarrow c_1 \\
  b &\rightarrow c_1 \\
  c &\rightarrow c_6 \\
  d &\rightarrow c_7 \\
  h c_4 &\rightarrow c_5 \\
\end{align*}
\]

\[
\begin{align*}
  g c_1 c_1 &\rightarrow c_2 \\
  f c_1 c_2 &\rightarrow c_3 \\
  g c_6 c_8 &\rightarrow c_9 \\
  h c_7 &\rightarrow c_8 \\
  f c_5 c_9 &\rightarrow c_{10}
\end{align*}
\]

C-rules representing the relation on vertices:

\[
\begin{align*}
  c_1 &\approx c_5 \\
  c_2 &\approx c_9 \\
  c_3 &\approx c_{10} \\
  c_4 &\approx c_7 \\
  c_5 &\approx c_5 \\
  c_5 &\approx c_8
\end{align*}
\]

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

to the nodes \(c_1, \ldots, c_8\). 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 dag 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 \(\Sigma\) and \(K\)) as defined in Section 2.3 correspond to storing contexts, rather than just symbols from \(\Sigma\), 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 corresponding 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 \((\text{Ext} \cdot \text{Sim})^t\)). 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 \(E_0 = \{a \approx b, f f a \approx f b\}\), 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 \(R_1 = \{a \rightarrow c_0, b \rightarrow c_1, f c_0 \rightarrow c_2, f c_2 \rightarrow c_3, f c_1 \rightarrow c_4\}\).

(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) A 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.

3.1. 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:

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

This expression should be interpreted as follows. Given a (start) state $(K, E, R)$

1. pick an equation $s \approx t$ from the set $E$.
2. apply simplification to this state to normalize $s$, i.e., $s \rightarrow_R s'$.
3. 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 $d$.
4. if $c$ and $d$ are identical then apply deletion (and continue with (i));
5. if not, create a $C$-rule, say $c \rightarrow d$, using orientation.
6. 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.

<table>
<thead>
<tr>
<th>$i$</th>
<th>$K_i$</th>
<th>$E_i$</th>
<th>$R_i$</th>
<th>Transition</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>$\emptyset$</td>
<td>$E_0$</td>
<td>$\emptyset$</td>
<td>Ext · Ext · Ori</td>
</tr>
<tr>
<td>1</td>
<td>${c_0, c_1}$</td>
<td>${f f a \approx f b}$</td>
<td>${a \rightarrow c_0, \ b \rightarrow c_1, c_0 \rightarrow c_1}$</td>
<td>Sim · Sim</td>
</tr>
<tr>
<td>2</td>
<td>${c_0, c_1}$</td>
<td>${f f c_1 \approx f b}$</td>
<td>${a \rightarrow c_0, \ b \rightarrow c_1, c_0 \rightarrow c_1}$</td>
<td>Ext · Ext</td>
</tr>
<tr>
<td>3</td>
<td>${c_0, \ldots, c_3}$</td>
<td>${c_3 \approx f b}$</td>
<td>$R_2 \cup {f c_1 \rightarrow c_2, \ f c_2 \rightarrow c_3}$</td>
<td>Sim · Sim</td>
</tr>
<tr>
<td>4</td>
<td>${c_0, \ldots, c_3}$</td>
<td>${c_3 \approx c_2}$</td>
<td>$R_3$</td>
<td>Sim · Sim</td>
</tr>
<tr>
<td>5</td>
<td>${c_0, \ldots, c_3}$</td>
<td>$\emptyset$</td>
<td>$R_4 \cup {c_3 \rightarrow c_2}$</td>
<td>Ori</td>
</tr>
</tbody>
</table>

3.2. 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:

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

where $\epsilon$ 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 $c \rightarrow d$ 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 $K = \{c_0, \ldots, c_1\}, D_1 = \{a \rightarrow c_0, \ b \rightarrow c_1, f c_0 \rightarrow c_2, f c_2 \rightarrow$
Abstract Congruence Closure

c_3, \ f c_1 \rightarrow c_4, \text{ and, } C_1 = \{c_0 \rightarrow c_1, \ c_3 \rightarrow c_1\}.

<table>
<thead>
<tr>
<th>#</th>
<th>Consts K_i</th>
<th>Eqns E_i</th>
<th>Rules R_i</th>
<th>Transition</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>K_1</td>
<td>\emptyset</td>
<td>D_1 \cup C_1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>K_1</td>
<td>\emptyset</td>
<td>{a \rightarrow c_0, \ b \rightarrow c_1, \ f c_1 \rightarrow c_2, \ f c_2 \rightarrow c_3, \ f c_1 \rightarrow c_4} \cup C_1</td>
<td>Col</td>
</tr>
<tr>
<td>3</td>
<td>K_1</td>
<td>{c_2 \approx c_4}</td>
<td>R_2 - {f c_1 \rightarrow c_2}</td>
<td>Ded</td>
</tr>
<tr>
<td>4</td>
<td>K_1</td>
<td>\emptyset</td>
<td>R_3 \cup {c_1 \rightarrow c_2}</td>
<td>Ori</td>
</tr>
</tbody>
</table>

Note that c_1 \approx c_2 was oriented in a way that no further collapses were needed thereafter.

3.3. Nelson and Oppen’s Algorithm

The Nelson-Oppen procedure is not exactly a completion procedure and it does not generate a congruence closure in our sense. The initial state of the Nelson-Oppen procedure is given by the tuple \((K_1, E_1, D_1)\), where \(D_i\) is the input dag, and \(E_1\) represents an equivalence on vertices of this dag. The sets \(K_1\) and \(D_1\) remain unchanged in the Nelson-Oppen procedure. In particular, the inference rule used for deduction is different from the conventional deduction rule\(^7\).

\[
\text{NODeduction: } (K, E, D \cup C) \quad \frac{(K, E \cup \{c \approx d\}, D \cup C)}{(K, E \cup \{c \approx d\}, D \cup C)}
\]

if there exist two \(D\)-rules \(f(c_1, \ldots, c_k) \rightarrow c\), and, \(f(d_1, \ldots, d_k) \rightarrow d\) in the set \(D\); and, \(c_i \rightarrow c_i^C \circ \leftarrow C d_i\), for \(i = 1, \ldots, k\).

The Nelson-Oppen procedure can now be (at a certain abstract level) represented as:

\[
\text{NO} = (\text{Sim}^4 \cdot (\text{Ori} \cup \text{Del}) \cdot \text{NODed}^4)^5
\]

which is applied in the following sense: (i) select a \(C\)-equation \(c \approx d\) from the \(E\)-component, (ii) simplify the terms \(c\) and \(d\) using simplification steps until the terms can’t be simplified any more, (iii) either delete, or orient the simplified \(C\)-equation, (iv) apply the NODeduction rule until there are no more non-redundant applications of this rule, (v) if the \(E\)-component is empty, then we stop, otherwise continue with step (i).

---

\(^7\) This rule performs deduction modulo \(C\)-equations, i.e., we compute critical pairs between \(D\)-rules modulo the congruence induced by \(C\)-equations. Hence, the Nelson-Oppen procedure can be described as an extended completion \([12]\) (or completion modulo \(C\)-equations) method over an extended signature.
Using the Nelson-Oppen strategy, assume we get a derivation 
\((K_1, E_1, D_1) \vdash_{NO} (K_n, E_n, D_n \cup C_n)\). One consequence of using a 
non-standard deduction rule, NO-Deduction, is that the resulting set 
\(D_n \cup C_n = D_1 \cup C_n\) need not necessarily be convergent, although the 
rewrite relation \(D_n/C_n\) [12] is convergent.

**Example 4.** Using the same set \(E_0\) as equations, we illustrate the 
Nelson-Oppen procedure. The initial state is given by \((K_1, E_1, D_1)\) 
where \(K_1 = \{c_0, c_1, c_2, c_3, c_4\}; E_1 = \{c_0 \approx c_1, c_3 \approx c_4\};\) and, 
\(D_1 = \{a \rightarrow c_0, b \rightarrow c_1, f c_0 \rightarrow c_2, f c_2 \rightarrow c_3, f c_1 \rightarrow c_4\}.\)

<table>
<thead>
<tr>
<th>#</th>
<th>Constants</th>
<th>Equations</th>
<th>Rules</th>
<th>Transition</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>(K_1)</td>
<td>(E_1)</td>
<td>(D_1)</td>
<td>(D_1 \cup {c_0 \rightarrow c_1}) Ori</td>
</tr>
<tr>
<td>2</td>
<td>(K_1)</td>
<td>({c_3 \approx c_1})</td>
<td>(D_1 \cup {c_0 \rightarrow c_1})</td>
<td>Ori</td>
</tr>
<tr>
<td>3</td>
<td>(K_1)</td>
<td>({c_2 \approx c_1, c_3 \approx c_4})</td>
<td>(R_2)</td>
<td>NO-Ded</td>
</tr>
<tr>
<td>4</td>
<td>(K_1)</td>
<td>({c_3 \approx c_4})</td>
<td>(R_2 \cup {c_2 \rightarrow c_4})</td>
<td>Ori</td>
</tr>
<tr>
<td>5</td>
<td>(K_1)</td>
<td>(\emptyset)</td>
<td>(R_4 \cup {c_3 \rightarrow c_4})</td>
<td>Ori</td>
</tr>
</tbody>
</table>

Consider deciding the equality \(fa \approx f fb\). Even though \(fa \leftrightarrow_{E_0} f fb\), 
the terms \(fa\) and \(f fb\) have distinct normal forms with respect to \(R_5\). 
But terms in the original term universe have identical normal forms.

### 4. Experimental Results

We have implemented several congruence closure algorithms, including 
those proposed by Nelson and Oppen (NO) [23], Downey, Sethi and 
Tarjan (DST) [15], and Shtetl (SHO) [28], 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 three implementations.

NO is an implementation of the pseudocode given on page 358 (with 
some details on page 359) of [23]. 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 [23]. 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 [15]. In particular, the 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 on page 8 of [28]. 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 [28]. A description of this algorithm (with only a slight difference in the order of calls to subroutine merge) is also present in [11, 18].

The completion procedure COM uses the following strategy:

\[(\text{Sim}^* \cdot \text{Ext}^*)^\dagger \cdot (\text{Del} \cup \text{Ori}) \cdot ((\text{Com}^* \cdot \text{Col}) \cdot \text{Ded} \cdot (\text{Del} \cup \text{Ori}))^\dagger)^\dagger.\]

More specifically, we processed 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

\[(\text{Sim}^* \cdot \text{Ext}^*)^\dagger \cdot ((\text{Del} \cup \text{Ori}) \cdot (\text{Col}^* \cdot \text{Com}^* \cdot \text{Ded}^*)^\dagger \cdot \text{Sim}^*)^\dagger)^\dagger.\]

As before, using \(\text{Sim}^* \cdot \text{Ext}^*\) we convert one equation to a C-equation. This equation is oriented and individually on every D-rule, we perform all simplifications using this C-rule, viz collapse, composition, followed by any deduction step \((\text{Col}^* \cdot \text{Com}^* \cdot \text{Ded}^*)\). 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 pre-processing 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. Example 12 is the same as 11, except that it contains five copies of all the equations. Example 13 requires slightly larger use lists. Finally, example 14 consists of equations that are oriented in the “wrong” way.

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 the dag (for NO and DST). In Table I 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.

---

8 The equation set is \( \{ f^2(a) \approx a, f^{10}a \approx f^3b, b \approx f^3b, a \approx f^9a, f^9b \approx b \} \).

9 The equation set is \( \{ g(a, a, b) \approx f(a, b), gabb \approx fba, gaab \approx gaba, gbab \approx gabb, gbaa \approx gbba, gaab \approx gaba, gbab \approx gabb, gbaa \approx gbba \} \).

10 The set is \( \{ g(f^2(a), h^{10}(b)) \approx g(a, b), i = \{ 1, \ldots , 25 \}, h^{10}(b) \approx b, b \approx h^{10}(b), h(b) \approx c0, c0 \approx c1, c1 \approx c2, c2 \approx c3, c3 \approx c4, c4 \approx a, a \approx f(a) \} \).
Table II. Total running time (in seconds) for randomly generated sets of equations.

<table>
<thead>
<tr>
<th>Exs</th>
<th>Vert</th>
<th>(\Sigma_0, \Sigma_1, \Sigma_2, d)</th>
<th>Class</th>
<th>DST</th>
<th>NO</th>
<th>SHO</th>
<th>IND</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ex:21</td>
<td>10000</td>
<td>2, 0, 2, 3</td>
<td>7472</td>
<td>11.1</td>
<td>3.19</td>
<td>10.2</td>
<td>13.0</td>
</tr>
<tr>
<td>Ex:22</td>
<td>5000</td>
<td>4163</td>
<td>3</td>
<td>2.28</td>
<td>306</td>
<td>3.09</td>
<td>0.77</td>
</tr>
<tr>
<td>Ex:23</td>
<td>5000</td>
<td>7869</td>
<td>2745</td>
<td>2.44</td>
<td>1.36</td>
<td>3.52</td>
<td>3.99</td>
</tr>
<tr>
<td>Ex:24</td>
<td>6000</td>
<td>8885</td>
<td>3, 0, 1, 3</td>
<td>9</td>
<td>3.55</td>
<td>1152</td>
<td>52.4</td>
</tr>
<tr>
<td>Ex:25</td>
<td>7000</td>
<td>9818</td>
<td>3, 0, 1, 3</td>
<td>1</td>
<td>4.63</td>
<td>1682</td>
<td>47.8</td>
</tr>
<tr>
<td>Ex:26</td>
<td>5000</td>
<td>645</td>
<td>4, 2, 0, 23</td>
<td>77</td>
<td>1.22</td>
<td>1.58</td>
<td>0.37</td>
</tr>
<tr>
<td>Ex:27</td>
<td>5000</td>
<td>1438</td>
<td>10, 2, 0, 23</td>
<td>290</td>
<td>1.45</td>
<td>3.67</td>
<td>0.39</td>
</tr>
</tbody>
</table>

Table II 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 \(\Sigma_i\) 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\(^{11}\).

In Table III 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 \(\text{IND}^*\) indexed completion based on a strategy that first constructs a dag. The examples are the same as in Table II.

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 deduction steps many times over a single execution. Not surprisingly, straight-forward completion without indexing is also inefficient when

\(^{11}\) Times for COM are not included as indexing is indispensable for larger examples.
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.

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

<table>
<thead>
<tr>
<th></th>
<th>DST</th>
<th>NO</th>
<th>IND*</th>
<th></th>
<th>DST</th>
<th>NO</th>
<th>IND*</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ex.21</td>
<td>0.919</td>
<td>0.296</td>
<td>0.076</td>
<td>Ex.25</td>
<td>0.958</td>
<td>1634.961</td>
<td>9.770</td>
</tr>
<tr>
<td>Ex.22</td>
<td>0.309</td>
<td>319.112</td>
<td>1.971</td>
<td>Ex.26</td>
<td>0.026</td>
<td>0.781</td>
<td>0.060</td>
</tr>
<tr>
<td>Ex.23</td>
<td>0.241</td>
<td>0.166</td>
<td>0.030</td>
<td>Ex.27</td>
<td>0.048</td>
<td>2.470</td>
<td>0.176</td>
</tr>
<tr>
<td>Ex.24</td>
<td>0.776</td>
<td>1117.239</td>
<td>7.301</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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 [15].

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

\[
\begin{align*}
  f(x_1, \ldots, x_k, y_1, \ldots, y_l, t, z_1, \ldots, z_m) & \approx f(x_1, \ldots, x_k, t, y_1, \ldots, y_l, z_1, \ldots, z_m) \\
  f(x_1, \ldots, x_k, y_1, \ldots, y_l, s, z_1, \ldots, z_m) & \approx f(x_1, \ldots, x_k, s, y_1, \ldots, y_l, z_1, \ldots, z_m)
\end{align*}
\]

where $f \in \Sigma_{AC}$, $k, l, m \geq 0$, and $k + l + m + 2 \in \alpha(f)$; and by $F$ the set of identities

\[
\begin{align*}
  f(x_1, \ldots, x_m, f(y_1, \ldots, y_r), z_1, \ldots, z_n) & \approx f(x_1, \ldots, x_m, y_1, \ldots, y_r, z_1, \ldots, z_n) \\
  f(x_1, \ldots, x_m, y_1, \ldots, y_r, z_1, \ldots, z_n) & \approx f(x_1, \ldots, x_m, y_1, \ldots, y_r, z_1, \ldots, z_n)
\end{align*}
\]

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 $\Sigma_{AC}$ are called associative-commutative operators\(^\dagger\). We require that $\alpha(f)$ be a singleton set for all $f \in \Sigma - \Sigma_{AC}$ and $\alpha(f) = \{2, 3, 4, \ldots \}$ for all $f \in \Sigma_{AC}$.

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

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

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 $AC \setminus R$ we denote the rewrite system consisting of all rules

\[^\dagger\text{The equations } F \cup P \text{ define a conservative extension of the theory of associativity and commutativity to variadic terms. For a fixed arity binary function symbol, the equations } f(x, y) \approx f(y, x) \text{ and } f(f(x, y), z) \approx f(x, f(y, z)) \text{ define an } AC\text{-theory.}\]
$u \rightarrow v$ such that $u \leftrightarrow^*_A C u \sigma$ and $v = v' \sigma$, for some rule $u' \rightarrow v'$ in $R$ and some substitution $\sigma$. We say that $AC \setminus R$ is confluent modulo $AC$ if for all terms $s, t$ such that $s \leftrightarrow^*_R AC t$, there exist terms $w$ and $w'$ such that $s \rightarrow^*_R AC w \leftrightarrow^*_A C u \sigma$ and $w' \rightarrow^*_A C v \sigma$. 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 [24]. Given an $AC$-operator $f$ and a rewrite rule $\rho : f(c_1, c_2) \rightarrow c$, we consider its extension $\rho': f(f(c_1, c_2), x) \rightarrow f(c, x)$. 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 $AC \setminus R^e$ and $t \leftrightarrow^*_AC t'$, then $t'$ is also reducible by $AC \setminus R^e$.

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

**DEFINITION 7.** Let $\Sigma$ be a signature and $K$ be a set of constants disjoint from $\Sigma$. A ground rewrite system $R = A \cup D \cup C$ is said to be an associative-commutative congruence closure (with respect to $\Sigma$ 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 $t \in T(\Sigma)$ via $R$, and

(ii) $AC \setminus R^e$ is ground convergent modulo $AC$ over $T(\Sigma \cup K)$.

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

(iii) If $s$ and $t$ are terms over $T(\Sigma)$, then $s \leftrightarrow^*_AC \cup E t$ if, and only if, $s \rightarrow^*_AC \cup Re \leftrightarrow^*_AC \cup E t$,

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

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

For example, let $\Sigma$ 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:

$$ R_1 = \{ a \rightarrow c_1, \ b \rightarrow c_2, \ c \rightarrow c_3, \ f(c_2, c_3) \rightarrow c_4, \ g(c_4) \rightarrow c_5, \ f(c_1, c_3) \rightarrow c_1, \ f(c_3, c_5) \rightarrow c_2, \ c_5 \rightarrow c_1 \}.$$
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 $AC$) process described in more detail in the next section, to obtain a congruence closure (details are given in Example 5),

$$R' = \{ a \to c_1, b \to c_2, c \to c_3, f_2 c_3 \to c_4, f c_3 c_1 \to c_2, f c_1 c_3 \to c_1, f c_2 \to f c_4 c_1, f c_1 c_2 \to f c_1 c_3, g c_1 \to c_4 \}$$

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 orient the $D$-rules in the right way, and orients all the $C$- and $A$-equations. The precedence-based $AC$-compatible ordering $\succ$ of [26], 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 [26]. 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 $\{c_1, \ldots, c_i\} \succ_{\text{mult}} \{c'_1, \ldots, c'_j\}$, i.e., if the two terms have the same number of arguments, we compare the multisets of constants using a multiset extension $\succ_{\text{mult}}$ of the precedence $\succ_{\Sigma \cup U}$, see [13].

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 $\Sigma$ 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.

\begin{align*}
\text{Extension:} \\
\frac{(K, E[t], R)}{(K \cup \{c\}, E[c], R \cup \{t \to c\})}
\end{align*}

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, \( s \rightarrow_{AC \setminus R} t \).

It is fairly easy to see that any equation in \( E \) can be transformed to a \( D \)-, \( C \)- or an \( A \)-equation by suitable extension and simplification\(^{13}\).

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.

**Orientation:** \[
\frac{(K, E \cup \{s \approx t\}, R)}{(K, E, R \cup \{s \rightarrow t\})}
\]
if \( s \approx t \), and, \( s \rightarrow t \) is either a \( D \)-rule, or a \( C \)-rule, or an \( A \)-rule.

Deletion allows us to delete trivial equations.

**Deletion:** \[
\frac{(K, E \cup \{s \approx t\}, R)}{(K, E, R)}
\]
if \( s \leftrightarrow_{A \setminus C} t \).

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

**ACSuperposition:** \[
\frac{(K, E, R)}{(K, E \cup \{f(s, x\sigma) \approx f(t, y\sigma)\}, R)}
\]
if \( f \in \Sigma_{AC} \), there exist \( D \)- or \( A \)-rules (fully flattened as) \( f(c_1, \ldots, c_k) \rightarrow s \) and \( f(d_1, \ldots, d_l) \rightarrow t \) in \( R \), the sets \( C = \{c_1, \ldots, c_k\} \) and \( D = \{d_1, \ldots, d_l\} \) are not disjoint, \( C \not\subseteq D \), \( D \not\subseteq C \), and the substitution \( \sigma \) is the ground substitution in a minimal complete set of \( AC \)-unifiers for \( f(c_1, \ldots, c_k, x) \) and \( f(d_1, \ldots, d_l, y)\)\(^{14}\).

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

**ACCollapse:** \[
\frac{(K, E, R \cup \{t \rightarrow s\})}{(K, E \cup \{t' \approx s\}, R)}
\]
if for some \( u \rightarrow v \in R \), \( t \rightarrow_{AC \setminus \{u \rightarrow v\}} t' \), and if \( t \leftrightarrow_{A \setminus C} u \) then \( s \leftrightarrow v \).

\(^{13}\) We do not need an explicit rule for flattening as Definition 5 allows for non-flattened terms to occur in \( A \)-rules.

\(^{14}\) For the special case in hand, a minimal complete set of \( AC \)-unifiers contains exactly two substitutions, exactly one of which is ground.
The Deduction inference rule in Section 2.1 (for non-AC terms) is subsumed by ACCollapse. 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.

$$
\text{Composition: } \frac{(K, E, R \cup \{t : s\})}{(K, E, R \cup \{t' : s'\})}
$$

if $s \rightarrow_{AC \setminus R} s'$.

Example 5. 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 $i < j$.

<table>
<thead>
<tr>
<th>$i$</th>
<th>Constants $K_i$</th>
<th>Equations $E_i$</th>
<th>Rules $R_i$</th>
<th>Transitions</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>$\emptyset$</td>
<td>$E_0$</td>
<td>$\emptyset$</td>
<td></td>
</tr>
</tbody>
</table>
| 1   | $\{c_1, c_3\}$ | $\{f \circ g \circ f \circ b \approx b, f \circ c_1 \circ c_3 \rightarrow c_3, \}

\{f \circ g \circ f \circ b \approx f \circ b\}

\{f \circ c_3 \rightarrow c_1\}$ | Ext$^2 \cdot$ Sim \cdot Ori |
| 2   | $K_1 \cup \{c_2, c_4\}$ | $\{f \circ g \circ f \circ b \approx b\}$ | $R_1 \cup \{b \rightarrow c_2, \}

\{f \circ c_2 \rightarrow c_4, g \circ c_4 \rightarrow c_4\}$ | Sim$^2 \cdot$ Ext$^2$, Sim \cdot Ori |
| 3   | $K_2$          | $\emptyset$    | $R_2 \cup \{f \circ c_4 \rightarrow c_2\}$ | Sim$^3 \cdot$ Ori |
| 4   | $K_2$          | $\emptyset$    | $R_3 \cup \{f \circ c_2 \rightarrow f \circ c_4\}$ | ACUsu \cdot Ori |
| 5   | $K_2$          | $\emptyset$    | $R_4 \cup \{f \circ c_2 \rightarrow f \circ c_4\}$ | ACUsu \cdot 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 $E_0$. There are more ACUsuperpositions, but the resulting equations get deleted. Note that the size-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 \(R_{\infty}\) of persisting rewrite rules is defined as \(\bigcup_i \cap_{j > i} R_j\); and similarly, \(K_{\infty} = \bigcup_i \cap_{j > i} K_j\).

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 \(K_{\infty}\) can be compared using the multiset inclusion relation. If \(K_{\infty}\) 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 \(R_{\infty}\) in any fair derivation starting from state \((\emptyset, E, \emptyset)\) is finite.

**Proof.** We first claim that \(K_{\infty}\) 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 the set \(R_{\infty}\) is finite. Suppose \(R_{\infty}\) is an infinite set. Since non \(\Sigma_{AC}\)-symbols have fixed arities, therefore, \(R_{\infty}\) contains infinitely many rules with top symbol from \(\Sigma_{AC}\). Since \(\Sigma_{AC}\) is finite, one \(AC\)-operator, say \(f \in \Sigma_{AC}\), must occur infinitely often as the top symbol in the left-hand sides of \(R_{\infty}\). By Dickson’s lemma, there exists an infinite chain of rules (written as fully flattened for simplicity), \(f(c_{11}, \ldots, c_{1k_1}) \rightarrow s_0, f(c_{21}, \ldots, c_{2k_2}) \rightarrow s_1, \ldots\), such that \(\{c_{11}, \ldots, c_{1k_1}\} \subseteq \{c_{21}, \ldots, c_{2k_2}\} \subseteq \cdots\), where \(\{c_{11}, \ldots, c_{1k_1}\}\) denotes a multiset and \(\subseteq\) denotes multiset inclusion. But, this contradicts fairness (in application of \text{ACC}\text{ollapse}).

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

\[
\begin{align*}
(s,t), \top, \top & \quad \text{if } u \equiv v \in E \\
(s), u, t & \quad \text{if } u \rightarrow v \in R \\
(t), v, s & \quad \text{if } v \rightarrow u \in R
\end{align*}
\]

where \( \top \) is a new symbol. Tuples are compared lexicographically using the multiset extension of the reduction ordering \( \succ \) on terms over \( \Sigma \cup K \) 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 \( \succ_p \). The ordering \( \succ_p \) 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 \( s,t \in T(\Sigma) \), it is the case that \( s \leftrightarrow^*_\varnothing \rightarrow^*_R t \) iff \( (s) \leftrightarrow^*_\varnothing \rightarrow^*_R t \). Further, for any \( s_0, s_k \in T(\Sigma \cup K) \), if \( \pi \) is a proof \( s_0 \leftrightarrow s_1 \leftrightarrow \cdots \leftrightarrow s_k \) in \( AC \cup E \cup R \), then there is a proof \( \pi' \) \( s_0 \leftrightarrow s_1' \leftrightarrow \cdots \leftrightarrow s_k' \) in \( AC \cup E' \cup R' \) such that \( \pi \geq \pi' \).

**Proof.** The first part of the lemma, which states that the congruence on \( T(\Sigma) \) 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 \((K',E',R') = (K \cup \{c\},E',R' = R \cup \{ t \rightarrow c \})\) is obtained from the state \((K,E,R)\) by extension. Now, if \( s \leftrightarrow^*_\varnothing \rightarrow^*_R t \), then clearly \( s \leftrightarrow^*_\varnothing \rightarrow^*_R t \). Conversely, if \( s \leftrightarrow^*_\varnothing \rightarrow^*_R t \), 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 \((E \rightarrow E') \cup (R \rightarrow R')\) 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 \( s[t] \rightarrow_E u \) is replaced by a proof \( s[t] \rightarrow_R s[c] \rightarrow_E u \) and the new proof is smaller as \( \{s[t],u\} \succ^m \{s[c],u\} \).

(ii) **Simplification.** The proof \( r[s] \rightarrow_E u \) is replaced by the new proof \( r[s] \rightarrow_A r' \rightarrow_E r[t] \rightarrow_E u \).\(^{15}\) Now, \( \{r[s],u\} \succ^m \{r[t],u\} \).

(iii) **ACCollaps.** The proof \( t \rightarrow_R s \) is transformed to the smaller proof

\(^{15}\) Note that we used extended rule in specifying simplification, but for purposes of proof transformations, we only consider the original (non-extended) rules as being present in the third component.
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 \( t'' \Rightarrow_{E'} s \) (in the first component).

(iv) Orientation. In this case, \( s \leftrightarrow_E t \) is more complex than the new proof \( s \Rightarrow_R t \), and this follows from \( \{s, t\} \geq^m \{s\} \).

(v) Deletion. We have \( s \leftrightarrow_E t \) more complex than \( s \leftrightarrow_{AC} t \) because \( \{s, t\} \geq^m \{s'\} \) for every \( s' \) in \( s \leftrightarrow_{AC} t \).

(vi) Composition. We have the proof \( t \Rightarrow_R s \) transformed to the smaller proof \( t \Rightarrow_R s' \leftrightarrow_R s'' \leftrightarrow_{AC} s \). This new proof is smaller because the rewrite step \( t \Rightarrow_R s \) is more complex than (a) the rewrite step \( t \rightarrow_{E'} s' \) in the third component, (b) all proof steps in \( s'' \leftrightarrow_{AC} s \) in the first component, and (c) the rewrite step \( s'' \rightarrow_{E'} s' \) 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 \( R_\infty \), we introduce extensions to take care of clifs in proofs.

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

**Proof.** Fairness implies that all critical pairs (modulo AC) between rules in \( R_\infty \) are contained in the set \( \cup_i E_i \). Since a fairness 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 \( \pi \) in \( E_\infty \cup R_\infty \cup AC \).

We argue by contradiction that certain proof patterns can not occur in the minimal proof \( \pi \): specifically, there can be no peaks \( s \leftarrow_{AC} u \rightarrow_{AC \setminus R_\infty} t \), non-overlap clifs or variable overlap clifs.

(i) Peaks. A peak caused by a non-overlap or a variable overlap \( s \leftarrow_{R_\infty} u \rightarrow_{AC \setminus R_\infty} t \) can be transformed to a simpler proof \( s \leftrightarrow_{AC} v \leftarrow_{AC \setminus R_\infty} t \). 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. In this case, it is easy to see that \( s \leftrightarrow_{AC} s' \leftrightarrow_{CPAC(R_\infty)} t' \leftrightarrow_{AC} t \), where \( CPAC(R_\infty) \) denotes the set of all equations created by ACSuperposition and ACCollapse transition rules applied on the rules in \( R_\infty \). Since by fairness \( CPAC(R_\infty) \subseteq \cup_k E_k \), there is a proof \( s \leftrightarrow_{AC} s' \leftrightarrow_{E_k} t' \leftrightarrow_{AC} t \) for some \( k \geq 0 \). This proof, which we name \( \pi' \), is strictly smaller than the original peak. Using Lemma 7, we may
infer that there is a proof $\pi'$ in $AC \cup R_\infty$ such that $\pi'$ is strictly smaller than the original peak, a contradiction.

(ii) Cliffs. A non-overlap cliff $w[v, s] \leftrightarrow_{AC} w[u, s] \rightarrow_{AC \setminus R_\infty} w[u, t]$ can be transformed to the following less complex proof: $w[v, s] \rightarrow_{AC \setminus R_\infty} w[v, t] \leftrightarrow_{AC} w[u, t]$. Clearly, $w[v, s] \Rightarrow w[v, t]$ 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 $w[u, s] \rightarrow_{AC \setminus R_\infty} w[u, t]$ is identical to the complexity of the proof $w[v, s] \rightarrow_{AC \setminus R_\infty} w[v, t]$.

In the case of $AC$, a variable overlap cliff $s \leftrightarrow_{AC} u \rightarrow_{AC \setminus R_\infty} t$ can be eliminated in favour of the proof $s \rightarrow_{AC \setminus R_\infty} t' \leftrightarrow_{AC} t$. Note that the proof $u \rightarrow_{AC \setminus R_\infty} t$ and the proof $s \rightarrow_{AC \setminus R_\infty} t'$ 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 $\pi$ can not contain peaks $s \leftarrow_{R_\infty} u \rightarrow_{AC \setminus R_\infty}$, or, non-overlap or variable overlap cliffs $s \leftrightarrow_{AC} u \rightarrow_{AC \setminus R_\infty} t$. The cliffs arising from proper overlaps can be replaced by extended rules, as $(R_\infty)^e = R_\infty$. The minimal proof $\pi$ in $R_\infty \cup AC$ can, therefore, only be of the form $s \rightarrow_{AC \setminus R_\infty}^* t' \leftrightarrow_{AC} t'$, 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 at 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 $R_\infty$ be the set of persisting rules of a fair derivation starting from state $(\emptyset, E, \emptyset)$. Then, the set $R_\infty$ is an associative-commutative congruence closure for $E$.

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

1. The transition rules ensure that $R_\infty$ consists of only $D$-rules, $C$-rules, and $A$-rules. We prove that every constant represents some term in $T(\Sigma)$ by induction. Let $c$ be any constant in $K_\infty$. 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. As induction hypothesis we can assume that all constants added before $c$ represent a term in $T(\Sigma)$ via $R_\infty$. Therefore, there exist terms $s_1, \ldots, s_k \in T(\Sigma)$ such that $s_i \rightarrow_{AC \setminus R_\infty}^* c_i$ and hence,

$$f(s_1, \ldots, s_k) \leftrightarrow_{AC \setminus R_\infty}^* f(c_1, \ldots, c_k) \rightarrow_{\cup_i R_i} c.$$

Using Lemma 7, $f(s_1, \ldots, s_k) \leftrightarrow_{R_\infty \cup E_\infty \cup AC}^* c$. Lemma 8 shows that $E_\infty = \emptyset$, and $f(s_1, \ldots, s_k) \leftrightarrow_{AC \setminus R_\infty}^* c$.
2. Lemma 8 shows that \( AC \setminus R^\infty_{\infty} \) is ground convergent.

3. Let \( s, t \in T(\Sigma) \). Using Lemma 7, we know \( s \not\leftrightarrow^k_{E \cup AC} t \) if, and only if, \( s \not\leftrightarrow^k_{E \cup R_{\infty} \cup AC} t \). Since \( E_{\infty} = \emptyset \), Lemma 8 implies that
\[
\left. s \rightarrow^k_{AC \setminus R^\infty_{\infty}} \right\} \not\leftrightarrow^k_{AC} \left\{ t \right. \quad \text{\( \iff \)}
\]

Since \( R_{\infty} \) is finite, there exists a \( k \) such that \( R_{\infty} \subseteq R_k \). Thus, the set of persisting rules can be obtained using only finite derivations.

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 \( E \).

\[
\text{Flattening:} \quad \frac{(K, E \cup \{ s \approx t \}, R)}{(K, E \cup \{ u \approx t \}, R)}
\]

where \( s \rightarrow u \).

However, now the correctness proof given above, Lemma 7 in particular, fails as the new proof \( s \leftrightarrow_{AC \cup E_\infty} t \) of the deleted equation \( s \approx t \) is larger than the old proof \( s \leftrightarrow_{E_\infty} t \). 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 \( s \leftrightarrow_{AC \cup E_\infty} t \), 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.

As a second optimization, the extension variable of a rewrite rule can be constrained to allow for fine-grained deletion of instances of rewrite rules. For example, after deducing the critical pair \( f_c \circ_c f_2 \approx f_c \circ_c f_3 \) that arises by overlapping the rules \( f \circ_c f_2 \rightarrow f \circ_c f_3 \) and \( f \circ_c f_2 \rightarrow f \circ_c f_3 \), we can delete the instance \( f \circ_c f_2 \rightarrow f \circ_c f_3 \) of the latter rule as it has a smaller proof \( f_c \circ_c f_2 \rightarrow f_c \circ_c f_3 \approx f_c \circ_c f_3 \) using the deduced equation.

We can delete this instance by replacing the rule \( f_c \circ_c f_2 \rightarrow f_3 \) by the new rule \( f_c \circ_c f_2 \rightarrow f_3 \) if \( C \), where \( C \) is the constraint that "\( y \) is not of the form \( f (c_2, z) \)". These new constraints can be carried to new equations generated in a deduction step.

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 ACCollipe 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

The results in the previous sections establish the decidability of the word problem for ground theories presented over a signature containing finitely many associative-commutative symbols. Note that we are implicitly 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 [19], but that algorithm was not based on critical pair completion.

Thus, using the approach proposed in our paper, we can construct an $AC$ congruence closure in time $O(n \Sigma T(n))$ and space $O(n^2 + S(n))$ using an algorithm for constructing Gröbner bases for binomial ideals that uses $O(T(n))$ time and $S(n)$ space. We have not worked out the time complexity of the critical pair completion based algorithm (as presented in our 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 on 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 (Def-
inition 3), then we pick a term \( t \in T(\Sigma) \) 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 non-terminating modulo AC (see 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 \( AC \backslash R^c \) 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 \( \Sigma \) 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.

\[
\text{Compression: } \frac{(K \cup \{c\}, R \cup \{c \to t\})}{(K, R(c \to t))}
\]

where \( \langle c \to t \rangle \) denotes the (homomorphic extension of the) mapping \( c \to t \), and \( R(c \to t) \) 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 \( \Sigma \)) in the equivalence class of \( c \), and replacing \( c \) by \( t \) everywhere in \( R \).

\[
\text{Selection: } \frac{(K \cup \{c\}, R \cup \{t \to c\})}{(K, R(c \to t) \cup R^c)}
\]

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

In case Σ_{AC} = ∅, we note that R' will always be empty. We also require that terms are not flattened after the application of mapping R(c → t). 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 5. A fully-reduced congruence closure for \( E_0 \) is given by the set \( R_0 \)

\[
\begin{align*}
    a &\rightarrow c_1 & b &\rightarrow c_2 & c &\rightarrow c_3 & fc_2c_3 &\rightarrow c_4 \\
    fc_2c_4 &\rightarrow c_2 & fc_1c_3 &\rightarrow c_1 & fc_2c_2 &\rightarrow fc_1c_4 & fc_1c_2 &\rightarrow fc_1c_4 \\
    gc_1 &\rightarrow c_4
\end{align*}
\]

under the ordering \( c_2 > c_1 \) 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

\[
\begin{align*}
    fcc_4 &\rightarrow b & fac &\rightarrow a & fbb &\rightarrow fc_3c_4 \\
    fbc &\rightarrow c_4 & gc_4 &\rightarrow c_4 & fab &\rightarrow fc_4c_4
\end{align*}
\]

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

\[
\begin{align*}
    fc(fbc) &\rightarrow b & fac &\rightarrow a & fbb &\rightarrow f(fbc)(fbc) \\
    fbcX &\rightarrow f(fbc)X & gc &\rightarrow fbc & fab &\rightarrow fa(fbc)
\end{align*}
\]

The relation \( \rightarrow_{AC,R_1} \) 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 \( \sigma \) is a substitution that maps variables to the sequences. If \( \sigma \) 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 \( \rho \) be a ground rule of the form \( f(t_1, \ldots, t_k) \rightarrow g(s_1, \ldots, s_m) \) where \( f \in \Sigma_{AC} \). We define the sequence extension \( \rho^+ \) of \( \rho \) as \( f(t_1, \ldots, t_k, X) \rightarrow f(s_1, \ldots, s_m, X) \) if \( f = g \), and as \( f(t_1, \ldots, t_k, X) \rightarrow f(g(s_1, \ldots, s_m), X) \) 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 \( R^+ \) 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 $s \rightarrow_{P/R}^* t$ if there exists a rule $l \rightarrow r \in R$, and a sequence substitution $\sigma$ such that $s = C[l^r]$, $l' \rightarrow^*_P l \sigma$, $r' = r \sigma$, and $t = C[r^r]$.

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 $\rho$ is $fac \rightarrow a$, then the term $f(f(a, b), c)$ is not reducible by $\rightarrow_{P/\rho}$, although it is reducible by $\rightarrow_{AC/\rho}$. The term $f(f(a, b), c, a)$ can be rewritten by $\rightarrow_{P/\rho}$ to $f(f(a, b), a)$.

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

$$fabc \rightarrow_{P/R_1^*} fa(fbc)c \rightarrow_{P/R_1^*} fab \rightarrow_{P/R_1^*} fa(fbc).$$

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

$$fc_1c_2c_3 \rightarrow_{P/R_0} fc_1c_2c_3 \rightarrow_{P/R_0} fc_1c_2 \rightarrow_{P/R_0} fc_1c_4.$$

A $P \setminus R_0^*$ proof step can be projected onto a $P \setminus R_1^*$ 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 $\Sigma$ which is convergent modulo $P$ and which defines the same equational theory over fully flattened terms over $\Sigma$. First note that any derivation starting from the state $(K, R)$, where $R$ is an AC congruence closure over $\Sigma$ 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 $(K_\infty, R_\infty)$, if $K_\infty = \emptyset$, then, $R_\infty$ is a rewrite system over $\Sigma$, the original signature. We will show that $K_\infty$ is actually empty, and that the reduction relation $P \setminus R_\infty^*$ 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 $P/\rho$ and $P/\rho^*$ for every other rule $\rho$ in $R$; and, $R$ is terminating (modulo $P$) if $P \setminus R^*$ is.
LEMMA 9. Let \((K_1, R_1 = R_1' \sigma)\) be obtained from \((K_0 = K_1 \cup \{c\}, R_0 = R_0' \cup \{c \rightarrow u\})\) using compression, where \(\sigma = \{c \mapsto u\}\). 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 \(s \rightarrow_{P \setminus R_0} t\), then \(s \sigma \rightarrow_{P \setminus R_1}^{0,1} t \sigma\).

(b) For any two terms \(s, t \in \mathcal{T}(\Sigma \cup K_1)\), if \(s \rightarrow_{P \setminus R_1} t\), then \(s \theta \rightarrow_{P \setminus R_0}^\theta t \theta\), where \(\theta = \{u \mapsto c\}\).\(^{16}\)

(c) \(R_1\) is left-reduced and terminating.

Proof. To prove (a), let \(s, t\) be two terms over \(\Sigma \cup K_0\) such that \(s = C[l_0^\sigma]_0, l_0^\sigma \rightarrow_P l_0^\sigma^+\), and \(t = C[r_0^\sigma^+]\), where \(l_0 \rightarrow r_0\) is a sequence extension of some rule in \(R_0\) and \(\sigma^+\) is a sequence substitution. Clearly, \((l_0^\sigma^+) \sigma = (l_0^\sigma) (\sigma^+ \sigma) = l_1 (\sigma^+ \sigma)\), and similarly \((r_0^\sigma^+) \sigma = (r_0^\sigma) (\sigma^+ \sigma) = r_1 (\sigma^+ \sigma)\), where either \(l_1 = r_1\), or, \(l_1 \rightarrow r_1\) is a sequence extension of some rule in \(R_1\). In the first case, \(s \sigma \rightarrow_P t \sigma\) and in the second case, \(s \sigma \rightarrow_{P \setminus R_1} t \sigma\).

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 \(s \rightarrow_{R_1} t\), then \(s \leftrightarrow_{R_0} t\). Therefore, \(s \theta \rightarrow_{\{c \rightarrow u\}}^\theta s \leftrightarrow_{R_0}^\theta t \leftrightarrow_{\{c \rightarrow u\}}^\theta t \theta\).

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 \((K_1, R_1 = R_1' \sigma \cup R^e)\) be obtained from \((K_0 = K_1 \cup \{c\}, R_0 = R_0' \cup \{u \mapsto c\})\) using selection, where \(\sigma = \{c \mapsto u\}\). 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 \(s \rightarrow_{P \setminus R_0} t\), then \(s \sigma \rightarrow_{P \setminus R_1}^{0,1} t \sigma\).

(b) For any two terms \(s, t \in \mathcal{T}(\Sigma \cup K_1)\), if \(s \rightarrow_{P \setminus R_1} t\), then \(s \theta \rightarrow_{P \setminus R_0}^\theta t \theta\), where \(\theta = \{u \mapsto c\}\).

(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 \(f \in \Sigma_{AC}\), \(s \leftrightarrow_P C[f(u_1, \ldots, u_k, X \sigma^+)]\), and \(t = C[f(c, X \sigma^+]\), the proof

\[
\begin{align*}
  s \sigma & \leftrightarrow_P (C \sigma) [f(u_1, \ldots, u_k, X \sigma^+ \sigma)] \\
  \rightarrow_{P \setminus R_1} (C \sigma) [f(f(u_1, \ldots, u_k), X \sigma^+ \sigma)] &= t \sigma
\end{align*}
\]

\(^{16}\) Note that if \(\theta\) is defined by \(\langle fab \mapsto c_0\rangle\), then \(fabc\theta = fabc\), but \(f(fab)\theta = fbc\).

---

\(\text{final.tex; 9/02/2002; 14:52; p.39}\)
uses the rule in the set $R'$.

To prove (b), let $s, t$ be two terms over $\Sigma \cup K_1$ such that $s \leftrightarrow^+_{p} C[l_{i}, \sigma^+]$ and $t = C[r_{0} \sigma^+]$, where $l_{i} \rightarrow r_{1}$ is (a sequence extension of) some rule in $R_1$. First consider the case when $l_{i} = f(u_{1}, \ldots, u_{k}, X) \rightarrow f(f(u_{1}, \ldots, u_{k}), X) = r_{1}$ is the rule in $R'$, and since $X\sigma^+$ is non-empty,

$$s\theta \leftrightarrow^+_{p} (C\theta)[f(u_{1}, \ldots, u_{k}, X \sigma^{\theta})] \rightarrow\rightarrow^{p}_{R_0} (C\theta)[f(c, X \sigma^{\theta})] = t\theta.$$ 

In the other case, assume $l_{i} = l_{0}\sigma$ and $r_{1} = r_{0}\sigma$, where $l_{0} \rightarrow r_{0}$ is (an extension of) some rule different from $u \rightarrow c$ in $R_0$. Since $R_0$ is left-reduced modulo $P$, $s\theta \leftrightarrow^+_{p} C[[l_{0}\sigma] \sigma^{\theta}] = (C\theta)[l_{0}(\sigma^{\theta})]$, and therefore we have,

$$s\theta \leftrightarrow^+_{p} (C\theta)[l_{0}(\sigma^{\theta})] \rightarrow R_0 (C\theta)[r_{0}(\sigma^{\theta})] \rightarrow^+_{[u \rightarrow c]} (C[[r_{0}\sigma] \sigma^{\theta}]) = t\theta.$$ 

Since $R_0$ is terminating, it follows from (b) that $R_1$ is also terminating. Finally, to prove that $R_1 = R_0 \sigma \cap R'$ 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 $R_0 \sigma \cap R'$ 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 $A\Sigma$) 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 $T(\Sigma)$ such that $t \rightarrow^{+}_{D_0/C_0} \cdot c^{19}$. 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 $t \in T(\Sigma)$ such that $t \rightarrow^{+}_{D_0/C_0} c$. It follows from convergence of $A\Sigma \cap R_0$ that,

$$t \rightarrow^{+}_{D_0/C_0} c \iff c.$$ 

Since $R_0$ is a left-reduced (modulo $A\Sigma$) congruence closure, therefore $R_0$ is left-reduced and terminating modulo $P$, and hence Lemma 9 and

---

17 $\rightarrow_{D/C} = (\leftrightarrow_{D} \circ \rightarrow_{P} \circ \leftrightarrow_{P})$. 
18 If $\Sigma_{A\Sigma} = \emptyset$, then this condition is satisfied by any abstract congruence closure. 
19 Note that if the non-extended form of an $A$-rule is a $D$-rule, it is included in the set $D_0$. 

---
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

$$t \rightarrow^{s}_{P_i | R_i} \Leftrightarrow t \rightarrow^{s}_{P_i | R_i} c,$$

where the right-hand side of each rule used in the above proof is either a constant or a term in $T(\Sigma)$. 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

$$t \rightarrow^{s}_{P_i | R_i} c.$$  

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 $(K_\infty, R_\infty)$ 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 $T(\Sigma)$ such that $t \rightarrow^{s}_{P_0 \cup \Sigma_0} c$, then (i) $K_\infty = \emptyset$, (ii) $\rightarrow^{s}_{P \cup \Sigma}$ is ground convergent on all fully flattened terms over $\Sigma$, and (iii) the equivalence over flattened $T(\Sigma)$ terms defined by this relation is the same as the equational theory induced by $R \cup AC$ over flattened $T(\Sigma)$ terms.

Proof. Statement (i) is a consequence of Lemma 11. It follows from Lemma 9(c) and Lemma 10(c) that $\rightarrow^{s}_{P_i | R_\infty}$ is terminating. Let $s, t$ be fully flattened terms over $T(\Sigma)$ such that $s \leftrightarrow^{s}_{P \cup R_\infty} t$. Using Lemma 9(b) and Lemma 10(b), it follows that $s \leftrightarrow^{s}_{AC \cup R} t$. This, in turn, implies that $s \rightarrow^{s}_{AC \cup R} \Leftrightarrow s \leftrightarrow^{s}_{AC} \Leftrightarrow s \rightarrow^{s}_{AC \cup R} t$, and hence, by projecting this proof onto fully flattened terms (normalize each term in the proof by $P_i$), we obtain a proof $s \rightarrow^{s}_{P_i | R_\infty} \Leftrightarrow s \rightarrow^{s}_{P_i | R_\infty} t$, 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 $s \rightarrow^{s}_{P_i | R_\infty} \Leftrightarrow s \rightarrow^{s}_{P_i | R_\infty} t$. This establishes claims (ii) and (iii).

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

7. Conclusion

Abstract Congruence Closure

Kapur [18] considered the problem of casting Shostak’s congruence closure [28] 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 [29]. Plaisted et. al. [25] 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 [29] uses a particular implementation of congruence closure due to which some post-processing 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 [29] 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 [29]. Finally, the approach in [29] is to solve the problem “by abandoning rewriting techniques altogether and recasting the problem in graph theoretic terms.” On the other hand, we stick to rewriting over extensions.

Plaisted and Sattler-Klein [25] 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
Abstract Congruence Closure

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. Phaisted 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 [20], [22] and [14]. 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 [22], 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 [21, 19].

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 [14]. 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 [14], 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])20. 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).

20 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.
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 $\Sigma$ in our 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 [16, 17] 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, Rakesh Verma, and the anonymous referees for their helpful comments.

References

