A Modular Associative Commutative (AC) Congruence Closure Algorithm

Deepak Kapur
Department of Computer Science, University of New Mexico, Albuquerque, NM, USA

Abstract

Algorithms for computing congruence closure of ground equations over uninterpreted symbols and interpreted symbols satisfying associativity and commutativity (AC) properties are proposed. The algorithms are based on a framework for computing the congruence closure by abstracting nonflat terms by constants as proposed first in Kapur's congruence closure algorithm (RTA97). The framework is general, flexible, and has been extended also to develop congruence closure algorithms for the cases when associative-commutative function symbols can have additional properties including idempotency, nilpotency and/or have identities, as well as their various combinations. The algorithms are modular; their correctness and termination proofs are simple, exploiting modularity. Unlike earlier algorithms, the proposed algorithms neither rely on complex AC compatible well-founded orderings on nonvariable terms nor need to use the associative-commutative unification and extension rules in completion for generating canonical rewrite systems for congruence closures. They are particularly suited for integrating into Satisfiability modulo Theories (SMT) solvers.

2012 ACM Subject Classification Software and its engineering; Theory of computation; Mathematics of computing

Keywords and phrases Congruence Closure, Associative and Commutative, Word Problems, Finitely Presented Algebras, Equational Theories

Digital Object Identifier 10.4230/LIPIcs.FSCD.2021.15

Funding Research partially supported by the NSF award: CCF-1908804.

Acknowledgements Heartfelt thanks to the referees for their reports which substantially improved the presentation.

1 Introduction

Equality reasoning arises in many applications including compiler optimization, functional languages, and reasoning about data bases, most importantly, reasoning about different aspects of software and hardware. The significance of the congruence closure algorithms on ground equations in compiler optimization and verification applications was recognized in the mid 70's and early 80's, leading to a number of algorithms for computing the congruence closure of ground equations on uninterpreted function symbols [8, 28, 25]. Whereas congruence closure algorithms were implemented in earlier verification systems [28, 25, 19, 32], their role has become particularly critical in Satisfiability modulo Theories (SMT) solvers as a glue to combine different decision procedure for various theories.

We present algorithms for the congruence closure of ground equations which in addition to uninterpreted function symbols, have symbols with the associative (A) and commutative (C) properties. Using these algorithms, it can be decided whether another ground equation follows from a finite set of ground equations with associative-commutative (AC) symbols and uninterpreted symbols. Canonical forms (unique normal forms) can be associated with congruence classes. Further, a unique reduced ground congruence closure presentation can be associated with a finite set of ground equations, enabling checks whether two different finite sets of ground equations define the same congruence closure or one is contained in the other. In the presence of disequations on ground terms with AC and uninterpreted symbols, a finite set of ground equations and disequations can be checked for satisfiability.

The main contributions of the paper are (i) a modular combination framework for the congruence closure of ground equations with multiple AC symbols, uninterpreted symbols, and constants, leading to (ii) modular and simple algorithms that can use flexible termination orderings on ground terms and do not need to use AC/E unification algorithms for generating canonical forms; (iii) the termination and correctness proofs of these algorithms are modular and easier. The key insights are based on extending the author's previous work presented in [13, 14]: introduction of new constants for nested subterms, resulting in flat and constant equations, extended to purification of mixed subterms with many AC symbols by flattening AC ground terms and introducing new constants for pure AC terms in each AC symbol, resulting in disjoint subsets of ground equations on single AC symbols with shared constants. The result of this transformation is a finite union of disjoint subsets of ground equations with shared constants: (i) a finite set of constant equations, (ii) a finite set of flat equations with uninterpreted symbols, and (iii) for each AC symbol, a finite set of equations on pure flattened terms in a single AC symbol.

With the above decomposition, reduced canonical rewrite systems are generated for each of the subsystems using their respective termination orderings that extend a common total ordering on constants. A combination of the reduced canonical rewrite systems is achieved by propagating constant equalities among various rewrite systems; whenever new implied constant equalities are generated, each of the reduced canonical rewrite systems must be updated with additional computations to ensure their canonicity. Due to this modularity and factoring/decomposition, the termination and correctness proofs can be done independently of each subsystem, providing considerable flexibility in choosing termination orderings.

The combination algorithm terminates when no additional implied constant equalities are generated. Since there are only finitely many constants in the input and only finitely many constants are needed for purification, the termination of the combination algorithm is guaranteed. The result is a reduced canonical rewrite system corresponding to the AC congruence closure of the input ground equations, which is unique for a fixed family of total orderings on constants and different pure AC terms in each AC symbol. The reduced canonical rewrite system can be used to generate canonical signatures of ground terms with respect to the congruence closure.

The framework provides flexibility in choosing orderings on constants and terms with different AC symbols, enabling canonical forms suitable for applications instead of restrictions imposed due to the congruence closure algorithms. Interpreted AC symbols can be further enriched with properties including idempotency, nilpotency, existence of identities, and simply commutativity, without restrictions on orderings on mixed terms. Termination and correctness proofs of congruence closure algorithms are modular and simple in contrast to complex arguments and proofs in [5, 24]. These features of the proposed algorithms make them attractive for integration into SMT solvers as their implementation does not need heavy duty infrastructure including AC unification, extension rules, and AC compatible orderings.

The next subsection contrasts in detail, the results of this paper with the previous methods, discussing the advantages of the proposed framework and the resulting algorithms. Section 2 includes definitions of congruence closure with uninterpreted and interpreted symbols. This is followed by a review of key constructions used in the congruence closure algorithm over uninterpreted symbols as proposed in [13]. Section 3 introduces purification and flattening of ground terms with AC and uninterpreted symbols by extending the signature and introducing new constants. This is followed by an algorithm first reported in [11] for computing the congruence closure and the associated canonical rewrite system from ground equations with a single AC symbol and constants. It is shown how additional properties of AC symbols such as idempotency, nilpotency and identity can be integrated into the algorithm. In the next subsection, an algorithm for computing congruence closure of AC ground equations with multiple AC symbols and constants is presented. Section 4 generalizes to the case of combination of AC symbols and uninterpreted symbols. Section 5 discusses a variety of examples illustrating the proposed algorithms. Section 6 illustrates the power and elegance of the proposed framework by demonstrating how the congruence closure algorithm for two AC symbols can be further generalized to get a Gröbner basis algorithm on polynomial ideals over the integers. Section 7 concludes with some ideas for further investigation. Appendix includes proofs of some of the results in the paper.

1.1 Related Work

Congruence closure algorithms have been developed and analyzed for over four decades [8, 28, 25]. The algorithms presented here use the framework first informally proposed in [13] for congruence closure in which the author separated the algorithm into two parts: (i) constant equivalence closure, and (ii) nonconstant flat terms related to constants by flattening nested terms by introducing new constants to stand for them, and (iii) update nonconstant rules as constant equivalence closure evolves. This simplified the presentation, the correctness argument as well as the complexity analysis, and made the framework easier to generalize to other settings including conditional congruence closure [14] and semantic congruence closure [1]. Further, it enables the generation of a reduced unique canonical rewrite system for a congruence closure, assuming a total ordering on constants; most importantly, the framework gives freedom in choosing orderings on ground terms, leading to desired canonical forms appropriate for applications.

To generate congruence closure in the presence of AC symbols, the proposed framework builds on the author and his collaborators' work dating back to 1985, where they demonstrated how an ideal-theoretic approach based on Gröbner basis algorithms could be employed for word problems and unification problems over commutative algebras [11].

Congruence closure algorithms on ground equations with interpreted symbols can be viewed as special cases of the Knuth-Bendix completion procedure [20] on (nonground) equations with universal properties characterizing the semantics of the interpreted symbols. In case of equations with AC symbols, Peterson and Stickel's extension of the Knuth-Bendix completion [27] using extension rules, AC unification and AC compatible orderings can be used for congruence closure over AC symbols. For an arbitrary set E of universal axioms characterizing the semantics of interpreted symbols, E-completion with coherence check and E-unification along with E-compatible orderings need to be used. Most of the general purpose methods do not terminate in general. Even though the Knuth-Bendix procedure can be easily proved to terminate on ground equations of uninterpreted terms, that is not necessarily case for its extensions for other ground formulas.

In [6], a generalization of the Knuth-Bendix completion procedure [20] to handle AC symbols [27] is adapted to develop decision algorithms for word problems over finitely presented commutative semigroups; this is equivalent to the congruence closure of ground equations with a single AC symbol on constants. Related methods using extension rules introduced to handle AC symbols and AC rewriting for solving word problems over other finite presented commutative algebras were subsequently reported in [29].

In [24], the authors used the completion algorithm discussed in [11] and a total AC-compatible polynomial reduction ordering on congruence classes of AC ground terms to establish the existence of a ground canonical AC system first with one AC symbol. To extend their method to multiple AC symbols, particularly the instances of distributivity property relating ground terms in two AC symbols $*$ and $+$: the authors had to combine an AC-compatible total reduction ordering on terms along with complex polynomial interpretations with polynomial ranges, resulting in a complicated proof to orient the distributivity axiom from left to right. Using this highly specialized generalization of polynomial orderings, it was proved in [24] that every ground AC theory has a finite canonical system which also serves as its congruence closure.

The proposed approach, in contrast, is orthogonal to ordering arguments on AC ground terms; instead a total ordering on constants in the extended signature is extended to many different possible orderings on pure terms with a single AC symbol is sufficient to compute a canonical ground AC rewrite system. Different orderings on AC terms with different AC symbols can be used; for example, for ground terms of an AC symbol $+$ could be oriented in a completely different way than ground terms for another AC symbol $*$. Instances of the distributivity property expressed on different AC ground terms can also be oriented in different nonuniform ways. This leads to flexible ordering requirements on uninterpreted and interpreted symbols based on the properties desired of canonical forms.

In [21], a different approach was taken for computing a finite canonical rewrite system for ground equations on AC symbols. Marche first proved a general result about AC ground theories that for any finite set of ground equations with AC symbols, if there is an equivalent canonical rewrite system modulo AC, then that rewrite system must be finite. He gave an AC completion procedure, which does not terminate even on ground equations; he then proved its termination on ground equations with AC symbols using a special control on its inference rules using a total ordering on AC ground terms in [24]. Neither in [24] nor in [21], any explicit mention is made of uninterpreted symbols appearing in ground equations.

Similar to [6], several approaches based on adapting Peterson and Stickel's generalization of the Knuth-Bendix completion procedure to consider special ground theories have been reported [29, 21]. In [4, 5], the authors adapted Kapur's congruence closure [13] using its key ideas to an abstract inference system ($Table$ for new constant symbols defining flat terms introducing during flattening of nested nonconstant terms in $Make\_Rule$ were called D-rule for defining a flat term and C-rule for introducing a new constant symbol). Various congruence closure algorithms, including Sethi, Downey and Tarjan [8], Nelson and Oppen [25] and Shostak [28], from the literature can be expressed as different combinations of these inference steps. They also proposed an extension of this inference system to AC function symbols, essentially integrating it with [27] of the Knuth-Bendix completion procedure using extension rules, adapted to ground equations with AC symbols. All of these approaches based on Paterson and Stickel's generalization used extension rules introduced in [27] to define rewriting modulo AC theories so that a local-confluence test for rules with AC symbols could be developed using AC unification. During completion on ground terms, rules with variables appear in intermediate computations. All of these approaches suffer from having to consider many unnecessary inferences due to extension rules and AC unification, as it is well-known that AC unification can generate doubly exponentially many unifiers [18].

An approach based on normalized rewriting was proposed in [22] and decision procedures were reported for ground AC theories with AC symbols satisfying additional properties including idempotency, nilpotency and identity as well as their combinations. This was an attempt to integrate Le Chenadec's method [29] for finitely presented algebraic structures with Peterson and Stickel's AC completion, addressing weaknesses in E-completion and constrained rewriting, while considering additional axioms of AC symbols, including identity, idempotency and nilpotency for which termination orderings are difficult to design. However, that approach had to redefine local confluence for normalized rewriting and normalized critical pairs, leading to a complex completion procedure whose termination and proof of correctness needed extremely sophisticated machinery of normalized proofs.

The algorithms presented in this paper, in contrast, are very different and are based on an approach first presented in [11] by the author with his collaborators. Their termination and correctness proofs are based on the termination and correctness proofs of a congruence closure algorithm for uninterpreted symbols (if present) and the termination and correctness of an algorithm for deciding the word problems of a finitely presented commutative semigroup using Dickson's Lemma. Since the combination is done by propagating equalities on shared constants among various components, the termination and correctness proofs of the combination algorithm become much easier since there are only finitely many constants to consider, as determined by the size of the input ground equations.

A detailed comparison leads to several reasons why the proposed algorithms are simpler, modular, easier to understand and prove correct: (i) there is no need in the proposed approach to use extension rules whereas almost all other approaches are based on adapting AC/E completion procedures for this setting requiring considerable/sophisticated infrastructure including AC unification and E/Normalized rewriting As a result, proofs of correctness and termination become complex using heavy machinery including proof orderings and normalized proof methods not to mention arguments dealing with fairness of completion procedures. (ii) all require complex total AC compatible orderings. In contrast, ordering restrictions in the proposed algorithms are dictated by individual components–little restriction for the uninterpreted part, independent orderings on $+$-monomials for each AC symbol $+$ insofar as orderings on constants are shared by all parts, thus giving considerable flexibility in choosing termination orderings. In most related approaches except for [24], critical pairs computed using expensive AC unification steps are needed, which are likely to make the algorithms inefficient; it is well-known that many superfluous critical pairs are generated due to AC unification. These advantages make us predict that the proposed algorithms can be easily integrated with SMT solvers since they do not require sophisticated machinery of AC-unification and AC-compatible orderings, extension rules and AC completion.

2 Preliminaries

Let F be a set of function symbols including constants and $GT(F)$ be the ground terms constructed from F; sometimes, we will write it as $GT(F, C)$ to highlight the constants of F. We will abuse the terminology by calling a k-ary function symbol as a function symbol if $k > 0$ and constant if $k = 0$. A function term is meant to be a nonconstant term with a nonconstant outermost symbol. Symbols in F are either uninterpreted (to mean no semantic property of such a function is assumed) or interpreted satisfying properties expressed as universally quantified equations (called universal equations).

2.1 Congruence Relations

▶ **Definition 1.** *Given a finite set $S = \{a_i = b_i | 1 \leq i \leq m\}$ of ground equations where $a_i, b_i \in GT(F)$, the congruence closure $CC(S)$ is inductively defined as follows: (i) $S \subseteq CC(S)$, (ii) for every $a \in GT(F)$, $a = a \in CC(S)$, (iii) if $a = b \in CC(S)$, $b = a \in CC(S)$, (iv) if $a = b$ and $b = c \in CC(S)$, $a = c \in CC(S)$, and (v) for every nonconstant $f \in F$ of arity $k > 0$, if for all $1 \leq k$, $a_i = b_i \in CC(S)$, then $f(a_1, \cdots, a_k) = f(b_1, \cdots, b_k) \in CC(S)$. Nothing else is in $CC(S)$.*

$CC(S)$ is thus the smallest relation that includes S and is closed under reflexivity, symmetry, transitivity, and under function application. It is easy to see that $CC(S)$ is also the equational theory of S [2, 1].

2.2 Kapur's Congruence Closure Algorithm for Uninterpreted Symbols

The algorithm in [13, 14] for computing congruence closure of a finite set S of ground equations serves as the main building block in this paper. The algorithm extends the input signature by introducing new constant symbols to recursively stand for each nonconstant subterm and generates two types of equations: (i) constant equations, and (ii) flat terms of the form $f(c_1, \cdots, c_k)$ equal to constants. A disequation is converted to a disequation on constants by introducing new symbols for the terms. It can be proved that the congruence closure of ground equations on the extended signature when restricted to the original signature, is indeed the congruence closure of the original equations [1].

Using a total ordering on constants (typically with new constants introduced to extend the signature being smaller than constants from the input), the output of the algorithm in [13, 14] is a reduced canonical rewrite system R_S associated with $CC(S)$ (as well as S) that includes *function*, also called *flat* rules of the form $f(c_1, \cdots, c_k) \rightarrow d$ and *constant* rules $c \rightarrow d$ such that no two left sides of the rules are identical; further, all constants are in canonical forms. As proved in [13] (see also [26]),

▶ **Theorem 2** ([13]). *Given a set S of ground equations, a reduced canonical rewrite system R_S on the extended signature, consisting of nonconstant flat rules $f(c_1, \ldots, c_k) \rightarrow d$, and constant rules $c \rightarrow d$, can be generated from S in $O(n^2)$ steps. The complexity can be further reduced to $O(n * log(n))$ steps if all function symbols are binary or unary. For a given total ordering $\gg$ on constants, R_S is unique for S, subject to the introduction of the same set of new constants for nonconstant subterms.*

As shown in [8] (see [26]), function symbols of arity > 2 can be encoded using binary symbols using additional linearly many steps.

The canonical form of a ground term g using R_S is denoted by $\hat{g}$ and is its canonical signature (in the extended language). Ground terms g_1, g_2 are congruent in $CC(S)$ iff $\hat{g_1} = \hat{g_2}$.

2.3 AC Congruence Closure

The above definition of congruence closure $CC(S)$ is extended to consider interpreted symbols. Let IE be a finite set of universally quantified equations with variables, specifying properties of interpreted function symbols in F. For example, the properties of an AC symbol f are: $\forall x, y, z,\ f(x, y) = f(y, x),\ f(x, f(y, z)) = f(f(x, y), z)$. An idempotent symbol g, for another example, is specified as $\forall x,\ g(x, x) = x$. To incorporate the semantics of these properties:

(vi) from a universal axiom $s = t \in IE$, for each variable x in s, t, for any ground substitution σ, i.e., $\sigma(x) \in GT(F)$, $\sigma(s) = \sigma(t) \in CC(S)$.

$CC(S)$ is thus the smallest relation that includes S and is closed under reflexivity, symmetry, transitivity, function application, and the substitution of variables in IE by ground terms. $CC(S)$ is also the ground equational theory of S.

Given a finite set S of ground equations with uninterpreted and interpreted symbols, the congruence closure membership problem is to check whether another ground equation $u = v \in CC(S)$ (meaning semantically that $u = v$ follows from S, written as $S \models u = v$). A related problem is whether given two sets S_1 and S_2 of ground equations, $CC(S_2) \subseteq CC(S_1)$, equivalently $S_1 \models S_2$. Birkhoff's theorem relates the syntactic properties, the equational theory, and the semantics of S.

If S also includes ground disequations, then besides checking the unsatisfiability of a finite set of ground equations and disequations, new disequations can be derived in case S is satisfiable. The inference rule for deriving new disequations for an uninterpreted symbol is:

$f(c_1, \ldots, c_k) \neq f(d_1, \cdots, d_k) \implies (c_1 \neq d_1 \vee \cdots \vee c_k \neq d_k)$.

In particular, if f is unary, then the disequation is immediately derived.

Disequations in case of interpreted symbols generate more interesting formulas. In case of a commutative symbol f, for example, the disequation $f(a,b) \neq f(c,d)$ implies $(a \neq c \vee a \neq c \vee b \neq d \vee c \neq d)$. For an AC symbol g, as an example, the disequation $g(a, g(b,c)) \neq g(a, g(a, g(a,c)))$ implies $(a \neq g(a,c) \vee b \neq c \vee b \neq g(a,a) \vee \cdots)$.

To emphasize the presence of AC symbols, let $ACCC(S)$ stand for the AC congruence closure of S with AC symbols; we will interchangeably use $ACCC(S)$ and $CC(S)$.

2.4 Flattening and Purification

Let F include a finite set C of constants, a finite set F_U of uninterpreted symbols, and a finite set F_{AC} of AC symbols. i.e., $F = F_{AC} \cup F_U \cup C$.

Following [13], ground equations in $GT(F)$ with AC symbols are transformed into three kinds of equations by introducing new constants for subterms: (i) constant equations of the form $c = d$, (ii) flat equations with uninterpreted symbols of the form $h(c_1, \cdots, c_k) = d$, and (iii) for each $f \in F_{AC}$, $f(c_1 \cdots, c_j) = f(d_1, \cdots, d_{j'})$, where c's, d's are constants in C, $h \in F_U$, and every AC symbol f is viewed to be variadic (including $f(c) = c$). Nested subterms of every AC symbol f are repeatedly flattened: $f(f(s_1, s_2), s_3)$ to $f(s_1, s_2, s_3)$ and $f(s_1, f(s_2, s_3))$ to $f(s_1, s_2, s_3)$ until all arguments to f are constants or nonconstant terms with outermost symbols different from f. Nonconstant arguments of a mixed AC term $f(t_1, \cdots, t_k)$ are transformed to $f(u_1, \cdots, u_k)$, where u_i's are new constants, with $t_i = u_i$ if t_i is not a constant. A subterm whose outermost function symbol is uninterpreted, is also flattened by introducing new constants for their nonconstant arguments. These transformations are recursively applied on the equations with new constants.

New constants are introduced only for nonconstant subterms and their number is minimized by introducing a single new constant for each distinct subterm irrespective of its number of occurrences (which is equivalent to representing terms by directed acyclic graphs (DAGs) with full sharing whose each non-leaf node is assigned a distinct constant). As an example, $((f(a,b) * g(a)) + f(a + (a + b), (a * b) + b)) * ((g(a) + ((f(a,b) + a) + a)) + (g(a) * b)) = a$ is purified and flattened with new constants u_i's, resulting in $\{f(a,b) = u_1, g(a) = u_2, u_1 * u_2 = u_3, a + a + b = u_4, a * b = u_5, u_5 + b = u_6, f(u_4, u_6) = u_7, u_3 + u_7 = u_8, u_2 * b = u_9, u_2 + u_1 + a + a + u_9 = u_{10}, u_7 * u_{10} = a$

The arguments of an AC symbol are represented as a multiset since the order does not matter but multiplicity does. For an AC symbol f, let $f(M)$ be a flattened term $f(a_1, \cdots, a_k)$ with $M = \{\{a_1, \cdots, a_k\}\}$, a multiset of constants; $f(M)$ is called an **f-monomial**. In case f has its identity e, i.e., $f(x, e) = x$, then e is written as is, or $f(\{\{\}\})$. A singleton constant c is written as is or equivalently $f(\{\{c\}\})$. An f-monomial $f(M_1)$ is equal to $f(M_2)$ iff the multisets M_1 and M_2 are equal.

Without any loss of generality, the input to the algorithms below are assumed to be the above set of flattened ground equations on constants.

3 Congruence Closure with Associative-Commutative (AC) Functions

The focus in this section is on interpreted symbols with the associative-commutative properties; later, uninterpreted symbols are considered.

Checking whether a ground equation on AC terms is in the congruence closure $ACCC(S)$ of a finite set S on ground equations is the word problem over finitely presented commutative algebraic structures, presented by S characterizing their interpretations as discussed in [11, 29, 6]. In the presence of disequations over AC ground terms, one is also interested in determining whether the set of ground equations and disequations is satisfiable or not.

Another goal is to associate a reduced canonical rewrite system as a unique presentation of $ACCC(S)$ and a canonical signature with every AC congruence class in the AC congruence closure of a satisfiable S.

For a single AC symbol f and a finite set S of monomial equations $\{f(M_i) = f(M_i') | 1 \leq i \leq k\}$, $ACCC(S)$ is the reflexive, symmetric and transitive closure of S closed under f: if $f(M_1) = f(M_2)$ and $f(N_1) = f(N_2)$ in $ACCC(S)$, then $f(M_1 \cup N_1) = f(M_2 \cup N_2)$ is also in $ACCC(S)$. In case of multiple AC symbols, for every AC symbol $g \neq f$, $g(f(M_1), f(N_1)) = g(f(M_2), f(N_2)) \in ACCC(S)$.

3.1 Congruence Closure with a Single AC Symbol

As in [13], we follow a rewrite-based approach for computing the AC congruence closure $ACCC(S)$ by generating a canonical rewrite system from S. To make rewrite rules from equations in S, a total ordering $\gg$ on the set C of constants is extended to a total ordering on f-monomials and denoted as $\gg_f$. One of the main advantages of the proposed approach is the flexibility in using termination orderings on f-monomials, both from the literature on termination orderings on term rewriting systems as well as well-founded orderings (also called admissible orderings) from the literature on symbolic computation including Gröbner basis.

Using the terminology from the Gröbner basis literature, an ordering $\gg_f$ on the set of f-monomials, $GT(\{f\}, C)$, is called *admissible* iff i) $f(A) \gg_f f(B)$ if the multiset B is a proper subset of the multiset A (subterm property) for any nonempty multiset M, and (ii) for any multiset B, $f(A_1) \gg_f f(A_2) \implies f(A_1 \cup B) \gg_f f(A_2 \cup B)$ (the compatibility property). $f(\{\{\}\})$ may or may not be included in $GT(F)$ depending upon an application.

From S, a rewrite system R_S is associated with S by orienting nontrivial equations in S (after deleting trivial equations $t = t$ from S) using $\gg_f$: a ground equation $f(A_1) = f(A_2)$ is oriented into a terminating rewrite rule $f(A_1) \to f(A_2)$, where $f(A_1) \gg_f f(A_2)$. The rewriting relation induced by this rewrite rule is defined below.

▶ **Definition 3.** *A flattened term $f(M)$ is* rewritten *in one step, denoted by $\to_{AC}$ (or simply $\to$), using a rule $f(A_1) \to f(A_2)$ to $f(M')$ iff $A_1 \subseteq M$ and $M' = (M - A_1) \cup A_2$, where $-, \cup$ are operations on multisets.*

Given that $f(A_1) \gg_f f(A_2)$, it follows that $f(M) \gg_f f(M')$, implying the rewriting terminates. Standard notation and concepts from [2] are used to represent and study properties of the reflexive and transitive closure and transitive closure of $\to_{AC}$ induced by R_S; the reflexive, symmetric and transitive closure of $\to_{AC}$ is the AC congruence closure $ACCC(S)$ of S. Below, the subscript AC is dropped from $\to_{AC}$, f is dropped from S_f and $\gg_f$ whenever obvious from the context.

A rewrite relation $\to$ defined by R_S is called terminating iff there are no infinite rewrite chains of the form $t_0 \to t_1 \to \cdots \to t_k \to \cdots$. A rewrite relation $\to$ is *locally confluent* iff for any term t such that $t \to u_1, t \to u_2$, there exists v such that $u_1 \to^* v, u_2 \to^* v$. $\to$ is confluent iff for any term t such that $t \to^* u_1, t \to^* u_2$, there exists v such that $u_1 \to^* v, u_2 \to^* v$. $\to$ is canonical iff it is terminating and locally-confluent (and hence also terminating and confluent). A term t is in normal form iff there is no u such that $t \to u$.

An f-monomial $f(M)$ is in normal form with respect to R_S iff $f(M)$ cannot be rewritten using any rule in R_S.

Define a nonstrict partial ordering on f-monomials, informally capturing when an f-monomial rewrites another f-monomial, called the **Dickson** ordering: $f(M) \gg_D f(M')$ iff M' is a subset of M. Observe that the strict subpart of this ordering, while well-founded, is not total; for example, two distinct singleton multisets (constants) $\{\{a\}\} \neq \{\{b\}\}$ cannot be compared. This ordering is later used to show the termination of the completion algorithm.

A rewrite system R_S is called *reduced* iff neither the left side nor the right side of any rule in R_S can be rewritten by any of the other rules in R_S.

As in [11], the local confluence of R_S can be checked using the following constructions of superposition and critical pair.

▶ **Definition 4.** *Given two distinct rewrite rules $f(A_1) \rightarrow f(A_2)$, $f(B_1) \rightarrow f(B_2)$, let $AB = (A_1 \cup B_1) - (A_1 \cap B_1)$; $f(AB)$ is then the* superposition *of the two rules, and the* critical pair *is $(f((AB - A_1) \cup A_2), f((AB - B_1) \cup B_2))$.*

To illustrate, consider two rules $f(a, b) \rightarrow a, f(b, c) \rightarrow b$; their superposition $f(a, b, c)$ leads to the critical pair $(f(a, c), f(a, b))$.

A rule can have a constant on its left side and a nonconstant on its right side. As stated before, a singleton constant stands for the multiset containing that constant.

A critical pair is *nontrivial* iff the normal forms of its two components in $\rightarrow_{AC}$ as multisets are not the same (i.e., they are not joinable). A nontrivial critical pair generates an implied equality relating distinct normal forms of its two components.

For the above two rewrite rules, normal forms of two sides are $(f(a, c), a)$, respectively, indicating that the two rules are not locally confluent. A new derived equality is generated: $f(a, c) = a$ which is in $ACCC(\{f(a, b) = a, f(b, c) = b\})$.

It is easy to prove that if A_1, B_1 are disjoint multisets, their critical pair is trivial. Many critical pair criteria to identify additional trivial critical pairs have been investigated and proposed in [7, 17, 3].

▶ **Lemma 5.** *An AC rewrite system R_{S_f} is* locally confluent *iff the critical pair: $(f((AB - A_1) \cup A_2), f((AB - B_1) \cup B_2))$ between every pair of distinct rules $f(A_1) \rightarrow f(A_2), f(B_1) \rightarrow f(B_2)$ is joinable, where $AB = (A_1 \cup B_1) - (A_1 \cap B_1)$.*

See the Appendix for a proof.

Using the above local confluence check, a completion procedure is designed in the classical manner; equivalently, a nondeterministic algorithm can be given as a set of inference rules [2]. If a given rewrite system is not locally confluent, then new rules generated from nontrivial critical pairs (that are not joinable) are added until the resulting rewrite system is locally confluent. New rules can always be oriented since an ordering on f-monomials is assumed to be total. This completion algorithm is a special case of Gröbner basis algorithm on monomials built using a single AC symbol. The result of the completion algorithm is a locally confluent and terminating rewrite system for $ACCC(S)$.

Doing the completion algorithm on the two rules in the above examples, the derived equality is oriented into a new rule $f(a, c) \rightarrow a$. The system $\{f(a, b) \rightarrow a, f(b, c) \rightarrow b, f(a, c) \rightarrow a\}$ is indeed locally-confluent. This canonical rewrite system is a presentation of the congruence closure of $\{f(a, b) = a, f(b, c) = b\}$. Using the rewrite system, membership in its congruence closure can be decided by rewriting: $f(a, b, b) = f(a, b, c) \in ACCC(S)$ whereas $f(a, b, b) \neq f(a, a, b)$.

A simple completion algorithm is presented for the sake of completeness. It takes as input, a finite set S_C of constant equations and a finite set S_f of equations on f-monomials, and a total ordering $\gg_f$ on f-monomials extending a total ordering ordering $\gg$ on constants, and computes a reduced canonical rewrite system R_f (interchangeably written as R_S) such that $ACCC(S) = ACCC(S_{R_f})$, where S_{R_f} is the set of equations $l = r$ for every $l \rightarrow r \in R_f$.

Algorithm 1 **1AC-Completion**$(S = S_f \cup S_C, \gg_f)$.

1. Orient constant equations in S_C into terminating rewrite rules R_C using $\gg$ and interreduce them. Equivalently, using Tarjan's Union-Find data structure, for every constant $c \in C$, compute, from S_C, the equivalence class $[c]$ of constants containing c and make $R_C = \cup_{c \in C}\{c \to \hat{c} \mid c \neq \hat{c}$ and $\hat{c}$ is the least element in $[c]\}$.
   Initialize R_f to be R_C. Let $T := S_f$.
2. Pick an f-monomial equation $l = r \in T$ using some selection criterion (typically an equation of the smallest size) and remove it from T. Compute normal forms $\hat{l}, \hat{r}$ using R_f. If equal, then discard the equation, otherwise, orient into a terminating rewrite rule using $\gg_f$. Without any loss of generality, let the rule be $\hat{l} \to \hat{r}$.
3. Generate critical pairs between $\hat{l} \to \hat{r}$ and every f-rule in R_f, adding them to T.[1]
4. Add the new rule $\hat{l} \to \hat{r}$ into R_f; interreduce other rules in R_f using the new rule.
   (i) For every rule $l \to r$ in R_f whose left side l is reduced by $\hat{l} \to \hat{r}$, remove $l \to r$ from R_f and insert $l = r$ in T. If l cannot be reduced but r can be reduced, then reduce r by the new rule and generate a normal form r' of the result. Replace $l \to r$ in R_f by $l \to r'$.
5. Repeat the previous three steps until the critical pairs among all pairs of rules in R_f are joinable, and T becomes empty.
6. Output R_f as the canonical rewrite system associated with S.

▶ **Theorem 6.** *The algorithm* **1AC-Completion** *terminates, i.e., in Step 4, rules to* R_f *cannot be added infinitely often.*

See the Appendix for a proof.

▶ **Theorem 7.** *Given a finite set* S *of ground equations with a single AC symbol* f *and constants, and a total admissible ordering* $\gg_f$ *on flattened AC terms and constants, a reduced canonical rewrite system* R_f *is generated by the above completion procedure, which serves as a decision procedure for* $ACCC(S)$*.*

The proof the theorem is classical, typical of a correctness proof of a completion algorithm based on ensuring local confluence by adding new rules generated from superpositions whose critical pairs are not joinable.

▶ **Theorem 8.** *Given a total ordering* $\gg_f$ *on* f*-monomials, there is a unique reduced canonical rewrite system associated with* S_f

See the Appendix for a proof.

The complexity of this specialized decision procedure has been proved to require exponential space and double exponential upper bound on time complexity [23, 31].

The above completion algorithm generates a unique reduced canonical rewrite system R_f for the congruence closure $ACCC(S)$ because of interreduction of rules whenever a new rule is added to R_f; R_f (R_S) thus serves as its unique presentation. Using the same ordering $\gg$ on f-monomials, two sets S_1, S_2 of AC ground equations have identical (modulo presentation of multisets as AC terms) reduced rewrite systems $R_{S_1} = R_{S_2}$ iff $ACCC(S_1) = ACCC(S_2)$, thus generalizing the result for the uninterpreted case. Every f-monomial in $GT(\{f\}, C)$ has its canonical signature–its canonical form computed using R_f generated from S.

3.2 Idempotent and/or Nilpotent AC Symbols with Identity

If an AC symbol f has additional properties such as nilpotency, idempotency and/or unit, the above completion algorithm can be easily extended by expanding the local confluence check. Along with the above discussed critical pairs from a distinct pair of rules, additional critical pairs must be considered from each rule in R_f. We discuss below the case of an AC symbol being idempotent in detail; analysis for a nilpotent AC symbol, an AC symbol with identity, and various combination of properties is similar.

For any rule $f(M) \to f(N)$ where f is idempotent and M, N do not have duplicates, for every constant $a \in M$, generate a superposition $f(M \cup \{\{a\}\})$, a new critical pair $(f(N \cup \{\{a\}\}), f(M))$ and check its joinability. It can be proved that R_f is locally confluent iff the critical pairs constructed as above, from each distinct pair of rules in R_f and the new critical pair from each rule are joinable; see the Appendix for a proof. For an example, from $f(a, b) \to c$ with an idempotent f, the superpositions are $f(a, a, b)$ and $f(a, b, b)$, leading to the critical pairs: $(f(a, c), f(a, b))$ and $(f(b, c), f(a, b))$, respectively, which further reduces to $(f(a, c), c)$ and $(f(b, c), c)$, respectively. See the Appendix for a proof sketch.

For a nilpotent AC symbol f with $f(x, x) = e$, for every rule $f(M) \to f(N)$, generate a critical pair $(f(N \cup \{\{a\}\}), f((M - \{\{a\}\}) \cup \{\{e\}\}))$. If f has identity, say e, no additional critical pair is needed since from every rule $f(M) \to f(N)$, $(f(N \cup \{\{e\}\}), f(M))$ are trivially joinable. The termination proof from the previous subsection extends to each of these cases and their combination.

3.3 Computing Congruence Closure with Multiple AC symbols

The extension of the above algorithm for computing congruence closure with a single AC symbol to multiple AC symbols is straightforward.

Given a total ordering $\gg$ on constants, for each AC symbol f, define a total well-founded admissible ordering $\gg_f$ on f-monomials extending $\gg$. However, nonconstant f-monomials are not comparable with nonconstant g-monomials for $f \neq g$.

From S_C and each S_f, reduced canonical rewrite systems R_C and R_f, respectively are independently generated using the algorithm of the previous subsection. Equalities on shared constants must be propagated until no additional implied equalities are generated.

Any constant equality $c = d$ generated in an $R_f, f \in F_{AC}$, is oriented as a rewrite rule; wlog $c \to d$ is added to R_C with c reduced to d everywhere in various R_f's. If c appears in the left side of a rule in some R_g, it is removed from R_g and instead viewed as an equation, which is further reduced and the normalized equation is oriented as a rewrite rule using $\gg_g$ and checked for local confluence with other rules in R_g. This process is continued until no new implied constant equalities are generated and local confluence of each R_f is restored.

A subtle issue is when two distinct R_f and R_g, $f \neq g$, have rewrite rules with the same constant on their left sides, i.e., there is a rule $c \to f(M) \in R_f$ and $c \to g(N) \in R_g, f \neq g$. This implies that a constant c is congruent to both nonconstant f-monomial as well as g-monomial. The above can happen if in $\gg_f$ and $\gg_g$, $c \gg_f f(M)$ and $c \gg_g g(N)$, respectively. However, $f(M)$ and $g(N)$ are noncomparable in $\gg_f$ or $\gg_g$ since they have two different AC symbols. We will call this case to be that of *a shared constant having two distinct normal forms in different AC symbols*.

A new constant u is introduced with $c \gg u$, as is usually the case with new constants; $\gg_f$ and $\gg_g$ are extended to include monomials in which u appears with the constraint that $f(M) \gg_f u$ and $g(N) \gg_g u$. Add a rule $c \to u \in R_C$, replace the rule $c \to f(M) \in R_f$ by $f(M) \to u$ and $c \to g(N) \in R_g$ by $g(N) \to u$. These restrictions are easily satisfied.

The replacements of $c \to f(M)$ to $f(M) \to u$ in R_f and similarly of $c \to g(N)$ to $g(N) \to u$ in R_g may violate the local confluence of R_f and R_g. New superpositions are generated in R_f as well as R_g, possibly leading to new rules. After local confluence is restored, the result is new R'_f and R'_g. To illustrate, consider $S = \{c = a + b, c = a * b\}$ with AC $+, *$. For an ordering $c \gg b \gg a$ with both $\gg_+$ and $\gg_*$ being pure lexicographic,

$R_+ = \{c \to a+b\}, R_* = \{c \to a*b\}$. These canonical rewrite systems have a shared constant c with two different normal forms. Introduce a new constant u with $c \gg b \gg a \gg u$; make $R_S = \{a+b \to u, a*b \to u, c \to u\}$. The reader must have observed that whereas in the extended signature, the above rewrite system is unique, reduced, and canonical, on the original signature, it is not even locally confluent. A choice about whether the canonical form of c is an f-monomial or a g-monomial is not made as part a of this algorithm since nonconstant f-monomials and g-monomials are noncomparable.

A reduced canonical rewrite system $R = R_C \cup \bigcup_{f \in F_{AC}} R_f$ is generated from S to compute the AC congruence closure $ACCC(S)$ in which rules have distinct left sides.

Algorithm 2 Combination Algorithm ($S = S_C \ \cup \ \bigcup_{f \in F_{AC}} S_f, \ \ \{\gg_f | f \in f_{AC}\}$).

1. Generate a reduced canonical rewrite system R_C from S_C using the total ordering $\gg$ on C; equivalently, as in step 1 of the algorithm for the single AC symbol, Tarjan's Union-Find data structure, can be employed.
2. Normalize each S_f using R_c, resulting in equations on f-monomials on canonical constants. Wlog, we will continue to call the result S_f. This step is eagerly applied as new constant equalities on constants are generated in the steps below.
3. Run the congruence closure algorithm on each S_f from the previous subsection using the ordering $\gg_f$, generating a reduced canonical rewrite system R_f for the congruence closure $ACCC(S_f)$.
4. If any of R_f's generates an implied constant equality, say $c \to d$, include it in R_C and inter-reduce. If $c \to d$ and any other constant rewrite rule generated from R_C, reduces a rule, say $g(M) \to g(N)$ in any R_g, two cases are considered: (i) $g(M)$ is rewritten using the new rules: move $g(M) = g(N)$ from R_g to ${T_g}^2$, and check for the local confluence of the modified R_g along with T_g by generating new superposition, if any, and adding new rules in R_g. (ii) $g(M)$ is not rewritten but $g(N)$ is, then replace $g(M) \to g(N')$, where $g(N')$ is a normal form of $g(N)$ using R_g.
5. **Shared constant with canonical forms in different AC symbols:**
   If two different canonical rewrite subsystems $R_f, R_g, f \neq g$ have identical constants as the left sides, i.e. if there is a rule $c \to f(M) \in R_f$ and $c \to g(N) \in R_g, f \neq g$, introduce a new constant u, make $c \gg u$ extending $\gg_f$ on f-monomials with u making $f(M) \gg_f u$ and $g(N) \gg_g u$, and add a rule $c \to u \in R_C$, replace rules $c \to f(M) \in R_f$ by $f(M) \to u$ and $c \to g(N) \in R_g$ by $g(N) \to u$. Since u is a new symbol, orderings on f-monomials and g-monomials are extended to satisfy these requirements.
   Replacing $c \to f(M)$ by $f(M) \to u$ can result in additional superpositions with other rules in R_f and possibly new rules using u; this applies to R_g as well. After ensuring local-confluence of all new superpositions and adding new rules if needed, reduced canonical rewrite systems are generated for each R_f.
6. If no new constant equalities are generated and the set of rewrite systems R_f's do not satisfy the shared constant condition, the algorithm terminates.
7. Output the combined rewrite system consisting of a reduced canonical R_C on constants and a reduced canonical R_f for each $f \in F_{AC}$. These canonical Rewrite systems do not share a constant symbol appearing on the left side of any rule.

The termination and correctness of the algorithm follows from the termination and correctness of the algorithm for a single AC symbol and the fact there are finitely many new constant equalities that can be added.

The result of the above algorithm is a finite reduced canonical rewrite system $R_S = R_C \cup \bigcup_{f \in F_{AC}} R_{S_f}$, a disjoint union of sets of reduced canonical rewrite rules on f-monomials for each AC symbol f, along with a canonical rewrite system R_C consisting of constant rules such that the left sides of rules are distinct. R_S is unique in the extended signature assuming a family of total admissible orderings on f-monomials for every $f \in F_{AC}$, extending a total ordering on constants; this assumes a fixed choice of new constants standing for the same set of subterms during purification and flattening. In the original signature, however, R_S is neither unique nor even locally confluent (or canonical) if it includes a shared constant having multiple canonical forms in two different AC symbols as illustrated in the above example. It then becomes necessary to compare monomials in different AC symbols.

▶ **Theorem 9.** *R_S as defined above is a unique reduced canonical rewrite system in the extended signature for a given family $\{\gg_f \mid f \in f_{AC}\}$ of admissible orderings on f-monomials extending a total ordering on constants, such that $ACCC(R_S)$, with rewrite rules in R_S viewed as equations, on the original signature is $ACCC(S)$.*

The proof of the theorem follows from the fact that (i) each R_{S_f} is reduced and canonical, and is unique for S_f using $\gg_f$, (ii) the left sides of all rules are distinct, (iii) these rewrite systems are normalized using R_C.

4 Congruence Closure with Uninterpreted and Multiple AC symbols

The algorithm presented in the previous subsection to compute AC congruence closure with multiple AC symbols is combined with Kapur's congruence closure algorithm for uninterpreted symbols. The combination is straightforward given that the output of the congruence closure algorithm is a unique reduced canonical rewrite system consisting of flat rules of the form $h(a_1, \cdots, a_k) \rightarrow b$ and constant rules $a \rightarrow b$. There is no interaction between flat rules and other rules generated from AC monomials. When new constant equalities are generated, they can reduce flat rules, making the left sides of some flat rules equal, resulting in additional equalities which are handled in the same way as constant equalities generated during completion on equations on f-monomials. All other steps are the same as in the case of the congruence closure algorithm in the previous subsection for multiple AC symbols. The output of this general algorithm share the properties of the output of the congruence closure over multiple AC symbols.

It is preferable to generate R_C first and then generate R_U to check if any implied constant equalities are generated. The result is a reduced canonical rewrite system $R_C \cup R_U$ for the congruence closure of $S_C \cup S_U$ over uninterpreted symbols. R_C, R_U can be computed very fast in $O(n * log(n))$ steps, whereas computing R_f from a set of f-monomial equations is very expensive, so it always pays off to deduce constant equalities from R_C and R_U. During the computation of generating canonical rewrite systems for R_f from $S_f, f \in F_{AC}$, if a new constant equality is implied and generated, it is eagerly used to update $R_C \cup R_U$ to generate any new implied equalities, and used to update monomial equations and monomial rewrite systems constructed so far.

The algorithm from the previous subsection for computing reduced canonical rewrite systems from each S_f is applied, looking for new constant equalities generated and checking shared constant condition. As discussed in the previous subsection, in both cases, local confluence of R_f's may have to restored for checking additional superpositions, leading to possibly new rules.

The result is a modular combination, whose termination and correctness is established in terms of the termination and correctness of its various components: (i) the termination and correctness of algorithms for generating canonical rewrite systems from ground equations in a single AC symbol, for each AC symbol in F_{AC}, and their combination together with each other, and (ii) the termination and correctness of congruence closure over uninterpreted symbols and its combination with the AC congruence closure for multiple AC symbols.

Given a total ordering $\gg$ on C, let $\gg_U = \gg \cup \{h \gg_U c, h \in F_U, c \in C\}$. For each AC symbol f, define a total admissible ordering $\gg_f$ on f-monomials extending $\gg$ on C.

Algorithm 3 General Congruence Closure($S = S_C \cup S_U \cup \cup_{f \in F_{AC}} S_f, \gg_U \cup \{\gg_f \mid f \in F_{AC}\}$).

```
1. From $S_C \cup S_U$, generate a reduced canonical rewrite system $R_C \cup R_U$ representing the
   congruence closure over uninterpreted symbols such that each rule has its left side as
   $h(c_1, \cdots, c_k) \to c$ or $a \to b$ and no two left sides are the same.
2. Normalize $S_f, f \in F_{AC}$ using $R_C$.
3. Run the AC congruence closure for multiple AC symbols from the previous subsection,
   on the output from the previous step, using $\gg_f$ for each $f \in F_{AC}$.
4. If new constant equalities are generated and/or shared constant condition is satisfied,
   redo steps 1, 2, 3, restoring local confluence of $R_C, R_U$ and each $R_f$.
5. Repeat this step until no more new constant equalities are generated and until shared
   constant condition is satisfied, leading to the termination of the algorithm.
```

Since there are finitely many constants, bounded by the size of input ground equations, only finitely many constant equalities on them can be added during the propagation. As a result, the termination of the general algorithm follows from the termination of the algorithms for each of the components–S_U and S_f, for every AC symbol f. The correctness proof of the general algorithm is also structured in a modular fashion using the correctness proofs of the components S_U and S_f, $f \in F_{AC}$.

▶ **Theorem 10.** *Given a set S of ground equations in $GT(F)$, the above algorithm generates a reduced canonical rewrite system R_S on the extended signature such that $R_S = R_C \cup R_u \cup \bigcup_{f \in F_{AC}} R_f$, where each of R_C, R_U, R_f is a reduced canonical rewrite system using a set of total admissible monomial orderings $\gg_f$ on f-monomials, which extends a ordering $\gg_U$ on uninterpreted symbols and constants as defined above, and $ACCC(R_S)$, with rules in R_S viewed as equations, on the original signature is $ACCC(S)$. Further, for this given set of orderings $\gg_U$ and $\{\gg_f \mid f \in F_{AC}\}$, R_S is unique for S in the extended signature.*

It should be noted that it is not necessary to run a completion algorithm for each AC symbol separately, instead, they can be interleaved along with any new constant equalities generated to reduce constants appearing in other R_f eagerly. Even though there are new constants introduced in the generation of a reduced canonical rewrite system if two different R_f, R_g have the same constant appearing on the left sides of rules with f-monomials and g-monomials on their right side, the number of choices for canonical forms is finite given that there are only finite many AC symbols and finitely many constants.

5 Examples

The proposed algorithms are illustrated using several examples which are mostly picked from the above cited literature with the goal of not only to show how the proposed algorithms work, but also contrast them with the algorithms reported in the literature.

▶ **Example 1.** Consider an example from [5]: $S = \{f(a,c) = a,\ f(c, g(f(b,c))) = b,\ g(f(b,c)) = f(b,c)\}$ with g being uninterpreted and f being an AC symbol. A number of variations of the same example will be considered.

Mixed term $f(c, g(f(b,c)))$ is purified by introducing new symbols u_1 for $f(b,c)$ and u_2 for $g(u_1)$, gives $\{f(a,c) = a,\ f(b,c) = u_1,\ g(u_1) = u_2,\ f(c,u_2) = b, u_2 = u_1\}$. (i) $S_C = \{u_2 = u_1\}$, (ii) $S_U = \{g(u_1) = u_2\}$, and (iii) $S_f = \{f(a,c) = a, f(b,c) = u_1, f(c,u_2) = b\}$.

Different total orderings on constants are used to illustrate how different canonical forms can be generated. Consider a total ordering $f \gg g \gg a \gg b \gg u_2 \gg u_1$. $R_C = \{1.\ u_2 \rightarrow u_1\}$ normalizes the uninterpreted equation and it is oriented as: $R_U = \{2.\ g(u_1) \rightarrow u_1\}$.

To generate a reduced canonical rewrite system for AC f-terms, an admissible ordering on f-terms must be chosen. The degree-lexicographic ordering on monomials will be used for simplicity: $f(M_1) \gg_f f(M_2)$ iff $|M_1| > |M_2|$ or $|M_1| = |M_2| \wedge M_1 - M_2$ includes a constant $\gg$ every constant in $M_2 - M_1$. f-equations are normalized using rules 1 and 2, and oriented: $\{3.\ f(a,c) \rightarrow a,\ 4.\ f(c,u_1) \rightarrow b,\ 5.\ f(b,c) \rightarrow u_1\}$.

Applying the AC congruence closure completion algorithm, the superposition between the rules $3, 5$ is $f(a,b,c)$ with the critical pair: $(f(a,b), f(a,u_1))$, leading to a rewrite rule 6. $f(a,b) \rightarrow f(a,u_1)$; the superposition between the rules $3, 4$ is $f(a,c,u_1)$ with the critical pair: $(f(a,u_1), f(a,b))$ which is trivial by rule 6. The superposition between the rules $4, 5$ is $f(b,c,u_1)$ with the critical pair: $(f(b,b), f(u_1,u_1))$ giving : 7. $f(b,b) \rightarrow f(u_1,u_1)$. The rewrite system $R_f = \{3,4,5,6,7\}$ is a reduced canonical rewrite system for S_f. $R_S = \{1,2\} \cup R_f$ is a reduced canonical rewrite system associated with the AC congruence closure of the input and serves as its decision procedure.

In the original signature, the rewrite system R_S is: $\{g(f(b,c))) \rightarrow f(b,c),\ f(a,c) \rightarrow a,\ f(b,c,c) \rightarrow b,\ f(a,b) \rightarrow f(a,b,c),\ f(b,b) \rightarrow f(b,b,c,c)\}$ with 5 becoming trivial. The reader would observe this rewrite system is locally confluent but not terminating.

Consider a different ordering: $f \gg g \gg a \gg b \gg u_1 \gg u_2$. This gives rise to a related rewrite system in which u_1 is replaced by u_2: $\{u_1 \rightarrow u_2, g(u_2) \rightarrow u_2, f(a,c) \rightarrow a, f(c,u_2) \rightarrow b, f(b,c) \rightarrow u_2, f(a,b) \rightarrow f(a,u_2), f(b,b) \rightarrow f(u_2,u_2)\}$. Compare it in the original signature with the above system: $R_S = \{f(b,c) \rightarrow g(f(b,c)), g(g(f(b,c))) \rightarrow g(f(b,c)), f(a,c) \rightarrow a, f(c, g(f(b,c))) \rightarrow b, f(a,b) \rightarrow f(a, g(f(b,c))), f(b,b) \rightarrow f(g(f(b,c)), g(f(b,c)))\}$.

▶ **Example 2.** This example illustrates interaction between congruence closures over uninterpreted symbols and AC symbols. Let $S = \{g(b) = a, g(d) = c, a * c = c, b * c = b, a * b = d\}$ where g is uninterpreted and $*$ is AC. Let $* \gg g \gg a \gg b \gg c \gg d$ with $g, *$. Applying the steps of the algorithm, R_U, the congruence closure over uninterpreted symbols, is $\{g(b) \rightarrow a, g(d) \rightarrow c\}$, Completion on the $*$-equations using degree lexicographic ordering, oriented as $\{a * c \rightarrow c, b * c \rightarrow b, a * b \rightarrow d\}$, generates an implied constant equality $b = d$ from the critical pair of $a * c \rightarrow c, b * c \rightarrow b$. Using the rewrite rule $b \rightarrow d$, the AC rewrite system reduces to: $R_* = \{a * c \rightarrow c, c * d \rightarrow d, a * d \rightarrow d\}$, which is canonical.

The implied constant equality $b = d$ is added to $R_C : \{b \rightarrow d\}$ and is also propagated to R_U, which makes the left sides of $g(b) \rightarrow a$ and $g(d) \rightarrow c$ equal, generating another implied equality $a = c$. This equality is oriented and added to R_C: $\{a \rightarrow c, b \rightarrow d\}$, and R_U becomes $\{g(d) \rightarrow c\}$. This implied equality is propagated to the AC rewrite system on $*$. R_C normalizes R_* to $\{c * c \rightarrow c, c * d \rightarrow d, a * d \rightarrow d\}$.

The output of the algorithm is a canonical rewrite system: $\{g(d) \rightarrow c, b \rightarrow d, a \rightarrow c, c * c \rightarrow c, c * d \rightarrow d\}$. In general, propagation of equalities can result in the left sides of the rules in AC subsystems change, generating new superpositions, much like in the uninterpreted case.

▶ **Example 3.** Consider another example with multiple AC symbols from [24]: $S = \{a + b = a * b, a * c = g(e), e = e'\}$ with $+, *$ being AC and g as uninterpreted, to illustrate flexibility in choosing orderings in our framework.

Purification leads to introduction of new constants : $\{1.\ a+b = u_0,\ 2.\ a*b = u_1,\ 3.\ a*c = u_2, 4.\ g(e) = u_2, 5.\ e = e', 6.\ u_0 = u_1\}$. Depending upon a total ordering on constants a, b, c, e, e', there are thus many possibilities depending upon the desired canonical forms.

Recall that $a + b$ cannot be compared with $a * b$, however u_0 and u_1 can be compared. If canonical forms are desired so that the canonical form of $a + b$ is $a * b$, the ordering should include $u_0 \gg u_1$. Consider an ordering $a \gg b \gg c \gg e \gg e' \gg u_2 \gg u_0 \gg u_1$. Degree-lexicographic ordering is used on $+$ and $*$ monomials. This gives rise to: $R_C = \{6.\ u_0 \to u_1, 5.\ e \to e'\}$, $R_U = \{4'.\ g(e') \to u_2\}$ along with $R_+ = \{1.\ a + b \to u_1\}$ and $R_* = \{2.\ a * b \to u_1, 3.\ a * c \to u_2\}$. The canonical rewrite system for $*$ is: $\{2.\ a * b \to u_1, 3.\ a * c \to u_2, 7.\ b * u_2 \to c * u_1\}$.

The rewrite system $R_S = \{1.\ a + b \to u_1,\ 2.\ a * b \to u_1,\ 2.\ u_0 \to u_1,\ 3.\ a * c \to u_2,\ 4'.\ g(e') \to u_2,\ 5.\ e \to e',\ 6.\ b * u_2 \to c * u_1\}$ is canonical. Both $a + b$ and $a * b$ have the same normal form u_1 standing for $a * b$ in the original signature.

With $u_1 \gg u_0$, another canonical rewrite system $R_S = \{1.\ a + b \to u_0,\ 2.\ a * b \to u_0,\ 6'.\ u_1 \to u_0,\ 3.\ a * c \to u_2,\ 4.\ g(e') \to u_2,\ 5.\ e \to e',\ 6.\ b * u_2 \to c * u_0\}$ in which $a + b$ and $a * b$ have the normal form u_0 standing for $a + b$, different from the one above.

6 A Gröbner Basis Algorithm as an AC Congruence Closure

Buchberger's Gröbner basis algorithm when extended to polynomial ideals over integers [10] can be interestingly viewed as a special congruence closure algorithm with multiple AC symbols $+$ and $*$ which in addition, satisfy the properties of a commutative ring with unit. A manuscript proposing this new perspective on Gröbner basis algorithms with interesting implications is under preparation [15]. Below, we illustrate this new insight using an example from [10]. Relationship between Gröbner basis algorithm and the Knuth-Bendix completion procedure has been investigated in [9, 30, 12, 22], but the proposed insight is novel.

The ring structure of polynomials gives rise to additional interaction when a canonical rewrite system for the congruence closure of $+$-monomials is combined with a canonical rewrite system for the congruence closure of $*$-monomials. Along with the identities for $+$, $x + 0 = x$, and for $*$, $x * 1 = x$, and the distributivity axiom, $+$ also has an inverse operation: $x + -(x) = 0, -0 = 0, -(-x) = x, -(x + y) = -x + -y$. Abusing the notation, $x + -(y)$ will be written as $x - y$. In addition, $x * 0 = 0$. With the distributivity rule: $x * (y + z) = (x * y) + (x * z)$, these axioms when oriented from left to right constitute a canonical rewrite system for a commutative ring with unit and are used for normalizing terms to polynomials.

An additive monomial $c\ t$, where $c \neq 0$, is an abbreviation of repeating $*$-monomial t c times; if c is positive, say 3, then $3\ t$ is an abbreviation for $t + t + t$; similarly if c is negative, say -2, then $-2\ t$ is an abbreviation for $-t - t$. A $*$-monomial with unit coefficient is a pure term expressed in $*$, but a monomial $3\ y * y * y$, for example,is a mixed term $y * y * y + y * y * y + y * y * y$ with $+$ as the outermost symbol which has $*$ subterms.

Consider an example [10]; a related example is also discussed in [22], so an interested reader is invited to contrast the proposed approach with the one there. The input basis is: $7\ x * x * y = 3\ x, 4\ x * y * y = x * y, 3\ y * y * y = 0$ of a polynomial ideal over the integers [10].

Purification of the above equations leads to: $1.\ 7\ u_1 = 3\ x, 2.\ 4\ u_2 = u_3, 3.\ 3\ u_4 = 0$ with $4.\ x * x * y = u_1, 5.\ x * y * y = u_2, 6.\ x * y = u_3. 7.\ y * y * y = u_4$.

Let a total ordering on all constants be: $u_1 \gg u_2 \gg u_4 \gg u_3 \gg x \gg y$. Extend it using the degree-lexicographic ordering on $+$-monomials as well as $*$-monomials, Orienting $*$ equations: $R_* = \{4.\ x*x*y \rightarrow u_1, 5.\ x*y*y \rightarrow u_2, 6.\ x*y \rightarrow u_3, 7.\ y*y*y \rightarrow u_4\}$. Orienting $+$ equations, $R_+ = \{1.\ 7\ u_1 \rightarrow 3\ x, 2.\ 4\ u_2 \rightarrow u_3, 3.\ 3\ u_4 \rightarrow 0\}$.

Completion on the ground equations on $*$ terms generates a reduced canonical rewrite system: $R_* = \{6.\ x*y \rightarrow u_3, 7.\ y*y*y \rightarrow u_4, 8.\ u_3*y \rightarrow u_2, 9.\ u_3*x \rightarrow u_1, 10.\ u_2*x \rightarrow u_3*u_3, 11.\ u_1*y \rightarrow u_3*u_3, 12.\ u_1*u_4 \rightarrow u_2*u_2, 13.\ u_4*x*x \rightarrow u_2*u_3, 14.\ u_3*u_3*u_3 \rightarrow u_1*u_2\}$. This system captures relationships among all product monomials appearing in the input. The canonical rewrite system for $+$ is: $R_+ = \{1.\ 7\ u_1 \rightarrow 3\ x, 2.\ 4\ u_2 \rightarrow u_3, 3.\ 3\ u_4 \rightarrow 0\}$.

Rules in R_+ and R_* interact, leading to new superpositions and critical pairs: for $c\ u \rightarrow r \in R_+, u*m \rightarrow r' \in R_*$, where $c \in \mathbb{Z} - \{0\}$, u is a constant and m is $*$-monomial, the superposition is $(c\ u)*m$ generating the critical pair $(r*m, c\ r')$, which is normalized using distributivity and other rules. It can be shown that only this superposition needs to be considered to check local confluence of $R_+ \cup R_*$[15].

As an example, rules $3:\ 3\ u_4 \rightarrow 0$ and $12:\ u_1*u_4 \rightarrow u_2*u_2$ give the superposition $u_1*u_4+u_1*u_4+u_1*u_4$, which is $3u_1*u_4$, generating the critical pair $(3\ u_2*u_2, 0)$; rules 3 and 13 gives a trivial critical pair. Considering all such superpositions, the resulting canonical rewrite system can be shown to include $\{3\ x \rightarrow 0, u_1 \rightarrow 0, u_2 \rightarrow u_3, 3\ u_4 \rightarrow 0\}$ among other rules. When converted into the original signature, this is precisely the Gröbner basis reported as generated using Kandri-Rody and Kapur's algorithm: $\{3\ x \rightarrow 0, x*x*y \rightarrow 0, x*y*y \rightarrow x*y, 3\ y*y*y \rightarrow 0\}$ as reported in [10].

The above illustrates the power and elegance of the proposed combination framework. Comparing with [22], it is reported there that a completion procedure using normalized rewriting generated 90 critical pairs in contrast to AC completion procedure [27] computed 1990 critical pairs; in the proposed approach, much fewer critical pairs are generated in contrast, without needing to use any AC unification algorithm or extension rules.

The proposed approach can also be used to compute Gröbner basis of polynomial ideals with coefficients over finite fields such as $\mathbb{Z}_p$ for a prime number p, as well as domain with zero divisor such as $\mathbb{Z}_4$ [16]. For example, in case of $\mathbb{Z}_5$, another rule is added: $1+1+1+1+1 \rightarrow 0$ added to the input basis.

7 Conclusion

A modular algorithm for computing the congruence closure of ground equations expressed using AC function symbols and uninterpreted symbols is presented. The algorithm is derived by generalizing the framework first presented in [13] for generating the congruence closure of ground equations over uninterpreted symbols. The key insight from [13]–flattening of subterms by introducing new constants and congruence closure on constants, is generalized by flattening mixed AC terms and purifying them by introducing new constants to stand for pure AC terms in every (single) AC symbol. The result of this transformation on a set of equations on mixed ground terms is a set of constant equations, a set of flat equations relating a nonconstant term in an uninterpreted symbol to a constant, and a set of equations on pure AC terms in a single AC symbol. Such decomposition and factoring enable using congruence closure algorithms for each of the subproblems independently, which propagate equalities on shared constants. Once the propagation of constant equalities stabilizes (reaches a fixed point), the result is (i) unique reduced canonical rewrite systems for each subproblem and finally, (ii) a unique reduced canonical rewrite system for the congruence closure of a finite set of ground equations over multiple AC symbols and uninterpreted symbols. The algorithms extend easily when AC symbols have additional properties such as idempotency, identity and nilpotency.

The modularity of the algorithms leads to easier and simpler correctness and termination proofs in contrast to those in [5, 22]. The complexity of the procedure is governed by the complexity of generating a canonical rewrite system for AC ground equations on constants.

The proposed algorithm is a direct generalization of Kapur's algorithm for the uninterpreted case, which has been shown to be efficiently integrated into SMT solvers including BarcelogicTools [26]. We believe that the AC congruence closure can also be effectively integrated into SMT solvers. Unlike other proposals, the proposed algorithm neither uses specialized AC compatible orderings on nonground terms nor extension rules often needed in AC/E completion algorithms and AC/E-unification, thus avoiding explosion of possible critical pairs for consideration.

A by-product of this new algorithm based on the proposed framework is a new way to view a Gröbner basis algorithm for polynomial ideals over integers, as a congruence closure algorithm over a commutative ring with unit, which is a congruence closure algorithm with two AC symbols $+$ and $*$, extended to consider the additional properties of $+$ and $*$.

References

1 Franz Baader and Deepak Kapur. Deciding the word problem for ground identities with commutative and extensional symbols. In Nicolas Peltier and Viorica Sofronie-Stokkermans, editors, *Automated Reasoning – 10th International Joint Conference, IJCAR 2020, Paris, France, July 1-4, 2020, Proceedings, Part I*, volume 12166 of *Lecture Notes in Computer Science*, pages 163–180. Springer, 2020. doi:10.1007/978-3-030-51074-9_10.

2 Franz Baader and Tobias Nipkow. *Term rewriting and all that.* Cambridge University Press, 1998.

3 Leo Bachmair and Nachum Dershowitz. Critical pair criteria for completion. *J. Symb. Comput.*, 6(1):1–18, 1988. doi:10.1016/S0747-7171(88)80018-X.

4 Leo Bachmair, I. V. Ramakrishnan, Ashish Tiwari, and Laurent Vigneron. Congruence closure modulo associativity and commutativity. In Hélène Kirchner and Christophe Ringeissen, editors, *Frontiers of Combining Systems, Third International Workshop, FroCoS 2000, Nancy, France, March 22-24, 2000, Proceedings*, volume 1794 of *Lecture Notes in Computer Science*, pages 245–259. Springer, 2000. doi:10.1007/10720084_16.

5 Leo Bachmair, Ashish Tiwari, and Laurent Vigneron. Abstract congruence closure. *J. Autom. Reason.*, 31(2):129–168, 2003. doi:10.1023/B:JARS.0000009518.26415.49.

6 A. Michael Ballantyne and Dallas Lankford. New decision algorithms for finitely presented commutative semigroups. *Computers and Mathematics Applications*, 7:159–165, 1981.

7 Thomas Becker, Volker Weispfenning, and Heinz Kredel. *Gröbner bases – a computational approach to commutative algebra*, volume 141 of *Graduate texts in mathematics*. Springer, 1993.

8 Peter J. Downey, Ravi Sethi, and Robert Endre Tarjan. Variations on the common subexpression problem. *J. ACM*, 27(4):758–771, 1980. doi:10.1145/322217.322228.

9 Abdelilah Kandri-Rody and Deepak Kapur. On relationship between buchberger's gröbner basis algorithm and the knuth-bendix completion procedure. Technical report no. ge-83crd286, General Electric Corporate Research and Development, Schenectady, NY, November 1983.

10 Abdelilah Kandri-Rody and Deepak Kapur. Computing a gröbner basis of a polynomial ideal over a euclidean domain. *J. Symb. Comput.*, 6(1):37–57, 1988. doi:10.1016/S0747-7171(88)80020-8.

11 Abdelilah Kandri-Rody, Deepak Kapur, and Paliath Narendran. An ideal-theoretic approach to work problems and unification problems over finitely presented commutative algebras. In Jean-Pierre Jouannaud, editor, *Rewriting Techniques and Applications, First International Conference, RTA-85, Dijon, France, May 20-22, 1985, Proceedings*, volume 202 of *Lecture Notes in Computer Science*, pages 345–364. Springer, 1985. doi:10.1007/3-540-15976-2_17.

12 Abdelilah Kandri-Rody, Deepak Kapur, and Franz Winkler. Knuth-bendix procedure and buchberger algorithm: A synthesis. In Gaston H. Gonnet, editor, *Proceedings of the ACM-SIGSAM 1989 International Symposium on Symbolic and Algebraic Computation, ISSAC '89, Portland, Oregon, USA, July 17-19, 1989*, pages 55–67. ACM, 1989. `doi:10.1145/74540.74548`.

13 Deepak Kapur. Shostak's congruence closure as completion. In Hubert Comon, editor, *Rewriting Techniques and Applications, 8th International Conference, RTA-97, Sitges, Spain, June 2-5, 1997, Proceedings*, volume 1232 of *Lecture Notes in Computer Science*, pages 23–37. Springer, 1997. `doi:10.1007/3-540-62950-5_59`.

14 Deepak Kapur. Conditional congruence closure over uninterpreted and interpreted symbols. *J. Syst. Sci. Complex.*, 32(1):317–355, 2019. `doi:10.1007/s11424-019-8377-8`.

15 Deepak Kapur. Weird gröbner bases: An application of associative-commutative congruence closure algorithm. Tech report under preparation, Department of Computer Science, University of New Mexico, May 2021.

16 Deepak Kapur and Yongyang Cai. An algorithm for computing a gröbner basis of a polynomial ideal over a ring with zero divisors. *Math. Comput. Sci.*, 2(4):601–634, 2009. `doi:10.1007/s11786-009-0072-z`.

17 Deepak Kapur, David R. Musser, and Paliath Narendran. Only prime superpositions need be considered in the knuth-bendix completion procedure. *J. Symb. Comput.*, 6(1):19–36, 1988. `doi:10.1016/S0747-7171(88)80019-1`.

18 Deepak Kapur and Paliath Narendran. Double-exponential complexity of computing a complete set of ac-unifiers. In *Proceedings of the Seventh Annual Symposium on Logic in Computer Science (LICS '92), Santa Cruz, California, USA, June 22-25, 1992*, pages 11–21. IEEE Computer Society, 1992. `doi:10.1109/LICS.1992.185515`.

19 Deepak Kapur and Hantao Zhang. An overview of rewrite rule laboratory (rrl). *Computers and Mathematics Applications*, 29:91–114, 1995.

20 Donald Knuth and Peter Bendix. Simple word problems in universal algebras. In Leech, editor, *Computational Problems in Abstract Algebra*, pages 263–297. Pergamon Press, 1970.

21 Claude Marché. On ground ac-completion. In Ronald V. Book, editor, *Rewriting Techniques and Applications, 4th International Conference, RTA-91, Como, Italy, April 10-12, 1991, Proceedings*, volume 488 of *Lecture Notes in Computer Science*, pages 411–422. Springer, 1991. `doi:10.1007/3-540-53904-2_114`.

22 Claude Marché. Normalized rewriting: An alternative to rewriting modulo a set of equations. *J. Symb. Comput.*, 21(3):253–288, 1996. `doi:10.1006/jsco.1996.0011`.

23 Ernst W. Mayr and Albert Meyer. The complexity of the word problems for commutative semigroups and polynomial ideals. *Advances in Mathematics*, 46:305–439, 1982.

24 Paliath Narendran and Michaël Rusinowitch. Any gound associative-commutative theory has a finite canonical system. In Ronald V. Book, editor, *Rewriting Techniques and Applications, 4th International Conference, RTA-91, Como, Italy, April 10-12, 1991, Proceedings*, volume 488 of *Lecture Notes in Computer Science*, pages 423–434. Springer, 1991. `doi:10.1007/3-540-53904-2_115`.

25 Greg Nelson and Derek C. Oppen. Fast decision procedures based on congruence closure. *J. ACM*, 27(2):356–364, 1980. `doi:10.1145/322186.322198`.

26 Robert Nieuwenhuis and Albert Oliveras. Fast congruence closure and extensions. *Inf. Comput.*, 205(4):557–580, 2007. `doi:10.1016/j.ic.2006.08.009`.

27 Gerald E. Peterson and Mark E. Stickel. Complete sets of reductions for some equational theories. *J. ACM*, 28(2):233–264, 1981. `doi:10.1145/322248.322251`.

28 Robert E. Shostak. An algorithm for reasoning about equality. *Commun. ACM*, 21(7):583–585, 1978. `doi:10.1145/359545.359570`.

29 Franz Winkler. *Canonical forms in the finitely presented algebras*. Ph.d. dissertation, U. of Paris 11, 1983.

30 Franz Winkler. *The Church-Rosser Property in Computer Algebra and Special Theorem Proving: An Investigation of Critical-Pair/Completion Algorithms.* Ph.d. dissertation, University of Linz, 1984.

31 Chee-Keng Yap. A new lower bound construction for commutative thue systems with aapplications. *J. Symb. Comput.*, 12(1):1–28, 1991. doi:10.1016/S0747-7171(08)80138-1.

32 Hantao Zhang. Implementing contextual rewriting. In Michaël Rusinowitch and Jean-Luc Rémy, editors, *Conditional Term Rewriting Systems, Third International Workshop, CTRS-92, Pont-à-Mousson, France, July 8-10, 1992, Proceedings*, volume 656 of *Lecture Notes in Computer Science*, pages 363–377. Springer, 1992. doi:10.1007/3-540-56393-8_28.

A Appendix: Proofs

▶ **Lemma 5.** *An AC rewrite system R_S is* locally confluent *iff the critical pair: $(f((AB - A_1) \cup A_2), f((AB - B_1) \cup B_2))$ between every pair of distinct rules $f(A_1) \to f(A_2), f(B_1) \to f(B_2)$ is joinable, where $AB = (A_1 \cup B_1) - (A_1 \cap B_1)$.*

Proof. Consider a flat term $f(C)$ rewritten in two different ways in one step using not necessarily distinct rules: $f(A_1) \to f(A_2), f(B_1) \to f(B_2)$. The result of the rewrites is: $(f((C - A_1) \cup A_2), f((C - B_1) \cup B_2))$. Since $A_1 \subseteq C$ as well as $B_1 \subseteq C$, $AB \subseteq C$; let $D = C - AB$. The critical pair is then $(f(D \cup ((AB - A_1) \cup A_2)) f(D \cup ((AB - B_1) \cup B_2)))$, all rules applicable to the critical pair to show its joinability, also apply, thus showing the joinability of the pair. The other direction is straightforward. The case of when at least one of the rules has a constant on its left side is trivially handled. ◀

▶ **Theorem 6.** *The algorithm* **1AC-Completion** *terminates, i.e., in Step 4, rules to R_f cannot be added infinitely often.*

Proof. Proof by Contradiction. A new rule $\hat{l} \to \hat{r}$ in Step 4 of the algorithm is added to R_f only if no other rule can reduce it, i.e, for every rule $l \to r \in R_f$, $\hat{l}$ and l are noncomparable in $\gg_D$. For R_f to be infinite, implying the nontermination of the algorithm means that R_f must include infinitely many noncomparable left sides in $\gg_D$, a contradiction to Dickson's Lemma. ◀

▶ **Theorem 8.** *Given a total ordering $\gg_f$ on f-monomials, there is a unique reduced canonical rewrite system associated with S_f.*

Proof. Proof by Contradiction. Suppose there are two distinct reduced canonical rewrite systems R_1 and R_2 associated with S_f for the same $\gg_f$. Pick the least rule $l \to r$ in $\gg_f$ on which R_1 and R_2 differ; wlog, let $l \to r \in R_1$. Given that R_2 is a canonical rewrite system for S_f and $l = R \in ACCC(S_f)$, l and r must reduce using R_2 implying that there is a rule $l' \to r' \in R_2$ such that $l \gg_D l'$; since R_1 has all the rules of R_2 smaller than $l \to r$, $l \to r$ can be reduced in R_1, contradicting the assumption that R_1 is reduced. If $l \to r' \in R_2$ where $r' \neq r$ but $r' \gg_f r$, then r' is not reduced implying that R_2 is not reduced. ◀

▶ **Lemma 5Idem.** *An AC rewrite system R_S with $f(x,x) = x$, is* locally confluent *iff (i) the critical pair: $(f((AB - A_1) \cup A_2), f((AB - B_1) \cup B_2))$ between every pair of distinct rules $f(A_1) \to f(A_2), f(B_1) \to f(B_2)$ is joinable, where $AB = (A_1 \cup B_1) - (A_1 \cap B_1)$, and (ii) for every rule $f(M) \to f(N) \in R_S$ and for every constant $a \in M$, the critical pair, $(f(M), f(N \cup \{\{a\}\})$, is joinable.*

Proof. Consider a flat term $f(C)$ rewritten in two different ways in one step using not necessarily distinct rules and/or $f(x,x) \to x$. There are three cases: (i) $f(C)$ is rewritten in two different ways in one step using $f(x,x) \to x$ to $f(C - \{\{a\}\})$ and $f(C - \{\{b\}\})$. After single step rewrites, the idempotent rule can be applied again on both sides giving $f(C - \{\{a,b\}\})$.

(ii) $f(C)$ is rewritten in two different ways, with one step using $f(x, x) \to x$ and another using $f(M) \to f(N)$. An application of the idempotent rule implies that C includes a constant a, say, at least twice; the result of one step rewriting is: $(f(C - \{\{a\}\}), f((C - M) \cup N)$. This implies there exists a multiset A such that $C = A \cup M \cup \{ a\}\}$. The critical pair generated from $f(M) \to f(N)$ is $(f(M), f(N \cup \{\{a\}\})$. The rewrite steps used to show the joinability of $(f(M), f(N \cup \{\{a\}\})$ apply also on $(f(C - \{\{a\}\}), f((C - M) \cup N)$, showing joinability. The third case is the same as that of Lemma 5 and is omitted. ◀

Similar local confluence lemmas can be proved in case f is nilpotent, has unit and various combinations.