

# MODULARITY AND COMBINATION OF ASSOCIATIVE COMMUTATIVE CONGRUENCE CLOSURE ALGORITHMS ENRICHED WITH SEMANTIC PROPERTIES

DEEPAK KAPUR

Department of Computer Science, University of New Mexico, Albuquerque, NM, USA  
*e-mail address:* kapur@unm.edu

**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 a congruence closure as a canonical ground rewrite system in which nonflat terms are abstracted by new 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, identities, cancelativity and group properties as well as their various combinations. 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 the Satisfiability modulo Theories (SMT) solvers. It is shown that a Gröbner basis algorithm for polynomial ideals with integer coefficients can be formulated as a combination algorithm for the congruence closure over an Abelian group with  $+$  and the congruence closure over the AC symbol  $*$  with the identity 1, linked using the distributivity property of  $*$  over  $+$ .

## 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 [DST80, Sho78, NO80]. Whereas congruence closure algorithms were implemented in earlier verification systems [Sho78, NO80, KZ95, Zha92], their role has become particularly critical in Satisfiability modulo Theories (SMT) solvers as a glue to combine different decision procedures for various

---

*Key words and phrases:* Congruence Closure, Canonical Forms, Canonical Rewrite Systems, Associative-Commutative, Idempotency, Nilpotency, Cancellation, Group, Gröbner basis.

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

theories. More recently, many new uses of the congruence closure are being explored in the **egg** project at the University of Washington [WNW<sup>+</sup>21].

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 as well as uninterpreted symbols. Canonical forms (unique normal forms) can be associated with congruence classes. Further, a unique reduced ground rewrite system serves as a presentation of the congruence closure of a finite set of ground equations, which allows checking whether two different finite sets of ground equations define the same congruence closure or if 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 [Kap97, Kap19]: introduction of new constants for nested subterms, resulting in flat and constant equations, extended to purification of mixed subterms with many AC symbols; AC ground terms are flattened and new constants for pure AC terms in each AC symbol are introduced, 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 commutativity, idempotency, nilpotency, existence of

identities, cancelation and group properties, without restrictions on orderings on mixed terms. Of particular interest is a new algorithm for generating congruence closure of ground equations with a cancelative AC symbol; this is discussed in detail as a separate topic because of new kinds of critical pairs, called **cancelative disjoint superposition**, needed to generate a canonical rewrite system. Termination and correctness proofs of these congruence closure algorithms are modular and simple in contrast to complex arguments and proofs in [BTV03, NR91]. 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 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 [Kap97]. 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 [KKN85] for computing the congruence closure and the associated canonical rewrite system from ground equations with a single AC symbol and constants. In section 4, it is shown how additional properties of AC symbols such as idempotency, nilpotency, identity and their combination can be integrated into the algorithm. Section 5 presents a new algorithm for generating a congruence closure algorithm for ground equations with a cancelative AC symbol; new kinds of superpositions and critical pairs are needed to generate a canonical rewrite system since unlike most congruence closure algorithms, it becomes necessary to explore terms larger than appearing in the input and/or a query to get congruences on terms using the cancelation inference rule. That section also presents a congruence closure algorithm for ground equations with an AC symbol satisfying group properties; it is shown to be related to Gaussian elimination. Section 6 discusses how the propagation of new constant equalities deduced from various algorithms update various kinds of ground canonical rewrite system. In Section 7, an algorithm for computing congruence closure of AC ground equations with multiple AC symbols and constants is presented. Section 8 generalizes to the case of combination of AC symbols and uninterpreted symbols. Section 9 shows how lexicographic orderings in which a constant can be bigger than a nonconstant ground term can be supported in the proposed framework. Section 10 discusses a variety of examples illustrating the proposed algorithms. Section 11 illustrates the power and elegance of the proposed framework by demonstrating how a congruence closure algorithm for two AC symbols with some additional properties, particularly distributivity, can be further generalized to formulate a Gröbner basis algorithm on polynomial ideals over the integers. Section 12 concludes with some ideas for further investigation.

**1.1. Related Work.** Congruence closure algorithms have been developed and analyzed for over four decades [DST80, Sho78, NO80]. The algorithms presented here use the framework first informally proposed in [Kap97] for congruence closure in which the author separated the algorithm into three 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) updating of nonconstant rules as constant equivalence closure evolves and propagates new constant equalities. 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 [Kap19] and semantic congruence closure [BK20]. Further, it enables the generation of a unique reduced ground 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 [KKN85].

Congruence closure algorithms on ground equations with interpreted symbols can be viewed as special cases of the Knuth-Bendix completion procedure [KB70] 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 [PS81] 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 the case for its extensions for other ground formulas.

In [BL81], a generalization of the Knuth-Bendix completion procedure [KB70] to handle AC symbols [PS81] 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 [LeC83].

In [NR91], the authors used the completion algorithm discussed in [KKN85] 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 [NR91] 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; any total ordering on original constants and new constants is extended to one of many possible orderings on pure terms with a single AC symbol; this ordering on AC terms suffices to compute a canonical ground AC rewrite system. Different orderings on AC terms with different AC symbols can be used; for example, 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 [Mar91], 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 [NR91]. Neither in [NR91] nor in [Mar91], any explicit mention is made of uninterpreted symbols appearing in ground equations.

Similar to [BL81], several approaches based on adapting Peterson and Stickel's generalization of the Knuth-Bendix completion procedure to consider special ground theories have been reported [LeC83, Mar91]. In [BRTV00, BTV03], the authors adapted Kapur's congruence closure [Kap97] using its key ideas to an abstract inference system. Various congruence closure algorithms, including Sethi, Downey and Tarjan [DST80], Nelson and Oppen [NO80] and Shostak [Sho78], 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 Peterson and Stickel's generalization [PS81] of the Knuth-Bendix completion procedure for AC symbols. All of these approaches based on Peterson and Stickel's generalization used extension rules introduced in [PS81] 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 [KN92].

An approach based on normalized rewriting was proposed in [Mar96] 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 [LeC83] for finitely presented algebraic structures with Peterson and Stickel's AC completion, addressing weaknesses in E-completion and constrained rewriting; AC compatible termination orderings are difficult to design in the presence of AC symbols with identity, idempotency and nilpotency properties. Such 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 [KKN85] 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 is imposed for the uninterpreted part; independent separate orderings on +-monomials for each AC symbol + that extend a common total ordering on constants can be used, thus giving considerable flexibility in choosing termination orderings overall. In most related approaches except for [NR91], 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.

Since the proposed research addresses combination of congruence closure algorithms, a brief comparison with the extensive literature on the modular combination of decision procedures for theories with disjoint signatures but sharing constants, as well as combining matching and unification algorithms must be made. Combination problems are a lot easier to consider for ground theories. The reader would notice that much like the seminal work of Nelson and Oppen where they showed that for combining satisfiability procedures of theories under certain conditions, it suffices to share equalities on shared constants and variables, the proposed combination framework only shares constant equalities even though there are no additional restrictions such as that ground equational theories in this paper satisfy the stable finiteness condition (see [BT97] for a detailed discussion of implications of Nelson and Oppen's requirements for equational theories). More general combination results for decidable theories can be found in [ABRS09] in which a variable inactivity condition (disallowing literals of the form  $x = t$  in clauses thus prohibiting paramodulation on a variable), a kind of "pseudo" collapse-free equation, are imposed; such combination frameworks are likely to need more sophisticated machinery to consider AC symbols.

## 2. PRELIMINARIES

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.,  $F = F_{AC} \cup F_U \cup C$ . Let  $GT(F)$  be the ground terms constructed from  $F$ ; sometimes, we will write it as  $GT(F, C)$  to highlight the constants  $C$  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 2.1.** Given a finite set  $S = \{a_i = b_i \mid 1 \leq i \leq m\}$  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  $f(a_1, \dots, a_k) = f(b_1, \dots, b_k) \in CC(S)$ . 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  $S$  [BN98, BK20].

**2.2. A Congruence Closure Algorithm for Uninterpreted Symbols.** The algorithm in [Kap97, Kap19] for computing congruence closure of a finite set  $S$  of ground equations over uninterpreted function symbols and constants 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  $h(c_1, \dots, c_k)$  equal to constants (also called flat equations).<sup>1</sup> 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 [BK20]. A disequation on ground terms is converted to a disequation on constants by introducing new symbols for the ground terms.

As given in Section 3 of [BK20], assuming a total ordering on constants, the algorithm (i) computes a canonical rewrite system from constant equations, picking a canonical representative, which is the least constant among all constants in each congruence class, and (ii) replaces constants in the left side of each flat rule by their canonical representatives, and identifies flat rules with identical left sides, replacing them by a single rule with the least constant on its right side, thus possibly generating new constant equalities. These steps are repeated if at least one new constant equality is generated in (ii).

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 [Kap97, Kap19, BK20] is a reduced canonical rewrite system  $R_S$  associated with  $CC(S)$  (as well as  $S$ ) that includes *function* rules, also called *flat* rules, of the form  $h(c_1, \dots, c_k) \rightarrow d$  and *constant* rules  $c \rightarrow d$  such that no two left sides of the rules are identical; further, all constants are in canonical forms. The following result is proved in [Kap97] (see also [NO07]).

**Theorem 2.2** [Kap97]. *Given a set  $S$  of ground equations, a reduced canonical rewrite system  $R_S$  on the extended signature, consisting of nonconstant flat rules  $h(c_1, \dots, c_k) \rightarrow d$ , 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 [DST80], 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  $\hat{g}_1 = \hat{g}_2$ .

The output of the above algorithm can also be viewed as consisting of disjoint rewrite systems corresponding to (i) a rewrite system on constants representing Union-Find algorithm in which the rewrite rules correspond to the edges in a forest of trees such that if there

<sup>1</sup>This representation of ground equations can also be viewed as an equivalence relation on nodes, named as constants, in a DAG representation of ground terms in which nonleaf nodes are labeled with function symbols and represent subterms, with an edge from the function labeling the node to the nodes serving as its arguments. A DAG representation supports full sharing of ground subterms.

is an edge from a constant  $c_i$  to  $c_j$  toward a root (implying that  $c_i > c_j$ ), then there is a rewrite rule  $c_i \rightarrow c_j$ , and (ii) for each nonconstant uninterpreted symbol  $h \in F_U$ , the set of uninterpreted function symbols, there is a finite set of flat rewrite rules of the form  $h(c_1, \dots, c_k) \rightarrow d$  on root constants in the forest with the property that no two left sides of the rewrite rules are identical. However, if Union-Find algorithm is used for generating a canonical rewrite system on constant equations, ordering on constants cannot be chosen a priori and instead should be chosen dynamically based on how various equivalence classes of constants get merged to maintain the balanced trees representing each equivalence class. For simplicity, in the rest of the paper, a total ordering on constants a priori will be assumed for generating a canonical rewrite system  $R_S$  to represent the congruence closure  $CC(S)$  of  $S$ .

**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:  $\forall x, y, z, f(x, y) = f(y, x)$  and  $f(x, f(y, z)) = f(f(x, y), z)$ . 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 all variables  $x_i$ 's in  $s, t$  and any ground substitution  $\sigma$  of  $x_i$ 's,  $\sigma(s) = \sigma(t) \in CC(S)$ .

$CC(S)$  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.

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:

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

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  $(a \neq c \vee b \neq d) \wedge (b \neq c \vee a \neq d)$ . For an AC symbol  $g$ , there are many possibilities; as an example, for the disequation  $g(a, g(b, c)) \neq g(a, g(a, g(a, c)))$  implies  $(a \neq g(a, c) \vee b \neq a \vee c \neq a) \wedge (b \neq g(a, c) \vee g(a, c) \neq g(a, a)) \dots$ .

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.** Following [Kap97], 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  $h(c_1, \dots, c_k) = d$ , and (iii) for each  $f \in F_{AC}$ ,  $f(c_1, \dots, c_j) = f(d_1, \dots, d_{j'})$ , where  $c$ 's,  $d$ 's are constants in  $C$ ,  $h \in F_U$ , and every AC symbol  $f$  is viewed to be variadic



(including  $f(c)$  standing for  $c$ ). 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  $f$ .<sup>2</sup> Nonconstant arguments of a mixed AC term  $f(t_1, \dots, t_k)$  are transformed to  $f(u_1, \dots, u_k)$ , 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 including those 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  $f$ , if a flat term is  $f(a_1, \dots, a_k)$ , it is written as  $f(M)$  with a multiset  $M = \{\{a_1, \dots, a_k\}\}$ ;  $f(M)$  is also called an  $f$ -**monomial**. In case  $f$  has its identity  $e$ , i.e.,  $\forall x, f(x, e) = x$ , then  $e$  is written as is, or  $f(\{\{e\}\})$ . A singleton constant  $c$  is ambiguously 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 flat ground equations on constants, equivalently, equations of the form  $f(M_i) = f(M_{i'})$ .

### 3. CONGRUENCE CLOSURE WITH A SINGLE ASSOCIATIVE-COMMUTATIVE (AC) SYMBOL

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

Checking whether a ground equation on AC terms is in the congruence closure of a finite set  $S$  of ground equations is the word problem over finitely presented commutative algebraic structures, presented by  $S$  characterizing their interpretations as discussed in [KKN85, LeC83, BL81]. 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  $f$ . Section 2.3 gives a general definition of semantic congruence closure both in the presence of uninterpreted and interpreted symbols including AC symbols. For a single AC symbol  $f$ ,  $ACCC(S)$  becomes: if  $f(M_1) = f(M_2)$  and  $f(N_1) = f(N_2)$  in  $ACCC(S)$ , where  $M_1, M_2, N_1, N_2$  could be singleton constants, then  $f(M_1 \cup N_1) = f(M_2 \cup N_2)$  is also in  $ACCC(S)$ .

<sup>2</sup>There are two types of flattening being used: (i) one for flattening arguments of AC symbols, converting them to be of variadic, (ii) creating flat terms in which function symbols have constants as arguments by extending the signature. This abuse of terminology can hopefully be disambiguated from the context of its use.

As in [Kap97], 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), 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. Two popular families of admissible orderings on monomials are degree ordering (also called degree-lexicographic orderings). A degree ordering compares multisets  $f(A)$  and  $f(B)$  first on their size and in case of equal size, the largest constant in the difference  $A - B$  is bigger than every constant in  $A - B$ :  $f(A) \gg f(B)$  if and only if  $|A| > |B|$  or if  $|A| = |B|$ , then  $\exists c \in A - B, \forall d \in B - A, c \gg d$ . In a lexicographic ordering, size does not matter:  $f(A) \gg f(B)$  if and only if  $\exists c \in A - B, \forall d \in B - A, c \gg d$ .

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  $f(A_1) \rightarrow f(A_2)$ , where  $f(A_1) \gg_f f(A_2)$ . The rewriting relation induced by this rewrite rule is defined below.

**Definition 3.1.** A flattened term  $f(M)$  is *rewritten* in one step, denoted by  $\rightarrow_{AC}$  (or simply  $\rightarrow$ ), using a rule  $f(A_1) \rightarrow f(A_2)$  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 [BN98] are used to represent and study properties of the reflexive and transitive closure of  $\rightarrow_{AC}$  induced by  $R_S$ ; the reflexive, symmetric and transitive closure of  $\rightarrow_{AC}$  is the AC congruence closure  $ACCC(S)$  of  $S$ . Below, the subscript  $AC$  is dropped from  $\rightarrow_{AC}$  and  $f$  is dropped both from  $S_f$  and  $\gg_f$  whenever it is obvious from the context.

A rewrite relation  $\rightarrow$  defined by  $R_S$  is called terminating iff there are no infinite rewrite chains of the form  $t_0 \rightarrow t_1 \rightarrow \dots \rightarrow t_k \rightarrow \dots$ . A rewrite relation  $\rightarrow$  is *locally confluent* iff for any term  $t$  such that  $t \rightarrow u_1, t \rightarrow u_2$ , there exists  $v$  such that  $u_1 \rightarrow^* v, u_2 \rightarrow^* v$ .  $\rightarrow$  is confluent iff for any term  $t$  such that  $t \rightarrow^* u_1, t \rightarrow^* u_2$ , there exists  $v$  such that  $u_1 \rightarrow^* v, u_2 \rightarrow^* v$ .  $\rightarrow$  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  $t \rightarrow u$ .

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, called the **Dickson** ordering, informally capturing when an  $f$ -monomial rewrites another  $f$ -monomial:  $f(M) \gg_D f(M')$  iff  $M'$  is a subset of  $M$ . It is easy to see that for any admissible ordering  $\gg_f, \gg_D \subseteq \gg_f$ . 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 using the Dickson's

lemma, which states that any infinite subset of  $f$ -monomials on finitely many constants must include at least two  $f$ -monomials comparable by the Dickson ordering. The Dickson's lemma is a combinatorial property on  $k$ -tuples of natural numbers which states that any infinite subset  $A \subseteq \mathbb{N}^k$  must include at least two comparable  $k$ -tuples by component-wise ordering. Assuming an ordering on  $k$  constants, an  $f$ -monomial can be represented as a  $k$ -tuple corresponding to the number of times various constants occur in the  $f$ -monomial.

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 [KKN85], the local confluence of  $R_S$  can be checked using the following constructions of superposition and critical pair.

**Definition 3.2.** 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 [BWK93, KMN88, BD88].

**Lemma 3.3.** *An AC rewrite system  $R_f$  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  $AB = (A_1 \cup B_1) - (A_1 \cap B_1)$ .*

*Proof.* Consider a flat term  $f(C)$  rewritten in two different ways in one step using not necessarily distinct rules:  $f(A_1) \rightarrow f(A_2), f(B_1) \rightarrow f(B_2)$ . 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.  $\square$

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 [BN98]. 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  $ACCC(S)$ .

Applying the completion algorithm on the two rules in the above example, 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  $S$  of equations on  $f$ -monomials, and a total ordering  $\gg_f$  on  $f$ -monomials extending a total 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  $l \rightarrow r \in R_f$ .

**SingleACCompletion**( $S = S_f \cup S_C, \gg_f$ ):

- (1) Orient constant equations in  $S_C$  into terminating rewrite rules  $R_C$  using  $\gg$ . 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  $R_C = \cup_{c \in C} \{c \rightarrow \hat{c} \mid c \neq \hat{c} \text{ and } \hat{c} \text{ is the least element in } [c]\}$ .  
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  $\hat{l} \rightarrow \hat{r}$ .
- (3) Generate critical pairs between  $\hat{l} \rightarrow \hat{r}$  and every  $f$ -rule in  $R_f$ , adding them to  $T$ .<sup>3</sup>
- (4) Add the new rule  $\hat{l} \rightarrow \hat{r}$  into  $R_f$ ; **interreduce** other rules in  $R_f$  using the new rule.
  - (i) For every rule  $l \rightarrow r$  in  $R_f$  whose left side  $l$  is reduced by  $\hat{l} \rightarrow \hat{r}$ , remove  $l \rightarrow r$  from  $R_f$  and insert  $l = r$  in  $T$ .
  - (ii) 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  $l \rightarrow r$  in  $R_f$  by  $l \rightarrow r'$ .
- (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  $S$ .

**Example 1:** As a simple example, consider ground equations on an AC symbol  $*$ : 1.  $a * a * b = a * a$ , 2.  $a * b * b = b * b$  with the ordering  $a > b$  on constants. They are oriented from left to right as terminating rewrite rules (using either total degree ordering or pure lexicographic ordering). The overlap of the two left sides gives the superposition  $a * a * b * b$ , giving the critical pair:  $(a * a * b, a * b * b)$ ; their normal forms are:  $(a * a, b * b)$ . The new rule is oriented as: 3.  $a * a \rightarrow b * b$ . This rule simplifies rule 1 to 1'.  $b * b * b \rightarrow b * b$ . The completion algorithm terminates with 1', 2, 3 as the canonical rewrite system.

**Theorem 3.4.** *The algorithm **SingleACCompletion** terminates, i.e., in Step 4, rules to  $R_f$  cannot be added infinitely often.*

*Proof.* By contradiction. A new rule  $\hat{l} \rightarrow \hat{r}$  in Step 4 of the algorithm is added to  $R_f$  only if no other rule can reduce it, i.e, for every rule  $l \rightarrow r \in R_f$ ,  $\hat{l}$  and  $l$  are noncomparable in the

<sup>3</sup>Critical pair generation can be done incrementally using some selection criterion for pairs of rules to be considered next, instead of generating all critical pairs of the new rule with all rules in  $R_f$ .

Dickson ordering  $\gg_D$ . 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 the Dickson's Lemma. This implies that there cannot be infinitely many noncomparable  $f$ -monomials serving as the left hand sides of rules in  $R_f$ . In the interreduction step, new rules, if any, added to replace a deleted rule  $l \rightarrow r$  if  $l$  can be reduced, are always smaller in  $\gg_f$ .  $\square$

**Theorem 3.5.** *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 canonical rewrite system  $R_f$  is generated by the above completion procedure, which serves as a decision procedure for  $ACCC(S)$ .*

The proof of 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 3.6.** *Given a total ordering  $\gg_f$  on  $f$ -monomials, there is a unique reduced canonical rewrite system associated with  $S_f$ .*

*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  $l \rightarrow r$  in  $\gg_f$  on which  $R_1$  and  $R_2$  differ; wlog, let  $l \rightarrow r \in R_1$ . 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  $l' \rightarrow r' \in R_2$  such that  $l \gg_D l'$ ; since  $R_1$  has all the rules of  $R_2$  smaller than  $l \rightarrow r$ ,  $l \rightarrow r$  can be reduced in  $R_1$ , contradicting the assumption that  $R_1$  is reduced. If  $l \rightarrow r' \in R_2$  where  $r' \neq r$  but  $r' \gg_f r$ , then  $r'$  is not reduced implying that  $R_2$  is not reduced.  $\square$

The complexity of this specialized decision procedure has been proved to require exponential space and double exponential upper bound on time complexity [MM82].

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  $R_f$ .  $R_f$  thus serves as its unique presentation. Using the same ordering  $\gg_f$  on  $f$ -monomials; two sets  $S_1, S_2$  of AC ground equations have identical (modulo presentation of multisets as AC terms) reduced canonical rewrite systems  $R_{S_1} = R_{S_2}$  iff  $ACCC(S_1) = ACCC(S_2)$ , thus generalizing the result for the uninterpreted case.

#### 4. CONGRUENCE CLOSURE OF A RICHER AC THEORY WITH IDEMPOTENCY, NILPOTENCY, IDENTITY, AND THEIR COMBINATION

If an AC symbol  $f$  has additional properties such as nilpotency, idempotency and/or unit, the above completion algorithm can be easily extended by modifying 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  $R_f$ . This section explores such extensions in case of an AC symbol having idempotency, nilpotency, cancelativity and being an Abelian group. In each case, it is shown that the joinability of additional critical pairs suffices to ensure generation of a canonical rewrite system from ground equations on an AC symbol with additional semantic properties.

**4.1. Idempotency.** If an AC symbol  $f$  is also idempotent, implying  $\forall x, f(x, x) = x$ , the above algorithm **SingleACCompletion** is extended with minor modifications. In the presence of idempotency, multiple occurrences of the same constant as arguments to an idempotent AC symbol can be normalized to a single occurrence, implying that the arguments to an AC symbol can be represented as sets, instead of multisets. For any rule  $f(M) \rightarrow f(N)$  where  $f$  is idempotent and  $M, N$  do not have duplicates, for every constant  $a \in M$ , a superposition  $f(M \cup \{\{a\}\})$  is generated, leading to a new critical pair  $(f(N \cup \{\{a\}\}), f(M))$  and check its joinability. This is in addition to critical pairs generated from pairs of distinct rules used in **SingleACCompletion**.

For an example, from  $f(a, b) \rightarrow c$  with the 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 reduce to  $(f(a, c), c)$  and  $(f(b, c), c)$ , respectively.

With the addition of critical pairs generated from each rule in a rewrite system, the local confluence check becomes as follows:

**Lemma 4.1.** *An AC rewrite system  $R_S$  with an idempotent AC symbol  $f$  (with  $f(x, x) = x$ ) in which no rule in  $R_S$  has  $f$ -monomials with duplicates, 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  $f(A_1) \rightarrow f(A_2), f(B_1) \rightarrow f(B_2)$  is joinable, where  $AB = (A_1 \cup B_1) - (A_1 \cap B_1)$ , and (ii) for every rule  $f(M) \rightarrow f(N) \in R_S$  and for every constant  $a \in M$ , the critical pair,  $(f(M), f(N \cup \{\{a\}\}))$ , is joinable.*

*Proof.* Consider a flat term  $f(C)$ , possibly with duplicates, rewritten in two different ways in one step using not necessarily distinct rules and/or  $f(x, x) \rightarrow x$ . There are three cases: (i)  $f(C)$  is rewritten in two different ways in one step using  $f(x, x) \rightarrow x$  to  $f(C - \{\{a\}\})$  and  $f(C - \{\{b\}\})$  with  $a \neq b$ : The idempotent rule can be applied again on both sides giving  $f(C - \{\{a, b\}\})$  since  $C - \{\{a\}\}$  includes  $\{\{b, b\}\}$  and  $C - \{\{b\}\}$  includes  $\{\{a, a\}\}$ .

(ii)  $f(C)$  is rewritten in two different ways, with one step using  $f(x, x) \rightarrow x$  and another using  $f(M) \rightarrow f(N)$ : 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  $f(M) \rightarrow f(N)$  is  $(f(M), f(N \cup \{\{a\}\}))$ . Irrespective of whether  $a \in M$  or not,  $C - \{\{a\}\}$  is a supermultiset of  $M$ ; to show joinability of  $(f(M), f(N \cup \{\{a\}\}))$  apply all rewrites to  $f(C - \{\{a\}\})$  as well as to  $f((C - M) \cup N)$ .<sup>4</sup>

The proof of the third case is the same as that of Lemma 3.3 and is omitted.  $\square$

The completion algorithm **SingleACCompletion** extends for an idempotent AC symbol by considering critical pairs for every rule in  $R_f$ . Its termination proof is similar based on Dickson's lemma. Theorems 3.5 and 3.6 also extend to the idempotent case with similar related proofs.

**Example 2:** Revisiting Example 1 from the previous section with the additional assumption that  $*$  is idempotent, the equations simplify to:  $a * b = a, a * b = b$ . When oriented into terminating rewrite rules,  $a * b \rightarrow a, a * b \rightarrow b$  generating a superposition  $a * b$  with the associated critical pair:  $(a, b)$  leading to a new rule:  $a \rightarrow b$ , assuming  $a > b$ . This rule simplifies all the other two rules, leading to a canonical rewrite system  $\{a \rightarrow b\}$ .

<sup>4</sup>Hence the need for this extra critical pair due to idempotency.

**4.2. Nilpotency.** Consider a nilpotent AC symbol  $f$  with the property that  $\forall x, f(x, x) = e$  where  $e$  is a constant, typically standing for the identity of  $f$  satisfying the property that  $\forall x, f(x, e) = x$ . Below we do not assume that  $f(x, e) = x$ ; that discussion is postponed to a later subsection where an AC symbol  $f$  has identity as well as is idempotent.

For every rule  $f(M) \rightarrow f(N) \in R$ , generate for every  $a \in M$ , an additional critical pair  $(f(N \cup \{\{a\}\}), f((M - \{\{a\}\}) \cup \{\{e\}\}))$ . With this additional check for joinability, local confluence can be proved as shown below.

**Lemma 4.2.** *An AC rewrite system  $R_S$  with a nilpotent AC symbol  $f$  (with  $f(x, x) = e$ ), such that no monomial in a rule in  $R_S$  has duplicates, 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  $f(A_1) \rightarrow f(A_2), f(B_1) \rightarrow f(B_2)$  is joinable, where  $AB = (A_1 \cup B_1) - (A_1 \cap B_1)$ , and (ii) for every rule  $f(M) \rightarrow f(N) \in R_S$  and for every constant  $a \in M$ , the critical pair,  $(f(N \cup \{\{a\}\}), f(M - \{\{a\}\} \cup \{\{e\}\}))$  is joinable.*

*Proof.* Consider a flat term  $f(C)$ , possibly with duplicates, rewritten in two different ways in one step using not necessarily distinct rules and/or  $f(x, x) \rightarrow e$ . As in the case of idempotency, there are three cases: (i)  $f(C)$  is rewritten in two different ways in one step using  $f(x, x) \rightarrow e$  to

$f((C - \{\{a, a\}\}) \cup \{\{e\}\})$  and  $f(C - \{\{b, b\}\} \cup \{\{e\}\})$ ,  $a \neq b$ . After single step rewrites, the idempotent rule can be applied again on both sides giving  $f(C - \{\{a, a, b, b\}\} \cup \{\{e, e\}\})$ .

(ii)  $f(C)$  is rewritten in two different ways, with one step using  $f(x, x) \rightarrow e$  and another using  $f(M) \rightarrow f(N)$ : An application of the nilpotency rule implies that  $C$  includes a constant  $a$ , say, at least twice; the critical pair is:  $(f((C - \{\{a, a\}\}) \cup \{\{e\}\}), f((C - M) \cup N))$ .

Consider two cases:  $a \notin M$ , implying that  $C$  still has  $\{\{a, a\}\}$  as a subset, so nilpotency can be applied to get:  $f(((C - M) - \{\{a, a\}\}) \cup \{\{e\}\} \cup N)$  from  $f((C - M) \cup N)$ ;  $f((C - \{\{a, a\}\}) \cup \{\{e\}\})$  can be rewritten using  $f(M) \rightarrow f(N)$  to get the same result.

The second case of  $a \in M$  implies there is a joinable critical pair:  $(f((M - \{\{a\}\}) \cup \{\{e\}\}), f(N))$ . Rewrite steps that make the above critical pair joinable, apply also on  $(f((C - \{\{a, a\}\}) \cup \{\{e\}\}), f((C - M) \cup N))$ , since  $(M - \{\{a\}\}) \cup \{\{e\}\}$  is a multisubset of  $(C - \{\{a, a\}\}) \cup \{\{e\}\}$ .

The third case is the same as that of Lemma 3.3 and is omitted.  $\square$

**Example 3:** Revisiting Example 1 from the previous section and assuming  $*$  to be nilpotent, the above ground equations simplify to  $e * b = e$ ,  $e * a = e$ . The superposition from the terminating rules corresponding to the above equations is:  $a * b * e$  leading to the critical pair:  $(a * e, b * e)$  whose two terms have the same normal form  $e$ . The rewrite system  $\{e * a \rightarrow e, e * b \rightarrow e\}$  is reduced canonical.

**4.3. Identity.** If  $f$  has identity, say  $e$  satisfying  $f(x, e) = x$ , no additional critical pair is needed since from every rule  $f(M) \rightarrow f(N)$ ,  $(f(N \cup \{\{e\}\}), f(M))$  are trivially joinable.

**Lemma 4.3.** *An AC rewrite system  $R_S$  in which an AC symbol  $f$  has an identity  $e$  (with  $f(x, e) = x$ ) such that every  $f$ -monomial in  $R_S$  is already normalized using  $f(x, e) \rightarrow x$ , 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  $AB = (A_1 \cup B_1) - (A_1 \cap B_1)$ .*

*Proof.* Consider a flat term  $f(C)$ , possibly with duplicates and the identity element  $e$ , rewritten in two different ways in one step using not necessarily distinct rules and/or  $f(x, e) \rightarrow x$ . There are three cases: (i)  $f(C)$  is rewritten in two different ways in one step using  $f(x, e) \rightarrow x$  to  $f(C - \{\!\{e\}\!\})$  and  $f(C - \{\!\{e\}\!\})$ , which is trivially joinable.

(ii)  $f(C)$  is rewritten in two different ways, with one step using  $f(x, e) \rightarrow x$  and another using  $f(M) \rightarrow f(N)$ : An application of the identity rule implies that  $C$  includes the identity  $e$ ; the result of one step rewriting is:  $(f(C - \{\!\{e\}\!\}), f((C - M) \cup N))$  The first element in the critical pair can still be rewritten using  $f(M) \rightarrow f(N)$ , giving  $f(((C - M) - \{\!\{e\}\!\}) \cup N)$ ; since  $M$  cannot include  $e$ ,  $C - M$  still has  $e$  because of which  $f(x, e) \rightarrow x$ , giving  $f((C - M) - \{\!\{e\}\!\}) \cup N$ .

The third case is the same as that of Lemma 3.3 and is omitted.  $\square$

**4.4. Idempotency and Identity.** Consider a combination of idempotency and identity, namely a rewrite system in which an AC symbol  $f$  has identity  $e$  as well as is idempotent (with  $\forall x, f(x, x) \rightarrow x, f(x, e) \rightarrow x$ ). In this case, it suffices to consider the additional critical pair due to idempotency since there is no additional critical pair to consider due to the identity.

**Lemma 4.4.** *An AC rewrite system  $R_S$  with an idempotent AC symbol  $f$  that also has identity  $e$ , 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  $f(A_1) \rightarrow f(A_2), f(B_1) \rightarrow f(B_2)$  is joinable, where  $AB = (A_1 \cup B_1) - (A_1 \cap B_1)$ , and (ii) for every rule  $f(M) \rightarrow f(N) \in R_S$  and for every constant  $a \in M$ , the critical pair,  $(f(M), f(N \cup \{\!\{a\}\!\}))$ , is joinable.*

*Proof.* It is assumed that all monomials in  $R_S$  are already normalized using  $f(x, x) \rightarrow x, f(x, e) \rightarrow x$ . The subpart of the proof when idempotency rule is used along with rules in  $R_S$ , or the identity rule is used along with rules in  $R_S$  is the same as in the proofs above for the idempotency and identity cases. The only case left is when idempotency and identity rules are both used.

Consider a flat term  $f(C)$ , possibly with duplicates and the identity element, rewritten in two different ways in one step using  $f(x, x) \rightarrow x$  in one case and  $f(x, e) \rightarrow x$  in the other case, giving the critical pair:  $(f(C - \{\!\{a\}\!\}), f(C - \{\!\{e\}\!\}))$  where wlog,  $a$  appears in duplicate in  $C$  which has also  $e$ . There are two cases:  $a \neq e$ , in which case, apply  $f(x, e) \rightarrow x$  on the first component of the critical pair and  $f(x, x) \rightarrow x$  on the second component with the result in both cases to be  $f(C - \{\!\{a, e\}\!\})$ . In case  $a = e$ , then if  $C$  has at least three  $e$ 's, then the above case applies; if  $C$  has only two  $e$ 's, then the critical pair is  $(f(C - \{\!\{e\}\!\}), f(C - \{\!\{e\}\!\}))$ , which is trivially joinable.  $\square$

**4.5. Nilpotency and Identity.** Consider a combination of nilpotency and identity:  $\forall x, f(x, x) \rightarrow e, f(x, e) \rightarrow x$ . In this case, it suffices to consider the additional critical pair due to nilpotency since there is no additional critical pair to consider due to the identity.

**Lemma 4.5.** *An AC rewrite system  $R_S$  with  $f(x, x) = e, f(x, e) = x$ , such that every rule in  $R_S$  is normalized using  $f(x, x) \rightarrow e, f(x, e) \rightarrow x$  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  $f(A_1) \rightarrow f(A_2), f(B_1) \rightarrow f(B_2)$  is joinable, where  $AB = (A_1 \cup B_1) - (A_1 \cap B_1)$ , and*



(ii) for every rule  $f(M) \rightarrow f(N) \in R_S$  and for every constant  $a \in M$ , the critical pair,  $(f(N \cup \{\{a\}\}), f((M - \{\{a\}\}) \cup \{\{e\}\}))$ , is joinable.

*Proof.* The subpart of the proof when nilpotency rule is used along with rules in  $R_S$ , or the identity rule is used along with rules in  $R_S$  is the same as in the proofs above for the nilpotency and identity cases. The only case left is when nilpotency and identity rules are both used.

Consider a flat term  $f(C)$ , possibly with duplicates and the identity element, rewritten in two different ways in one step using  $f(x, x) \rightarrow e$  in one case and  $f(x, e) \rightarrow x$  in the other case, giving the critical pair:  $(f((C - \{\{a, a\}\}) \cup \{\{e\}\}), f(C - \{\{e\}\}))$ , where wlog,  $a$  appears in duplicate in  $C$  which has also  $e$ . There are two cases:  $a \neq e$ , in which case, apply  $f(x, e) \rightarrow x$  on the first component of the critical pair and  $f(x, x) \rightarrow e$  on the second component with the result in both cases to be  $f(C - \{\{a, a, e\}\})$ . In case  $a = e$ , then if  $C$  has at least three  $e$ 's, then the above case applies; if  $C$  has only two  $e$ 's, then the critical pair is  $(f(C - \{\{e\}\}), f(C - \{\{e\}\}))$ , which is trivially joinable.  $\square$

Algorithm **SingleACCompletion** can be appropriately modified to include the joinability checks for additional critical pairs. The termination proof from the previous subsection extends to each of these cases and their combination. Theorems 3.5 and 3.6 generalize for each of the above cases as well.

## 5. AN AC SYMBOL WITH CANCELATION

An AC function symbols  $f$  has the **cancelation** property, equivalently is called **cancelative** iff  $\forall x, y, z, f(x, y) = f(x, z) \implies y = z$ . If  $f$  does not have a constant, say  $e$  serving as an identity of  $f$  (i.e.,  $\forall x, f(x, e) = x$ ), then  $y, z$  must be a nonempty multiset of constants for flat terms with  $f$ . In the presence of identity for  $f$ ,  $y$  or  $z$  could be the empty subset, representing the identity  $e$ .

Congruence closure with cancelative function symbols can be subtle because it enables generating smaller equalities from larger equalities: from a cancelative AC  $f$ , for example,  $f(a, b) = f(a, c) \implies b = c$ . More interestingly, if a cancelative  $f$  does not have an identity, then the cancelative congruence closure of  $f(a, c) = a$  also includes  $f(b, c) = b$  for any constant  $b \in C$ ; this is so because from  $f(a, c) = a$ , it follows that  $\forall b \in C, f(a, b, c) = f(a, b)$ , giving  $f(b, c) = b$  by cancelation. This is a way to cancel  $a$  by including a common subterm on both sides so that the result after cancelation is not meaningless.

Given a finite set  $S$  of ground equations with a cancelative AC  $f$ , its congruence closure  $CACCC(S)$  is defined by the closure of the  $ACCC(S)$  with the above universal cancelative axiom, i.e., if  $f(A) = f(B) \in CACCC(S)$ , then for any common nonempty multisubset  $C$  of  $A$  and  $B$  and assuming neither  $A - C$  nor  $B - C$  become empty causing  $f(A - C)$  or  $f(B - C)$  to be meaningless,  $f(A - C) = f(B - C) \in CACCC(S)$ . It also follows that for any nonempty multiset  $D$ , given that  $f(A \cup D) = f(B \cup D) \in CACCC(S)$ , every equation obtained after canceling every possible nonempty common multisubset  $G$  such that neither  $f((A \cup D) - G)$  nor  $f((B \cup D) - G)$  are the empty set,  $f((A \cup D) - G) = f((B \cup D) - G) \in CACCC(S)$ . Of course, if  $f$  has the identity  $e$ , then any of the above multisets can be the empty set representing  $e$ .

For example, the AC congruence closure of  $SC_1 = \{f(a, a, b) = f(a, b, b)\}$  includes all pairs of  $f$ -monomials that can be generated by adding  $f(a, a, b, X) = f(a, b, b, X)$  where  $X$  is any multiset of constants; assuming the degree ordering on  $f$ -monomials extending  $a > b$ , its

canonical rewrite system is  $R = \{f(a, a, b) \rightarrow f(a, b, b)\}$ . If  $f$  is assumed to be also cancelative, the cancelative congruence closure then includes  $a = b$  along with the above congruences, since  $f(a, f(a, b)) = f(b, f(a, b)) \implies a = b$  by cancelation; from  $a = b$ , one gets many other congruences including  $f(a, a) = f(a, b)$ ,  $f(a, b) = f(b, b)$ ,  $f(a, \dots, a) = f(b, \dots, b)$  with equal numbers of  $a$  and  $b$ . A canonical rewrite system associated with the cancelative congruence closure  $CACCC(SC_1)$  is  $R_C = \{a \rightarrow b\}$  using which each of the above pairs can be shown to be in the cancelative congruence closure; the canonical forms of the two sides in each pair above are identical.

Given a finite set  $S = \{f(A_i) = f(B_i) \mid 1 \leq i \leq k\}$  of ground equations, it is easy to see that if  $f(A'_i) = f(B'_i)$  is generated from  $f(A_i) = f(B_i)$  after cancelation of common subterms on both sides, the cancelative congruence closure of  $S$  is contained in that of  $S'$  in which  $f(A'_i) = f(B'_i)$  replaces  $f(A_i) = f(B_i)$ . That motivates simplifying ground equations by canceling out common subterms from both sides.

**5.1. Cancelativity Closed:** As a first step toward generating a cancelative congruence closure, equations are made **cancelatively closed**. An equation  $f(A) = f(B)$  is **cancelatively closed** for a cancelative AC  $f$  iff (i) if  $f$  has an identity  $e$ , then  $A \cap B$  is the empty multiset, implying there is no subterm common among the two sides; (ii) if  $f$  does not have an identity, then  $A \cap B$  is a singleton multiset with exactly one of  $A$  and  $B$  being a constant (i.e., singleton multiset); this captures the property that a constant can be common on both sides to avoid the meaningless term  $f(\{\})$ . For examples,  $f(a, b) = f(c, d)$  is cancelativity closed irrespective of whether  $f$  has an identity or not;  $f(a, b) = a$  is not cancelativity closed if  $f$  has an identity, in which case  $a$  can be canceled from both sides of the equation resulting in  $b = e$ ; however,  $f(a, b) = a$  is cancelativity closed in the absence of  $f$  having an identity, but  $f(a, b, b) = f(a, b)$  is not cancelatively closed since either  $a$  or  $b$  can be canceled without making the resulting equation meaningless.

A set of equations is cancelatively closed iff every equation in the set is cancelatively closed. Similarly, a rewrite rule  $f(A) \rightarrow f(B)$  is cancelatively closed if the corresponding equation  $f(A) = f(B)$  is cancelativity closed; a set of rules is cancelatively closed iff every rule in the set is cancelatively closed.

**5.2. Cancelative Closure of  $S$ :** Given a system  $S$  of ground equations with a cancelative AC symbol  $f$ , an equivalent system  $S'$  is generated from  $S$  that is cancelativity closed such that  $CACCC(S) = CACCC(S')$  as follows: for every ground equation  $f(A) = f(B) \in S$ , (i) if it is cancelatively closed, then it is included in  $S'$ ; (ii) if  $f$  has an identity, then in  $f(A - B) = f(B - A)$  is included in  $S'$  with one of the sides possibly being the identity; (iii) otherwise, if  $f$  does not have an identity and (a) neither  $B - A$  nor  $A - B$  are the empty sets, then  $f(A - B) = f(B - A)$  is included in  $S'$ , (b) if  $A - B$  or  $B - A$  is the empty set, then for every constant  $c \in C$ , a new equation  $f((A - B) \cup \{c\}) = f((B - A) \cup \{c\})$  is included in  $S'$ . Clearly,  $CACCC(S) \subseteq CACCC(S')$ ; using cancelativity, it also follows that  $CACCC(S') \subseteq CASC(S)$ . Let  $CancelClose(S)$  be the set of cancelatively closed equations generated from  $S$ .

For example,  $CancelClose(\{f(a, a, a) = f(b, b)\}) = \{f(a, a, a) = f(b, b)\}$ ,  $CancelClose(\{f(a, b, b) = f(a, b, a)\}) = \{b = a\}$ , and  $CancelClose(\{f(a, a, a, b, b) = f(a, b, b, b)\}) = \{f(a, a) = b\}$  irrespective of whether  $f$  has the identity or not. If  $f$  has the identity  $e$ , then  $CancelClose(\{f(a, b) = a\}) = \{b = e\}$ , but in the absence of  $f$  having an

identity,  $CancelClose(\{f(a, b) = a\}) = \{f(b, c) = c \mid c \in C\}$  which will include  $f(a, b) = a$ ,  $f(b, b) = b$  and for any other constant  $c \in C$ ,  $f(b, c) = c$ .

Running **SingleACCompletion** on a finite set of cancelatively closed ground equations as well as keeping new ground equations derived during the completion as cancelatively closed (such as cancelatively closing rules before adding to the current basis in steps 2 and 3(ii)) is not sufficient to generate a canonical rewrite system for a cancelative congruence closure. This is illustrated by the following example: Let  $SC_2 = \{f(a, a, a) = f(b, b), f(b, b, b) = f(a, a)\}$ . They are cancelatively closed since the two sides of each equation do not have any common subterms. Orienting the above equations from left to right, generates a canonical rewrite system which is also cancelativity closed; however, the rewrite system does not represent a cancelative congruence closure of  $SC_2$  since  $f(a, a, b) = a$  as well as  $f(a, b, b) = b$  are in the cancelative congruence closure (since  $f(a, a, a, b) = f(b, b, b) = f(a, a)$  as well as  $f(b, b, b, a) = f(a, a, a) = f(b, b)$ ) but they have distinct normal forms with respect to the rules generated from  $SC_2$  by orienting the equations in  $SC_2$  from left to right. The rules obtained from orienting the equations in  $SC_2$  from left to right are non-overlapping, so **SingleACCompletion** does not generate any new rule.

**Theorem 5.1.** *SingleACCompletion with steps 2 and 3(ii) modified to keep rules cancelativity closed does not generate a canonical rewrite system for an AC cancelative  $f$  symbol.*

In a typical proof of local-confluence of rewrite rules, any term that includes both left sides of two distinct rules, are joinable when rewritten using these rules in two different ways. However, for a cancelative  $f$ , this step can generate a nontrivial cancelative superposition as follows: given two distinct rules  $f(A_1) \rightarrow f(B_1)$  and  $f(A_2) \rightarrow f(B_2)$ ,  $f(A_1 \cup A_2)$  can be disjointly rewritten using the above rules and shown to be joinable; but  $f(A_1 \cup A_2) = f(B_1 \cup B_2)$  may not be cancelatively closed, possibly giving rise to new critical pairs  $(f(A'_1 \cup A'_2), f(B'_1 \cup B'_2))$  obtained from the cancelative closure of  $f(A_1 \cup A_2) = f(B_1 \cup B_2)$ . For the local confluence of  $\{f(A_1) \rightarrow f(B_1), f(A_2) \rightarrow f(B_2)\}$ , the joinability of  $(f(A'_1 \cup A'_2), f(B'_1 \cup B'_2))$  must be checked.

For example, from  $SC_2$  above, joining the two equations gives a congruent pair  $(f(a, a, a, b, b, b), f(a, a, b, b))$  that is not cancelatively closed. The pair is trivially joinable as the rules corresponding to  $SC_2$  can be disjointly applied; however, a new congruence pair is generated because of cancelation. From  $f(a, a, a, b, b, b) = f(a, a, b, b)$ , new congruences,  $f(a, b, b) = b$  and  $f(a, a, b) = a$  are generated because of cancelation and in the absence of  $f$  having an identity; in the presence of the identity  $e$  for  $f$ , only one congruence pair  $f(a, b) = e$  is generated. Consequently, orienting the two equations in  $SC_2$  from left to right does not result in a canonical rewrite system for its cancelative congruence closure. It will be shown that with the cancelative closure of the disjoint superposition which was trivial in the absence of cancelativity, must be accounted for in the local confluence check of a rewrite system for the cancelative congruence closure. For  $SC_2$ ,  $\{f(a, a, a) \rightarrow f(b, b), f(b, b, b) \rightarrow f(a, a), f(a, a, b) \rightarrow a, f(a, b, b) \rightarrow b\}$  is a canonical rewrite system assuming  $C = \{a, b\}$ . In case  $f$  has the identity  $e$ , then the third and fourth rules can be simplified to be  $f(a, b) \rightarrow e$ .

**5.3. Cancelative Disjoint Superposition:** Given two distinct cancelativity closed rewrite rules  $f(A_1) \rightarrow f(B_1), f(A_2) \rightarrow f(B_2)$  such that the pair  $(f(A_1 \cup A_2), f(B_1 \cup B_2))$  has common subterms, a critical pair  $(f((A_1 \cup A_2) - (B_1 \cup B_2)), f((B_1 \cup B_2) - (A_1 \cup A_2)))$  is generated;

if  $f$  does not have an identity, then the terms in the critical pair cannot be  $f(\{\!\!\{\}\!\!\})$  and multiple critical pairs are generated by ensuring that neither term in a critical pair is  $f(\{\!\!\{\}\!\!\})$ . Further, the cancelative closure of these critical pairs must be generated. This is illustrated above for  $SC_2$ . In case of the example  $SC_3$ , a cancelative disjoint superposition is generated since  $(f(a, b, a, c), f(c, d, b, d'))$  have the common subterm  $f(b, c)$  generating the critical pair  $(f(a, a), f(d, d'))$ .

It is easy to see that new equations generated from cancelative disjoint superposition are in the cancelatively closed congruence closure.

**Theorem 5.2.** *Let  $R$  be a canonical AC rewrite system  $R = \{f(A_i) \rightarrow f(B_i) \mid 1 \leq i \leq k\}$  with an AC symbol  $f$ . If  $f$  is cancelative,  $R$  is cancelativity closed and the critical pairs generated from cancelative disjoint superpositions of distinct pairs of rules in  $R$  are also joinable, then  $R$  is a canonical rewrite system for its cancelative congruence closure  $CACCC(S)$ , where  $S$  is the finite set of equations corresponding to rewrite rules in  $R$ .*

In a typical AC congruence relation, a sequence of derivations is of the form:  $f(C_1) \leftrightarrow f(C_2) \leftrightarrow \dots \leftrightarrow f(C_i) \leftrightarrow f(C_{i+1})$ , where  $f(C_j) \leftrightarrow f(C_{j+1})$  using a rule  $f(A) \rightarrow f(B)$  is such that either  $A \subseteq C_j$  in which case  $C_{j+1} = (C_j - A) \cup B$ , or  $B \subseteq C_j$  in which case  $C_{j+1} = (C_j - B) \cup A$ . In a cancelative congruence relation, however, from two congruent terms in the above sequence of derivations, say  $f(C_{j_1})$  and  $f(C_{j_2})$ , by cancelation,  $f(C'_{j_1}) \leftrightarrow f(C'_{j_2})$  can be derived by canceling a common subterm  $f(X)$  (i.e.,  $C_{j_1} = C'_{j_1} \cup X$ ,  $C_{j_2} = C'_{j_2} \cup X$ ), from which another sequence of derivations is generated. Further, a cancellation free derivation of congruence of  $f(C_{j_1})$  and  $f(C_{j_2})$  is not a single step by some equation  $f(L) = f(R) \in S$ , but rather involves multiple steps by multiple equations in  $S$ . Without any loss of generality, a cancellation-free derivation can be rearranged so that cancelation is on two terms congruent by two different equations, say  $f(L_1) = f(R_1)$ ,  $f(L_2) = f(R_2) \in S$ . Cancelative disjoint superposition and the associated critical pairs capture such interaction among equations.

To illustrate, consider a proof of congruence of  $f(a, a, b) \leftrightarrow^* a$  in  $SC_2$ :  $f(a, a, a, b, b, b) \leftrightarrow f(b, b, b, b, b) \leftrightarrow f(a, a, b, b)$  involving two rules  $f(a, a, a) = f(b, b)$ ,  $f(b, b, b) = f(a, a)$ , from which by cancelation,  $f(a, a, b) \leftrightarrow a$ .

In a proof below, it is shown that for any sequence of derivations in  $CACCC(S)$  possibly using cancelation, a new sequence of derivations in  $CACCC(R)$  can be constructed without any cancelation using additional rules generated from cancelative disjoint critical pairs.

To check whether  $f(A) = f(B) \in CACCC(R)$ , both  $A$  and  $B$  may have to be enlarged to  $A', B'$  respectively using some multiset  $C$  of constants such that  $A' = A \cup C$ ,  $B' = B \cup C$ , and  $f(A'), f(B')$  have the same canonical form using  $R$ , i.e.,  $f(A') \leftrightarrow f(B')$  in  $ACCC(R)$ . Determining how much to enlarge  $f(A), f(B)$  can however be a challenge; left-right superpositions address this challenge.

*Proof. Sketch:* Consider a sequence of inferences showing the congruence of  $f(C) \leftrightarrow^* f(D) \in CACCC(S)$ . In general, because of cancelation, this sequence cannot be  $f(C) = f(C_1) \leftrightarrow f(C_2) \dots f(C_i) \leftrightarrow f(C_{i+1}) = f(D)$ , where  $f(C_j) = f(L \cup X)$ ,  $f(C_{j+1}) = f(R \cup X)$  for any  $1 \leq j < i + 1$ , or  $f(C_j) = f(R \cup X)$ ,  $f(C_{j+1}) = f(L \cup X)$  for some  $f(L) = f(R) \in S$ . But instead it is broken into a chain of such subsequences where subsequences are connected using the cancelation inference rule.

For illustration, for  $SC_3$ , a possible inference sequence of  $f(a, a, b) \leftrightarrow a$  is a chain of subsequence  $f(a, a, a, b) \leftrightarrow f(b, b, b) \leftrightarrow f(a, a)$  in which the first and second inferences are due to the first equation and second equation, respectively, but then there is another subsequence  $f(a, a, b) \leftrightarrow a$ , connected to the previous subsequence by cancelation on two congruent terms

$f(a, a, a, b) \leftrightarrow f(a, a)$ , leading to a sequence of derivations:  $f(a, a, a, b) \leftrightarrow f(b, b, b) \leftrightarrow f(a, a)$ ,  $f(a, a, b) \leftrightarrow a$ . Using the rule  $f(a, a, b) \rightarrow a$  generated from cancelative disjoint superposition from rules 1 and 2, a cancelative-free derivation of  $f(a, a, b) \leftrightarrow a$  can be generated.

The following proof involves generating a cancelation-free sequence of derivations from an arbitrary sequence of derivations involving cancelation. When no cancelation is involved, then there is nothing new to construct.

Assume the sequence of inferences relating  $s \leftrightarrow^* t$  employs  $k$  cancelation steps. Proof is by induction on  $k$ .

**Basis  $k = 1$ :**  $s \leftrightarrow^* t \in CACCC(S)$  can be decomposed into two subsequences  $s \leftrightarrow^* s_1$  without cancelation, followed by  $t_1 \leftrightarrow t_2 \leftrightarrow^* t$  without cancelation. In other words,  $s \leftrightarrow^* s_1 \in ACCC(S) = ACCC(R)$  and  $t_1 \leftrightarrow t_2 \leftrightarrow^* t \in ACCC(S) = ACCC(R)$  where  $t_1 \leftrightarrow t_2$  was generated by applying cancelation on two congruent terms  $s_{j_1} \leftrightarrow^* s_{j_2}$  in a cancelation-free sequence of inferences  $s \leftrightarrow^* s_1$ . Without any loss of generality this sequence can be arranged so that the two congruent terms are at the end of the sequence as:  $s_{j_1} \leftrightarrow s_{j_3} \leftrightarrow s_{j_2}$  that includes a cancelative disjoint superposition. By assumption, the critical pairs corresponding to them are joinable, implying that there is a rewrite proof of the pair  $t_1 \leftrightarrow t_2$  generated after cancelation. Since both cancelation-free sequences have rewrite proofs, they can be glued together using a rewrite proof of  $t_1 \leftrightarrow t_2$ , resulting in a cancelation-free derivation.

**Induction Step:** The induction hypothesis is that any derivation with  $k$  cancelations can be converted into a cancelation free derivation. To prove that a derivation with  $k + 1$  cancelations can also be converted into a cancelation free derivation, isolate the last cancelation step by breaking the original derivation with  $k + 1$  cancelations into  $k + 1$  subsequences. Since by the induction hypothesis, for the first  $k$  subsequences, there is a cancelation free derivation, that derivation can serve as a single cancelation-free subsequence and is linked to the  $k + 1$  subsequence built using the  $(k + 1)^{th}$  cancelation step. This case is the same as the  $k = 1$  case. Using the same argument as for the  $k = 1$  case, a cancelation free derivation is generated for these two new subsequences, which by induction hypothesis, gives a cancelation-free derivation for the original derivation with cancelation.  $\square$

### CancelativeACCompletion( $S = S_f \cup S_C, \gg_f$ ):

- (1) Orient constant equations in  $S_C$  into terminating rewrite rules  $R_C$  using  $\gg_C$  and interreduce them. Equivalently, using a union-find data structure, for every constant  $c \in C$ , compute, from  $S_C$ , the equivalence class  $[c]$  of constants containing  $c$  and make  $R_C = \cup_{c \in C} \{c \rightarrow \hat{c} \mid c \neq \hat{c} \text{ and } \hat{c} \text{ is the least element in } [c]\}$ .
- (2)  $S_f = \bigcup_{f(A) - f(B) \in S_f} CancelClose(\{f(A) - f(B)\})$ .
- (3) If  $S_f$  has any constant equalities, remove them from  $S_f$  and update  $R_C$  using them.
- (4) Initialize  $R_f$  to be  $R_C$ .
- (5) Let  $T := S_f$ .
- (6) 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, for each  $l' = r' \in CancelClose(\{\hat{l} = \hat{r}\})$ ,

- (a) orient the result into a terminating rewrite rule using  $\gg_f$ ; abusing the notation and without any loss of generality, each is oriented from left to right.

- (b) Generate both (classical) critical pair (similar to Step 3 in **SingleACCompletion**) and disjoint superposition critical pair between  $l' \rightarrow r'$  and every rule in  $R_f$ , adding them to  $T$ .<sup>5</sup>
- (c) Add the new  $l' \rightarrow r' \in$  into  $R_f$ .
- (d) Interreduce other rules in  $R_f$  using  $l' \rightarrow r'$  (as in Step 4 in **SingleACCompletion**).
- (7) Repeat Step 6 until both types of critical pairs among all pairs of rules in  $R_f$  are joinable, and  $T$  becomes empty.
- (8) Output  $R_f$  as the canonical rewrite system associated with  $CACCC(S)$ .

Using proofs similar to those for the correctness and termination of **SingleACCompletion** including using Dickson's lemma, the correctness and termination of the above algorithm can be established. Theorems 3.5 and 3.6 generalize for this case also.

**Theorem 5.3.** *The algorithm **CancelativeACCompletion** terminates, i.e., in Step 4, rules to  $R_f$  cannot be added infinitely often.*

**Example 4:** Run the above algorithm on  $SC_3 = \{1. f(a, b) = f(c, d), 2. f(a, c) = f(b, d')\}$ , where  $f$  is cancelative, using the degree ordering on  $f$ -terms extending the constant ordering  $a > b > c > d > d'$ . Both the rules are cancelatively closed. Critical pairs among them are generated: the classical critical pair construction from the two rules is 3.  $f(b, b, d') \rightarrow f(c, c, d)$ ; cancelative disjointly superposition constructions gives: 4.  $f(a, a) \rightarrow f(d, d')$ . The resulting reduced canonical rewrite system for the AC cancelative congruence closure of  $SC_3$  is:  $R = \{1. f(a, b) \rightarrow f(c, d), 2. f(a, c) \rightarrow f(b, d'), 3. f(b, b, d') \rightarrow f(c, c, d), 4. f(a, a) \rightarrow f(d, d')\}$ .

**5.4. An AC function symbol being a group operation.** If an AC function symbol  $f$  has in addition to identity, a unique inverse for every element, then it belongs to an algebraic structure of an Abelian group. The completion procedure in that case becomes much simpler in contrast to the **SingleACCompletion** and **cancelativeACCompletion** discussed in the previous section, both in its behavior and complexity. Completion reduces to Gaussian elimination which can also be formulated in matrix form using Smith normal form or Hermite normal form.

Given a finite set  $S$  of ground equations with an AC  $f$  on constants satisfying the group axioms with the identity 0, its congruence closure  $AGCC(S)$  is defined by the closure of  $ACCC(S)$  with the universal group axioms, i.e., if  $f(A) = f(B) \in AGCC(S)$ , then  $f(-A) = f(-B) \in AGCC(S)$ , where  $-A$  stands for the multiset of the inverses of elements in  $A$ , i.e., if  $A = \{a_1, \dots, a_k\}$  in  $f(A)$ , then  $-A = \{-a_1, \dots, -a_k\}$ . Thus,  $f(A) = f(-(-A))$ , as well as  $f(A \cup \{0\}) = f(A) \in AGCC(S)$ .

Assuming a total ordering on constants, because of the cancelation property and using properties of the inverse function, including  $-f(x, y) = f(-x, -y)$ , a ground equation  $f(A) = f(B)$  can be standardized into  $f(A') = f(B')$  where  $A' \cap B' = \emptyset$  and furthermore,  $A'$  consists of the positive occurrences of the largest constant in the ordering in  $A' \cup B'$  whereas  $B'$  has both negative and positive occurrences of constants smaller than the one in  $A'$ .

From a standardized equation  $f(A') = f(B')$ , a rewrite rule  $f(A') \rightarrow f(B')$  is generated since  $A'$  already has the largest constant on both sides of the equation.

<sup>5</sup>Critical pair generation can be done incrementally using some selection criterion for pairs of rules to be considered next, instead of generating all critical pairs of the new rule with all rules in  $R_f$ .

A rewrite rule  $f(A_1) \rightarrow f(B_1)$  rewrites a term  $f(C)$  iff (i)  $f(C)$  has a submonomial  $f(D)$  with the constant of  $A_1$  such that  $A_1 \subseteq D$  and the result of rewriting is the standardized form of  $f((C - A_1) \cup B_1)$ , (ii)  $f(C)$  has a submonomial  $f(D)$  that has negative occurrences of the constant in  $f(A_1)$ , i.e.,  $-A_1 \subseteq D$  and the result of rewriting is the standardized form of  $f((C \cup A_1) \cup (-B_1))$ . For a simple example,  $f(-a - a - b)$  is rewritten using  $f(a, a) \rightarrow f(-b, c)$  to  $f(-a - a - b + a + a + b - c)$  which standardizes to  $f(-c) = -c$ . A term  $f(C)$  is in normal form with respect to a rewrite rule  $f(A_1) \rightarrow f(B_1)$  iff neither  $A_1$  nor  $-A_1$  is a subset of  $C$ .

A rewrite rule  $f(A_1) \rightarrow f(B_1)$  rewrites another rule  $f(A_2) \rightarrow f(B_2)$  with the same constant in  $A_1, A_2$  iff  $A_1 \subseteq A_2$ , leading to  $f(A_2 - A_1) = f(B_2 - B_1)$ . If  $A_2 - A_1$  is empty, then the equation reduces to  $0 = f(B_2 - B_1)$ , which is then brought to standardized form by picking the largest constant in the result.

Let  $+$  be the binary symbol of an Abelian group traditionally instead of  $f$  with 0 as the identity. Let  $ka, k > 0$ , stand for  $a + \dots + a$ , added  $k$  times; similarly  $ka, k < 0$ , stand for  $-a + \dots - a$  added  $|k|$  times, the absolute value of  $k$ .

**Example 5:** For a simple illustration, if there are rewrite rules  $4a \rightarrow B_1, 6a \rightarrow B_2, 10a \rightarrow B_3$ , then using the extended gcd computation,  $2a \rightarrow B_2 - B_1$  is generated along with other rules without  $a$  on their left sides are generated from equations:  $2B_2 = 3B_1, 5B_2 - 5B_1 = B_3$ .

A rewrite rule  $f(A_1) \rightarrow f(B_1)$  can repeatedly reduce another rewrite rule  $f(A_2) \rightarrow f(B_2)$  with the same constant on their left sides until its normal form. Inter-reduction among such rewrite rules can be sped up by using the extended gcd of positive coefficients of the standardized rewrite rules with the same constant appearing in their left hand sides. Particularly, given a set  $\{f(A_1) \rightarrow f(B_1), \dots, f(A_k) \rightarrow f(B_k)\}$  with the identical constant  $c$  in their left hand sides, with  $A_i$  is the multiset with  $j_i$  occurrences of  $c$ , a new rule  $f(A) \rightarrow f(B)$  is generated, where  $A$  is the multiset of  $c$ 's with as many occurrences as the extended gcd  $g$  of  $j_1, \dots, j_k$ , i.e.,  $g = \sum_l i_l * j_l$ , and  $f(B) = f(B'_1 \cup \dots \cup B'_k)$  where  $B'_l$  is the number of occurrences of  $B_l$  multiplied by  $i_l$ . In addition, other rules get simplified after eliminating  $c$  from them using the newly generated rewrite rule which eliminates the constant from other rules by rewriting.

For the above example, the extended gcd of  $4a, 6a, 10a$  is  $2a$  with a Bezout identity being  $2a = 6a - 4a$ . Using the rule  $2a \rightarrow B_2 - B_1$ , the rules  $4a \rightarrow B_1, 6a \rightarrow B_2, 10a \rightarrow B_3$  are normalized eliminating  $a$  from them.

The above process of interreduction is repeated thus generating a triangular form with at most one rewrite rule for each distinct constant, which also serves as a canonical rewrite system  $R_S$  for  $CC(S)$ .

**Theorem 5.4.** *Given a finite set  $S$  of ground equations of terms in an Abelian group and a total ordering on constants, a canonical rewrite system  $R_S$  is generated from the rewrite rules from standardized equations generated from  $S$  by rewriting and inter-reduction among rules.*

*Proof.* Let  $S'$  be the standardized equations generated from  $S$ . Starting with a standardized equation with the largest constant symbol with the least positive integer coefficient. Using inter-reduction on two rules at a time (or using extended gcd computation using multiple rules),  $R$  consisting of rewrite rules with at most one rule for every constant on its left side is generated using the above algorithm.  $\square$

**Example 6:** As another example, consider  $S = \{a + a + b + c = -a + b + b - c, a + b = -a + c + 0, -b - b - c = a - b + c\}$ ; Under a total ordering  $a > b > c$ , the above equations are standardized to  $\{3a = b - 2c, 2a = -b + c, a = -b - 2c\}$ , which orient from left to right as rewrite rules. The third rule  $a \rightarrow -b - 2c$  rewrites the other two rules to:  $\{-3b - 6c = b - 2c, -2b - 4c = -b + c\}$ , which after standardization, generate:  $\{4b \rightarrow -4c, b \rightarrow -5c\}$ . Using the rewrite rule for  $b$ , the canonical rewrite system  $\{a \rightarrow -b - 2c, b \rightarrow -5c, 16c \rightarrow 0\}$  is generated, further fully reduced to,  $\{a \rightarrow 3c, b \rightarrow -5c, 16c \rightarrow 0\}$  which represents the associated  $AGCC(S)$ .

## 6. COMBINING CANONICAL REWRITE SYSTEMS SHARING CONSTANTS

In the forthcoming sections, canonical rewrite systems generated from ground equations expressed using various AC symbols and uninterpreted symbols are combined for the case when the rewrite systems only share common constants. A critical operation performed on an already generated canonical rewrite system is when a new equality on constants implied by another rewrite system, is added/propagated to it. This section explores such an update in depth. It is done for two distinct cases but with similar analysis: (i) a constant equality generated from an AC canonical rewrite system such as from Section 3 is combined with a canonical rewrite system generated from uninterpreted ground equations such as using [Kap97], and (ii) a constant equality generated from a canonical rewrite system generated from uninterpreted ground equations or another AC canonical rewrite system is added to another AC canonical rewrite system. For simplicity, it is assumed that in the monomial termination orderings  $\gg_f$  used in generating canonical rewrite systems, nonconstant ground terms are bigger than constants as is typically the case; this restriction will be relaxed later in Section 9 however to allow great flexibility in choosing termination orderings on ground terms.

Let  $R_U$  be a reduced canonical rewrite system representing a congruence closure of ground equations on uninterpreted symbols assuming a total ordering  $>$  on constants in  $C$ ; sometimes,  $\gg_C$  is used to stand for a total ordering on constants to be consistent with the notation for monomial orderings  $\gg_f$ .

Let  $R'_U = \mathbf{update}_U(R_U, \{a_i = b_i \mid 1 \leq i \leq k\})$  be a reduced canonical rewrite system generated from  $R_U$ , when a finite set of constant equalities  $\{a_i = b_i \mid 1 \leq i \leq k\}$  are added to  $R_U$ ; if needed, the total ordering on constant is extended to consider any new constants not in  $R_U$ .  $R'_U$  is generated as follows. (i) The constant congruence relation defined by  $R_U$  is updated using the constant rules  $\{a_i = b_i \mid 1 \leq i \leq k\}$  by merging congruence classes. Using the total ordering  $\gg_C$  on constants. the least element in a merged congruence class is picked as its canonical form and a new canonical constant rewrite system is generated. Later this update operation on constant rules due to the addition of new constant rules is called  $\mathbf{update}_C(R_C, \{a_i = b_i \mid 1 \leq i \leq k\})$ , where  $R_C$  is a subset of  $R_U$  consisting only of constant rules. (ii) Flat rules with uninterpreted symbols from  $U$  are then updated by replacing constants appearing in them by their new canonical representatives, if any; all flat rules having identical left sides  $h(c_1, \dots, c_j) \rightarrow d_1, h(c_1, \dots, c_j) \rightarrow d_2, \dots, h(c_1, \dots, c_j) \rightarrow d_l$  are replaced by a single rule  $h(c_1, \dots, c_j) \rightarrow d_i$ , where  $d_i$  is the least constant among  $\{d_1, \dots, d_l\}$ ; additional constant rules from  $d_j \rightarrow d_i, d_j \neq d_i$  are added. If during this second step, any new constant rule is generated, then the above steps are repeated until no new constant rules are generated.



Note that the **update<sub>U</sub>** is implicitly used in Section 2.2 during the construction of a canonical rewrite system representing  $CC(S)$ . The correctness of **update<sub>U</sub>** follows from the correctness of a congruence closure algorithm as given in [Kap97, BK20].

**Theorem 6.1.**  *$R'_U$  as defined above is a reduced canonical rewrite system for the congruence closure  $CC(S_U \cup \{a_i = b_i \mid 1 \leq i \leq k\})$  using a total ordering  $\gg_C$  on constants, where  $S_U$  is the set of equations obtained from the rewrite rules in  $R_U$ .*

Another **update<sub>U</sub>** could have been defined in which constant rewrite rules, instead of constant equations, are added to  $R_U$ .

The second case is **update<sub>AC</sub>** when a finite set of constant rewrite rules are added to an AC canonical rewrite system  $R_f$  consisting of rewrite rules on  $f$ -monomials. This case is more interesting since adding a constant equality not only changes the left side of the AC rewrite rules but due to this change, unlike in the uninterpreted case **update<sub>U</sub>**, the orientation of the updated rewrite rules can also change if the left side of an updated rule becomes smaller than its updated right side. This implies that new superpositions and critical pairs need to be considered; thus **SingleACCompletion** must be rerun. Heuristics and data structures can be developed so as to avoid repeating unnecessary work, especially if a pair of rules do not change. New constant rewrite rules generated during the update are kept track since they need to be propagated to other rewrite systems, if needed.

All rules in  $R_f$  that get updated due to new constant rules are collected in the variable  $S$ . They are normalized, oriented and then their superpositions with other rules in  $R_f$  as well as among themselves are recomputed (steps 7–18).

Variable  $T$  keeps track of critical pairs generated from superpositions as rules change; it also includes new equations generated from rewrite rules whose left sides are rewritten when a new rule is added; initially  $T$  is  $S$ . Variable  $NR_C$  is the set of any new constant rules generated which may have to be propagated to other rewrite systems in a combination, leading to changes in them. Otherwise, if  $NR_C$  remains empty, that indicates that even though  $R'_f$  may be different from  $R_f$ , no new constant rewrite rules were generated; consequently, other rewrite systems involved in a combination are not affected.  $NF(l, R_C)$  computes a normal form of an  $f$ -monomial  $l$  using the constant rules in  $R_C$ , and  $NF(l, R_f)$  computes a normal form of  $l$  using an AC rewrite system  $R_f$ .  $NR_C$  stores any new constant rewrite rules generated.

Updates due to new constant rules are repeated until no new constant rules are generated.

Observe that interreduction in Step 16 does not introduce any new constant rule, since if the left side of a rule already in  $R'_f$  gets reduced by the addition of a new rule  $l' \rightarrow r'$ , then that rule is moved from  $R'_f$  to  $T$ ; if only the right side of a rule in  $R'_f$  gets reduced, then the original rule is not a constant rule.

The output of **update<sub>AC</sub>** is an updated canonical AC rewrite system  $R'_f$  generated from a canonical AC rewrite system  $R_f$  due to constant rules in  $R_C$  and the set  $NR_C$  of new constant rewrite rules generated.

**update<sub>AC</sub>** can also be designed so that as soon as new constant equality is generated in Step 15 generating a constant rewrite rule  $c \rightarrow d$ , the rewrite rule is eagerly used by moving to  $T$ , all those rules in  $R'_f$  whose left sides have occurrences of  $c$ .

Generation of additional constant equalities and additional critical pairs among AC rules terminates because of the Dickson's lemma, as in the case of **SingleACCompletion**,

and the fact that there are only finitely many constants which can be possibly made equal. The correctness of  $\text{update}_{AC}$  is patterned after the correctness of **SingleACCompletion**.

---

**Algorithm 1:  $\text{update}_{AC}(\mathbf{R}_f, \mathbf{R}_C, \gg_f)$** 


---

**Input:**  $R_f$  : An AC canonical rewriting system on  $f$ -monomials,  
 $R_C$  : A canonical rewriting system on constants,  
 $\gg_f$ : a monomial ordering on  $f$ -monomials extending  $\gg_C$  constants

**Output:** An updated AC canonical rewrite system, new implied constant rules

- 1  $S \leftarrow \{l = r \mid NF(l, R_C) \neq l, l \rightarrow r \in R_f\}$
- 2  $R_f \leftarrow R_f - \{l \rightarrow r \mid l = r \in S\}$
- 3  $R'_f \leftarrow R_C \cup R_f$
- 4  $NR_C \leftarrow \emptyset$
- 5 **while**  $S \neq \emptyset$  **do**
- 6  $T \leftarrow \emptyset$
- 7 **for each**  $l = r \in S$  **do**
- 8  $S \leftarrow S - \{l = r\}$
- 9 **if**  $NF(l, R'_f) \neq NF(r, R'_f)$  **then**
- 10 Orient  $(NF(l, R'_f), NF(r, R'_f))$  as  $l' \rightarrow r'$  using  $\gg_f$
- 11 **for each**  $l'' \rightarrow r'' \in R'_f$  **do**
- 12 Compute a critical pair  $(s_1, s_2)$  between  $l' \rightarrow r'$  and  $l'' \rightarrow r''$
- 13 **if**  $NF(s_1, R'_f) \neq NF(s_2, R'_f)$  **then**
- 14  $T \leftarrow T \cup \{NF(s_1, R'_f) = NF(s_2, R'_f)\}$
- 15 **end for**
- 16  $R'_f \leftarrow R'_f \cup \{l' \rightarrow r'\}$
- 17 Interreduce other rules in  $R'_f$  using  $l' \rightarrow r'$  as in step 4 of **SingleACCompletion**
- 18 **if**  $l' \rightarrow r'$  is a new constant rule **then**  $NR_C \leftarrow NR_C \cup \{l' \rightarrow r'\}$
- 19 **end for**
- 20  $S \leftarrow T$
- 21 **end while**
- 22 **return**  $(R'_f, NR_C)$

---

**Theorem 6.2.**  $R'_f$  as computed by the above algorithm is a canonical rewrite system for the congruence closure  $ACCCC(S_f \cup S_C)$  using a total ordering  $\gg_f$  on  $f$ -monomials extending  $\gg_C$  on constants, where  $S_f$  is the set of equations obtained from the rewrite rules in  $R_f$  and  $S_C$  are constant equations obtained from constant rules in  $R_C$ .

*Proof. Sketch:* If the left sides of rules in  $R_f$  do not change due to  $R_C$ , then  $R_f$  is still a canonical rewrite system leading to  $R_f \cup R_C$  being a canonical rewrite system for the congruence closure  $ACCCC(S_f \cup S_C)$ .

Otherwise, all rewrite rules in  $R_f$  whose left sides change due to  $R_C$  are processed as equations and normalized, and critical pairs among them as well as with the rest of the rules in  $R_f$  are generated as in **SingleACCompletion**. Completion is redone on all the modified rules, generating a new  $R'_f$  which may generate new constant equalities, leading to

further propagation; this is kept track using a flag *new*. The process is repeated until  $R'_f$  is canonical and no new constant rules are generated.  $\square$

As the reader would notice, **update** operations are used in the combination algorithms in Sections 7 and 8.

## 7. COMPUTING CONGRUENCE CLOSURE WITH MULTIPLE AC SYMBOLS

The extension of the algorithms for computing congruence closure with a single AC symbol in Section 3 to multiple AC symbols is straightforward when constants are shared. Given a total ordering  $\gg_C$  on constants, for each AC symbol  $f$ , define a total well-founded admissible ordering  $\gg_f$  on  $f$ -monomials extending  $\gg_C$ . It is not necessary for nonconstant  $f$ -monomials to be comparable with nonconstant  $g$ -monomials for  $f \neq g$ , thus providing considerable flexibility in choosing termination orderings for computing AC ground congruence closure.

We will abuse the terminology and continue to call the AC congruence closure of a finite set  $S$  of ground equations with multiple AC symbols to be  $ACCC(S)$ . Whereas the general definition of semantic congruence closure of ground equations for uninterpreted and interpreted symbols is given in Section 2.3,  $ACCC(S)$  can be obtained as follows: if  $f(M_1) = f(M_2)$  and  $f(N_1) = f(N_2)$  in  $ACCC(S)$ , where  $M_1, M_2, N_1, N_2$  could be singleton constants, then clearly  $f(M_1 \cup N_1) = f(M_2 \cup N_2)$  is in  $ACCC(S)$ . Further, for every AC symbol  $g \neq f$ ,  $g(d_1, e_1) = g(d_2, e_2) \in ACCC(S)$ , where  $d_1, d_2, e_1, e_2$  are new constants introduced for  $f(M_1), f(M_2), f(N_1), f(N_2)$ , respectively, assuming they are already not constants; this purifies mixed AC terms.

It is possible to generate a combined reduced canonical rewrite system for multiple AC symbols in many different ways. One possibility is to independently generate a reduced canonical rewrite system for each  $f \in AC$  using the **SingleACCompletion** algorithm discussed above from a finite set  $S_f$  of equations on  $f$ -monomials using a well-founded ordering  $\gg_f$  on  $f$ -monomials; combine the resulting reduced canonical rewrite systems for each  $f$  by propagating constant rewrite rules among them with **update<sub>AC</sub>** until no new constant rewrite rules are generated.

Another possibility is to successively use **SingleACCompletion** to generate a canonical rewrite system for each  $f \in AC$ ; if any new constant rewrite rules are generated, propagate them first to every reduced canonical rewrite system generated so far, to obtain any new reduced canonical rewrite systems due to the propagation. This process is repeated until no new constant rewrite rules are generated, resulting in a combined reduced canonical rewrite system for all AC symbols considered so far. Only after that,  $S_f$  for an AC symbol  $f$  not considered so far is processed. Below, the first option is employed as it is simpler and easier to understand as well as prove correct.

The algorithm below is divided into two parts: Steps 4–9 constitute the first part; using **SingleACCompletion**, a reduced canonical rewrite system  $R_f$  for each AC symbol is generated after  $S_f$  has been normalized using constant rules in  $R_C$ . Any new constant rules generated are accumulated in the variable  $NR_C$ . The new rules in  $NR_C$  are then used to update  $R_C$ .

If  $R_C$  changes implying that new constant rules must be propagated to each  $R_f$ , then the second part of the algorithm (steps 12–21) is executed. The propagation of new constant rewrite rules to an AC rewrite system  $R_f$  is computed using the **update<sub>AC</sub>** operation presented in the previous section, possibly generating an updated canonical rewrite system

$R'_f$  with perhaps additional constant rules, which must be further propagated. Variable  $NNR_C$  keeps track of the new constant rules during this phase. This process is repeated until  $NNR_C$  becomes empty, implying no new constant rules get generated during updates.

---

**Algorithm 2: CombinationMultAC( $S, \gg$ )**


---

**Input:**  $S = S_C \cup \bigcup_{f \in F_{AC}} S_f$ , a finite set of constant and ground equations over multiple AC symbols;  $\gg = \{\gg_C, \gg_f \mid f \in F_{AC}\}$ , a family of monomial orderings extending a total ordering  $\gg_C$  on constants

**Output:**  $R$ , a canonical rewriting system representing AC congruence closure over multiple AC symbols

- 1 Let  $R_C$  be the reduced canonical rewrite system generated from  $S_C$  using  $\gg_C$
- 2  $NNR_C \leftarrow \emptyset$
- 3  $new \leftarrow false$
- 4 **for** each  $f \in F_{AC}$  **do**
- 5     Replace constants in  $S_f$  by their canonical forms using  $R_C$
- 6     Abusing the notation, call the normalized  $S_f$  also as  $S_f$
- 7      $R_f \leftarrow \mathbf{SingleACCompletion}(S_f, \gg_f)$
- 8      $NNR_C = NNR_C \cup \{l \rightarrow r \in R_f \mid l, r \in C\}$
- 9 **end for**
- 10  $R'_C \leftarrow \mathbf{update}_C(R_C, NNR_C)$
- 11 **if**  $R'_C \neq R_C$  **then**  $new \leftarrow true$
- 12 **while**  $new$  **do**
- 13      $NNR_C \leftarrow \emptyset$
- 14     **for** each  $R_f$  with a rule  $l \rightarrow r$  such that  $l$  is not in canonical form wrt  $R_C$  **do**
- 15          $(R'_f, NNR_C) \leftarrow \mathbf{update}_{AC}(R_f, R_C, \gg_f)$
- 16          $R_f \leftarrow R'_f$
- 17          $NNR_C \leftarrow NNR_C \cup NNR_C$
- 18     **end for**
- 19     **if**  $NNR_C = \emptyset$  **then**  $new \leftarrow false$
- 20     **else**  $NNR_C \leftarrow NNR_C$  ;  $R'_C \leftarrow \mathbf{update}_C(R_C, NNR_C)$
- 21 **end while**
- 22 **return**  $R_S = R_C \cup \bigcup_{R_f \in \mathcal{R}} R_f$

---

This repeated propagation of constant rules and subsequent recomputing of canonical AC rewrite systems eventually terminate because (i) recomputing of an updated canonical rewrite system terminates due to the Dickson's lemma, and (ii) there are only finitely many constants and no new constants are introduced during the combination; steps 11–21 in the algorithm compute this fixed point.

The termination and correctness of the combination algorithm follows from the termination and correctness of the algorithm for a single AC symbol, the fact there are finitely many new constant equalities that can be added, and  $\mathbf{update}_{AC}$  on each  $R_f$  terminates when new constant equalities are added. The result of the combination algorithm is a finite canonical rewrite system  $R_S = R_C \cup \bigcup_{f \in F_{AC}} R_f$ , a disjoint union of sets of 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.

Given that **SingleACCompletion** and **update**<sub>AC</sub> generate reduced canonical rewrite systems,  $R_S$  is unique for a given set of ground equations  $S$  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.

**Theorem 7.1.**  *$R_S$  as defined above is a canonical rewrite system associated with  $ACCC(S)$  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  $\gg_C$  on constants, such that  $ACCC(R_S)$ , with rewrite rules in  $R_S$  viewed as equations, and restricted to the original signature is  $ACCC(S)$ .*

The proof of the theorem follows from the fact that (i) each  $R_f$  is canonical using  $\gg_f$ , (ii) the left sides of all rules are distinct, (iii) these rewrite systems are normalized using  $R_C$ . A stronger result follows if **SingleACCompletion** generates a reduced unique canonical rewrite system for an AC rewrite system using  $\gg_f$  and **update**<sub>AC</sub> keeps the rules in the rewrite system in reduced form, thus maintaining a unique reduced AC canonical rewrite system after updates by new constant rules.

**Example 7:** To illustrate, consider the following set of ground equations after purification, involving  $+$  and  $*$ :  $\{1. a * a * b * b = a, 2. a * b * b * b = b, 3. a * a * a * b = a, 4. a + b = b, 5. b + b = a\}$ . A subsystem of  $*$ -equation is:  $\{1, 2, 3\}$ , and a subsystem of  $+$ -equations is  $\{4, 5\}$ . Using a total ordering  $b > a$  on constants, and degree-ordering on  $+$  and  $*$  monomials, the equations are oriented from left to right.

A canonical rewrite system for  $+$ -equations is generated using the algorithm **SingleACCompletion**: There is a critical pair between rules 4 and 5 with the superposition  $a + b + b$  producing the critical pair:  $(a + a, b + b)$ , giving a new rule: 6.  $a + a \rightarrow a$ ; The set  $\{4, 5, 6\}$  is a reduced canonical rewrite system on  $+$ -rules.

Similarly, **SingleACCompletion** is run on the rules corresponding to 1, 2, 3; a new rule 7.  $a * b \rightarrow a * a$  is generated from 1, 3 (the superposition is  $a * a * a * b * b$  leading to the critical pair  $(a * a, a * b)$ ), reducing all other rules, giving a canonical rewrite system on  $*$ -terms:  $\{8. a * a * a * a \rightarrow a, 9. b \rightarrow a\}$ .

The two canonical rewrite system are combined using the constant rewrite rule  $b \rightarrow a$ . It reduces the canonical rewrite system for  $+$ , collapsing all  $+$  rules to 10.  $a + a \rightarrow a$ . The combined canonical rewrite system for the two subsystems is:  $\{a * a * a * a \rightarrow a, a + a \rightarrow a, b \rightarrow a\}$ .

Below, variations of Example 7 are considered to illustrate other subtle aspects of the combination algorithm. If the set of ground equations in Example 7 include additional ground equations in  $+$ -subsystem:  $\{a + c = d, b + d = c\}$ , with  $d > c > b > a$ , further propagation is caused. Given that rule 9 reduces  $b$  to  $a$  from the above steps, the additional equations when oriented from left to right, would lead to a critical pair between  $a + c \rightarrow d, a + d \rightarrow c$ , generating new rule  $d + d \rightarrow c + c$ . This illustrates that the propagation of new constant rules can lead to additional superpositions among rules which were previously canonical.

**Example 8:** Consider another variation on Example 7 in which the  $+$ -subsystem is different:  $\{c + c = b + b, c + b = c + c\}$ . Using the lexicographic ordering induced by  $b > c > a$  on  $+$  monomials, the canonical rewrite system is:  $\{b + b \rightarrow c + c, c + b \rightarrow c + c\}$ . The canonical rewrite system on  $*$  monomials is still  $\{8. a * a * a * a \rightarrow a, 9. b \rightarrow a\}$ . With the propagation of  $b \rightarrow a$  generated from the canonical rewrite system for  $*$ -monomials to the canonical rewrite system for  $+$ -monomials, the orientation of the rewrite rules in it changes

to  $c + c \rightarrow a + a, c + c \rightarrow c + a$ , giving a new rule:  $c + a \rightarrow a + a$ . The resulting combined reduced rewrite system is:  $\{a * a * a * a \rightarrow a, c + c \rightarrow a + a, c + a \rightarrow a + a, b \rightarrow a\}$ .

**7.1. A Nondeterministic Version.** The above combination algorithm can also be presented as an inference system; inference rules can be non-deterministically applied, dovetailing various steps in generating AC rules in different  $R_f$ 's without generating a full canonical rewrite system for any  $R_f$  one at a time. Constant rules can be eagerly propagated and applied everywhere as soon as they are generated, making the combination algorithm more efficient in practice, since the complexity of an AC congruence closure algorithm as well as the size of the associated canonical rewrite systems depend upon the number of constants used in the algorithm: fewer the number of constant, fewer the steps. A possible disadvantage of this view is that it becomes necessary to give correctness and termination arguments using complex proof and termination orderings (i.e., union of all  $\gg_f$  in this case) as a single object; consequently, proofs cannot be factored out across different AC symbols, sacrificing modularity.

## 8. CONGRUENCE CLOSURE WITH MULTIPLE AC AND UNINTERPRETED SYMBOLS

The algorithm presented in the previous section to compute AC congruence closure with multiple AC symbols is combined with the congruence closure algorithm for uninterpreted symbols in Section 2.2. Similar to combining canonical rewrite systems for various AC congruence closures in the previous section, the key operation is to update various canonical rewrite systems when equalities on shared constants are generated and propagated. Since the algorithm for generating a rewrite system for congruence closure over uninterpreted symbols is of the least computational complexity in contrast to the complexity of AC congruence closure algorithms, it is always preferred to be run before running other AC congruence closure algorithms. The combination algorithm is thus centered around the congruence closure algorithm for the uninterpreted symbols.

In the combination algorithm below,  $CC$  stands for the congruence closure algorithm on constant equations  $S_C$  and uninterpreted ground equations  $S_U$ , using a total ordering  $\gg_C$  on constants and  $\gg_U$ , an ordering in which uninterpreted function symbols are bigger than constants in  $C$ .  $R_U$ , the output of  $CC$ , is a reduced ground canonical rewrite system representing the congruence closure of  $S_C \cup S_U$ .  $R_C$  stands for the constant rules in  $R_U$ .

For every  $f \in AC$ , a finite set  $S_f$  of equations on  $f$ -monomials is successively picked; constants in the equations are replaced by their canonical forms using  $R_C$  and the rewrite rules are made from the resulting equations using a total admissible ordering  $\gg_f$  on  $f$ -monomials. **SingleACCompletion** algorithm is used to generate a canonical AC rewrite system  $R_f$  from  $S_f$  using a monomial ordering  $\gg_f$ . Variable  $NR_C$  accumulates the constant rules generated for each  $f$  in its respective reduced canonical rewrite system  $R_f$  (steps 5–10).

The function **update<sub>C</sub>** is used to update the canonical constant rewrite system  $R_C$ , using any new constant rules in  $NR_C$ . If the update results in a new canonical constant rewrite system, then **update<sub>U</sub>** updates the ground canonical rewrite system for the congruence closure of uninterpreted equations, possibly leading to new constant equalities and associated rewrite rules. A flag variable *new* keeps track of whether any new constant rules have been generated which must be propagated.

Steps 14–26 perform the propagation of new constant rewrite rules to a reduced canonical rewrite system  $R_f$  for each AC symbol  $f$  by invoking **update<sub>AC</sub>**. If new constant rewrite

rules are generated, they are first propagated to  $R_U$  (as in step 13), checking for any new constant rewrite rules, which again must be propagated back to canonical AC rewrite systems in which those constants appear. This propagation among various AC canonical rewrite systems and a rewrite system for congruence closure over uninterpreted symbols, continues until no additional constant rewrite rules are generated.

---

**Algorithm 3: GeneralCongruenceClosure( $S, \gg$ )**


---

**Input:**  $S = S_C \cup S_U \cup \bigcup_{f \in F_{AC}} S_f$ , a finite set of equations on constant, terms with uninterpreted symbols, and AC monomials,  
 $\gg = \{\gg_C, \gg_U\} \cup \{\gg_f \mid f \in F_{AC}\}$  a family of orderings on constants, uninterpreted symbols, and  $f$  monomials extending a total ordering on constants with uninterpreted symbols bigger than constants

**Output:** A canonical rewriting system  $R_S = R_C \cup R_U \cup \bigcup_{f \in AC} R_f$ , representing general congruence closure

```

1   $R_U \leftarrow CC(S_C \cup S_U, \gg_U \cup \gg_C)$ 
2   $R_C \leftarrow \{l \rightarrow r \in R_U \mid l, r \in C\}$ 
3   $NR_C \leftarrow \emptyset$ 
4   $new \leftarrow false$ 
5  for each  $f \in F_{AC}$  do
6    Replace constants in  $S_f$  by their canonical forms using  $R_C$ 
7    Abusing the notation, call the normalized  $S_f$  also as  $S_f$ 
8     $R_f \leftarrow \text{SingleACCompletion}(S_f, \gg_f)$ 
9     $NR_C = NR_C \cup \{l \rightarrow r \in R_f \mid l, r \in C\}$ 
10 end for
11  $R'_C \leftarrow \text{update}_C(R_C, NR_C)$ 
12 if  $R'_C \neq R_C$  then  $new \leftarrow true$ 
13  $R_U \leftarrow \text{update}_U(R_U, NR_C)$ 
14 while  $new$  do
15    $NNR_C \leftarrow \emptyset$ 
16    $new \leftarrow false$ 
17   for each  $R_f$  with a rule  $l \rightarrow r$  such that  $l$  is not in canonical form wrt  $R_C$  do
18      $(R'_f, NNR_C) \leftarrow \text{update}_{AC}(R_f, R_C, \gg_f)$ 
19      $R_f \leftarrow R'_f$ 
20      $NNR_C \leftarrow NNR_C \cup NNR_C$ 
21   end for
22   if  $NNR_C \neq \emptyset$  then
23      $NR_C \leftarrow NNR_C$ 
24      $R_U \leftarrow \text{update}_U(R_U, R_C \cup NNR_C)$ 
25      $new \leftarrow true$ 
26 end while
27 return  $R_U \cup \bigcup_{R_f \in R} R_f$ 

```

---

Let  $\gg_U = > \cup \{h \gg c, h \in F_U, c \in C\}$ . For each AC symbol  $f$ , define a total admissible ordering  $\gg_f$  on  $f$ -monomials extending  $>$  on  $C$ . The input to the **General Congruence Closure** algorithm is a union of finite set  $S_C$  of constant equations, finite set  $S_U$  of nonconstant flattened equations in uninterpreted symbols in  $U$ , and for each AC symbol  $f$ ,

a finite set  $S_f$  of  $f$ -monomial equations in which at least one  $f$ -monomial is not a constant, and total orderings  $\gg_U$  and  $\gg_f$  on  $f$ -monomials extending a total ordering  $\gg_C$  on constants in  $C$ .

The termination and correctness proofs of the above algorithm are factored: they follow from the termination and correctness of the algorithm for a single AC symbol, **update<sub>U</sub>**, **update<sub>AC</sub>** and the fact there are finitely many new constant equalities that can be added. Further the outputs in each case are reduced canonical rewrite systems.

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, and (iii) there are only finitely many constants being shared and finitely many possibly equalities on the shared constants.

The result of the above algorithm is a finite reduced canonical rewrite system from  $S$ ,  $R_S = R_C \cup R_U \cup \bigcup_{f \in F_{AC}} R_f$ , 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 and  $R_U$  consisting of flat rules on uninterpreted symbols, such that the left sides of rules are distinct and the right sides are reduced.  $R_S$  is unique in the extended signature assuming a family of total admissible orderings on  $f$ -monomials for every  $f \in F_{AC}$  and making uninterpreted symbols  $h \in U$  extending a total ordering on constant if  $R_C, R_U$  are reduced, **SingleACCompletion** generates a reduced AC canonical rewrite system and **update<sub>U</sub>** as well as **update<sub>AC</sub>** ensure that the canonical rewrite system generated by them are also reduced.

**Theorem 8.1.** *Given a set  $S$  of ground equations, 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 extend orderings  $\gg_U$  on uninterpreted symbols and  $>$  on constants as defined above, and  $ACCC(R_S)$ , with rules in  $R_S$  viewed as equations, is  $ACCC(S)$  when restricted on the original signature. 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.*

The proof follows from the respective proofs of each of the component algorithm and is routine.

As stated in the previous section on the combination algorithm for multiple AC symbols, a nondeterministic version can be formulated in which various inference steps are interleaved along with any new constant equalities generated to reduce constants appearing in other  $R_f$  eagerly, instead of having to complete an AC rewrite system for each AC symbol. Correctness and termination proofs are however more complicated requiring complex orderings.

If any AC symbol  $f$  has additional properties such as idempotency, nilpotency, identity, cancelativity, or being a group operation, and various combinations (see discussion in Sections 4 and 5), the corresponding reduced canonical  $R_f$  generated from the specialized extensions of **SingleACCompletion** algorithms discussed in Sections 4 and 5 can be used and the above results extend.



## 9. PURE LEXICOGRAPHIC ORDERINGS: CONSTANTS BIGGER THAN NONCONSTANT TERMS

So far, it is required that constant symbols are smaller in monomial orderings than nonconstant terms for any function symbol, be it AC or nonAC. This restriction can be relaxed. In case of uninterpreted (non-AC) symbols, this is achieved by introducing a new constant symbol as already done in Kapur's congruence closure [Kap97]. A similar approach works in case of nonconstant terms with multiple AC symbols.

Using a monomial ordering in which a constant is bigger than a nonconstant monomial, the above combination algorithm for generating a canonical rewrite system in the presence of multiple AC symbols could produce at least two canonical rewrite subsystems for two different AC symbols  $f$  and  $g$  such that  $R_f$  has a rule  $c \rightarrow f(A)$  and  $R_g$  has another rule  $c \rightarrow g(B)$ , with  $A$  and  $B$  of size  $> 1$  (implying that the right sides are nonconstants). Then,  $c$  can possibly have either a canonical form with  $f$  as its outermost symbol or  $g$  as its outermost symbol. Call this case to be that of *a shared constant possibly having multiple distinct normal forms in different AC symbols*. This case must also be considered along with the propagation of constant rewrite rules.

One way to address the above case is by introducing a new constant  $u$ , making  $c > u$ ;  $\gg_f$  and  $\gg_g$  are extended to include monomials in which  $u$  appears with the constraint that  $f(A) \gg_f u$  and  $g(B) \gg_g u$ . Add a new rule  $c \rightarrow u \in R_C$ , replace the rule  $c \rightarrow f(A) \in R_f$  by  $f(A) \rightarrow u$  and  $c \rightarrow g(B) \in R_g$  by  $g(B) \rightarrow u$ ; this way,  $c, f(A), g(B)$  all have a normal form  $u$ . The replacements of  $c \rightarrow f(A)$  to  $f(A) \rightarrow u$  in  $R_f$  and similarly of  $c \rightarrow g(B)$  to  $g(B) \rightarrow u$  in  $R_g$  may violate the local confluence of  $R_f$  and  $R_g$  since reorientation of these equations may result in superpositions with the left sides of other rules. New superpositions are generated in  $R_f$  as well as  $R_g$ , possibly leading to new rules including new constant rules. After local confluence is restored, the result is new  $R'_f$  and  $R'_g$  with  $u$  being the canonical form of  $c$ .

To illustrate, consider  $S = \{c = a + b, c = a * b\}$  with AC  $+, *$ . For an ordering  $c > b > a$  with both  $\gg_+$  and  $\gg_*$  being pure lexicographic,  $R_+ = \{c \rightarrow a + b\}, R_* = \{c \rightarrow a * b\}$ . These reduced canonical rewrite systems have a shared constant  $c$  with two different normal forms implying that  $R_+ \cup R_*$  is not canonical. Introduce a new constant  $u$  with  $c > b > a > u$  (other orderings including  $c > u > b > a$  or  $c > u > a > b$  are also possible); make  $R_S = \{a + b \rightarrow u, a * b \rightarrow u, c \rightarrow u\}$ , which is reduced canonical in the extended signature; however, it is not even locally confluent when expressed in the original signature. A choice about whether the canonical form of  $c$  is an  $f$ -monomial or a  $g$ -monomial is not made as a part of this algorithm since nonconstant  $f$ -monomials and  $g$ -monomials are noncomparable.

Other cases, for instance, the one in which  $c \rightarrow f(A)$  in  $R_f$  but  $g(B) \rightarrow c$  in  $R_g$ , can be considered in a similar way by introducing a new constant  $u$ , to preserve modularity by not including mixed rules on  $f$  monomials and  $g$  monomials.

Only finitely many new constants need to be introduced in this construction; their number depends upon the number of canonical forms a constant can have with different outermost AC symbols.

## 10. 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 9:** Consider an example from [BTV03]:  $S = \{f(a, c) = a, f(c, g(f(b, c))) = b, g(f(b, c)) = f(b, c)\}$  with a single AC symbol  $f$  and one uninterpreted symbol  $g$ .

Mixed term  $f(c, g(f(b, c)))$  is purified by introducing new symbols  $u_1$  for  $f(b, c)$  and  $u_2$  for  $g(u_1)$ ; thus  $\{f(a, c) = a, f(b, c) = u_1, g(u_1) = u_2, f(c, u_2) = b, u_2 = u_1\}$ . (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 (i)  $|M_1| > |M_2|$  or (ii)  $|M_1| = |M_2| \wedge M_1 - M_2$  includes a constant bigger than every constant in  $M_2 - M_1$ .  $f$ -equations are normalized using rules 1 and 2, and are oriented as:  $\{3. f(a, c) \rightarrow a, 4. f(c, u_1) \rightarrow b, 5. f(b, c) \rightarrow u_1\}$ .

Applying the **SingleACCompletion** algorithm, the superposition between 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 above 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 being 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 rewritten to  $u_2$ :  $\{u_1 \rightarrow u_2, g(u_2) \rightarrow u_2, f(a, c) \rightarrow a, f(c, u_2) \rightarrow b, f(b, c) \rightarrow u_2, f(a, b) \rightarrow f(a, u_2), f(b, b) \rightarrow f(u_2, u_2)\}$ . In the original signature, the rewrite system is:  $R'_S = \{f(b, c) \rightarrow g(f(b, c)), g(g(f(b, c))) \rightarrow g(f(b, c)), f(a, c) \rightarrow a, f(c, g(f(b, c))) \rightarrow b, f(a, b) \rightarrow f(a, g(f(b, c))), f(b, b) \rightarrow f(g(f(b, c)), g(f(b, c)))\}$ . The reader should compare  $R'_S$  which seems more complex using complicated terms, with the above system  $R_S$  with different orientation between  $u_1, u_2$ .

**Example 10:** 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  $* \gg g \gg a \gg b \gg c \gg d$ . 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 rule  $b \rightarrow d$  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 constant rule  $a \rightarrow c$  which is added to  $R_C : \{a \rightarrow c, b \rightarrow d\}$ .  $R_U$  becomes  $\{g(d) \rightarrow c\}$ . The implied constant rule is propagated to the AC rewrite system on  $*$ .  $R_C$  normalizes  $R_*$  to  $\{c * c \rightarrow c, c * d \rightarrow d\}$ .

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, the propagation of equalities can result in the left sides of the rules in AC subsystems change, generating new superpositions.

**Example 11:** Consider an example with multiple AC symbols and a single uninterpreted symbol from [NR91]:  $S = \{a + b = a * b, a * c = g(d), d = d'\}$  with AC symbols  $+, *$  and an uninterpreted symbol  $g$ . This example also illustrates flexibility in choosing orderings in the proposed framework.

Purification leads to introduction of new constants :  $\{1. a + b = u_0, 2. a * b = u_1, 3. a * c = u_2, 4. g(d) = u_2, 5. d = d', 6. u_0 = u_1\}$ . Depending upon a total ordering on constants  $a, b, c, d, d'$ , 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  $u_0 > u_1$ . Consider a total ordering  $a \gg b \gg c \gg d \gg d' \gg u_2 \gg u_0 \gg u_1$ . Degree-lexicographic ordering is used on  $+$  and  $*$  monomials. This gives rise to:  $R_C = \{6. u_0 \rightarrow u_1, 5. d \rightarrow d'\}$ ,  $R_U = \{4'. g(d') \rightarrow u_2\}$  along with  $R_+ = \{1. a + b \rightarrow u_1\}$  and  $R_* = \{2. a * b \rightarrow u_1, 3. a * c \rightarrow u_2\}$ . The canonical rewrite system for  $*$  is:  $\{2. a * b \rightarrow u_1, 3. a * c \rightarrow u_2, 7. b * u_2 \rightarrow c * u_1\}$ .

The rewrite system  $R_S = \{1. a + b \rightarrow u_1, 2. a * b \rightarrow u_1, 3. a * c \rightarrow u_2, 4'. g(d') \rightarrow u_2, 5. d \rightarrow d', 6. u_0 \rightarrow u_1, 7. b * u_2 \rightarrow c * u_1\}$  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  $R_S = \{1. a + b \rightarrow u_0, 2. a * b \rightarrow u_0, 6'. u_1 \rightarrow u_0, 3. a * c \rightarrow u_2, 4. g(d') \rightarrow u_2, 5. d \rightarrow d', 7. b * u_2 \rightarrow c * u_0\}$  in which  $a + b$  and  $a * b$  have the normal form  $u_0$  standing for  $a + b$ , different from the one above.

## 11. A GRÖBNER BASIS ALGORITHM AS AN AC CONGRUENCE CLOSURE

Buchberger's Gröbner basis algorithm when extended to polynomial ideals over integers [KK88] can be interestingly viewed as a special congruence closure algorithm, extending the congruence closure algorithm over multiple AC symbols  $+$  and  $*$  which in addition, satisfy the properties of a commutative ring with unit, especially the distributivity property.  $+$  satisfies the properties of an Abelian group with 0 as its identity;  $*$  is AC with identity 1 and additionally with the distributivity property:  $\forall x, y, z, x * (y + z) = (x * y) + (x * z)$ ;  $x * 0 = 0$  as well. To incorporate the distributivity property in the combination, additional superpositions and associated critical pairs must be considered. Relationship between Gröbner basis algorithm and the Knuth-Bendix completion procedure has been investigated in [KK83, Win84, KKW89, Mar96], but the proposed insight is novel. Below, we illustrate this new insight using an example from [KK88].

Abusing the notation,  $x + -(y)$  will be written as  $x - y$ . An additive monomial  $c t$ , where  $c \neq 0$ , is an abbreviation of repeating the monomial  $t$ ,  $c$  times; if  $c$  is positive, say 3, then  $3 t$  is an abbreviation for  $t + t + t$ ; if  $c$  is negative, say  $-2$ , then  $-2 t$  is an abbreviation for  $-t - t$ . A  $*$ -monomial with the coefficient 1 is a pure term expressed in  $*$ , but a monomial  $3 y * y * y$  is a mixed term  $y * y * y + y * y * y + y * y * y$  with  $+$  as the outermost symbol which has  $*$  monomials as subterms.

It can be easily proved that a ground term in  $+, *$  and constants can be brought into a polynomial form using the rules of a commutative ring with unity and by collecting identical  $*$  monomials; for example,  $y * (y + y) + y * y$  is equivalent to its canonical form  $3 y * y * y$ .

Similarly, a ground equation built using these function symbols can be transformed to a polynomial equation.

Associated with a finite set  $S$  of ground equations in  $+, *$  and constants, is a basis  $B$  of polynomial equations over the constants. It can be proved that the congruence closure  $CRICC(S)$  of  $S$  using the properties of a commutative ring with 1 corresponds to the ideal  $I(B)$  generated by the polynomials in the basis  $B$ ; the membership test in  $CRICC(S)$  is precisely the membership test in the associated ideal  $I(B)$ .

Because of the distributivity property, a mixed term of the form  $m*(t_1+t_2)$  can possibly be rewritten by the rules in a rewrite system of  $*$ -monomials as well as by the rules in a rewrite system of  $+$  monomials; further it can be expanded using the distributivity property as well to be  $m*t_1+m*t_2$  which can be further rewritten. This is captured below using new superpositions and the associated critical pairs due to the distributivity property. This is first illustrated using an example.

**Example 12:** Consider an example [KK88]. The input basis is:  $B = \{7 x * x * y = 3 x, 4 x * y * y = x * y, 3 y * y * y = 0\}$  of a polynomial ideal over the integers [KK88]. Ground terms in  $B$  can be written in many different ways, resulting in different sets of ground equations on terms built using  $+, 0, -, *, 1$ ; the above basis  $B$  is one such instance.

None of these equations is pure in  $+$  or  $*$ ; in other words, they are mixed terms both in  $*$  and  $+$ . Purification of the above equations using new constants leads to:  $\{1. 7 u_1 = 3 x, 2. 4 u_2 = u_3, 3. 3 u_4 = 0\}$  with  $4. x*x*y = u_1, 5. x*y*y = u_2, 6. x*y = u_3, 7. y*y*y = u_4\}$ . These equations with new constant symbols are now pure with 1, 2, 3 purely in  $+$  and 4, 5, 6, 7 purely in  $*$ . Let a total ordering on all constants be:  $u_1 \gg u_2 \gg u_4 \gg u_3 \gg y \gg x$ . 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,  $R_+ = \{1. 7 u_1 \rightarrow 3 x, 2. 4 u_2 \rightarrow u_3, 3. 3 u_4 \rightarrow 0\}$ .

**SingleACCompletion** 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. This computation can be utilized in other examples specified using the same monomial set and thus does not have to be repeated.

**SingleACCompletion** generating the congruence closure of the ground equations on  $+$  terms produces  $R_+$  as a reduced canonical rewrite system:  $\{1. 7 u_1 \rightarrow 3 x, 2. 4 u_2 \rightarrow u_3, 3. 3 u_4 \rightarrow 0\}$ .

There are no constant rewrite rules to be propagated from one rewrite system to another.

Due to the distributivity property relating  $*$  and  $+$ , however, there are interactions between a rule on  $*$  monomials and a rule on  $+$  monomials. To illustrate, a  $+$  rule 3,  $3 u_4 \rightarrow 0$  and a  $*$  rule 12 interact  $u_1 * u_4 \rightarrow u_2 * u_2$  because of the common constant  $u_4$ ; Using the distributivity property,  $u_1 * (3 u_4) = u_1 * (u_4 + u_4 + u_4) = u_1 * u_4 + u_1 * u_4 + u_1 * u_4$ . Rule 3 applies on the pure  $+$  monomial in the mixed term  $u_1 * (3 * u_4)$  giving another mixed term  $u_1 * 0$  which rewrites to 0 using a property of CR1; rule 12 repeatedly applies on the pure  $*$  monomials in the mixed term  $u_1 * u_4 + u_1 * u_4 + u_1 * u_4$ , producing  $u_2 * u_2 + u_2 * u_2 + u_2 * u_2$ . Consequently, a new equality on mixed terms  $u_2 * u_2 + u_2 * u_2 + u_2 * u_2 = 0$  is derived. It is purified resulting in new relations on  $*$ -monomials as well as  $+$ -monomials. Using a new constant, say  $v_1$ , for  $u_2 * u_2$ , a new  $+$  rule  $3v_1 \rightarrow 0$  is generated and added to  $R_+$ ;  $u_2 * u_2 \rightarrow v_1$  is added to  $R_*$ .

The pair  $(u_1 * 0, u_2 * u_2 + u_2 * u_2 + u_2 * u_2)$  is a new kind of critical pair, generated from the superposition  $u_1 * (3 u_4) = u_1 * (u_4 + u_4 + u_4) = u_1 * u_4 + u_1 * u_4 + u_1 * u_4$  due to the distributivity property.

In general, given a  $+$  rule  $c u \rightarrow r \in R_+$  and a  $*$  rule  $u * m \rightarrow r' \in R_*$  with a common constant  $u$ , where  $c \in \mathbb{Z} - \{0\}$ ,  $r$  is a  $+$ -monomial,  $m$  is  $*$ -monomial, the superposition is  $(c u) * m$  generating the critical pair  $(r * m, c r')$  on mixed terms.  $r * m$  and  $c r'$  are normalized using distributivity and other rules. The resulting mixed terms are purified by introducing new constants, leading to new equalities on  $*$  monomials as well as  $+$  monomials. They are, respectively, added to  $R_*$  and  $R_+$  after normalizing and orienting them using the monomial ordering.

It can be proved that only this superposition needs to be considered to check local confluence of  $R_+ \cup R_*$  [Kap21].

For more examples of superpositions and critical pairs between rules in  $R_+$  and  $R_*$ , consider rules 3 and 13, which give a trivial critical pair.

Considering all such superpositions, the method terminates, leading to the canonical rewrite system  $\{3 x \rightarrow 0, u_1 \rightarrow 0, u_2 \rightarrow u_3, 3 u_4 \rightarrow 0\}$  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 [KK88].

A very simple example taken from [KK88] consisting of 2 ground equations  $2 x * x * y = y$ ,  $3 x * y * y = x$  is also discussed in [Mar96] illustrating how normalized rewriting and completion can be used to simulate Gröbner basis computations on polynomial equations over the integers. Due to lack of space, a detailed comparison could not be included; an interested reader is invited to contrast the proposed approach with the one there. Comparing with [Mar96], it is reported there that a completion procedure using normalized rewriting generated 90 critical pairs in contrast to AC completion procedure [PS81] which 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 above illustrates the power and elegance of the proposed combination framework. 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  $\mathbb{Z}_4$  [KC09]. For example, in case of  $\mathbb{Z}_5$ , another rule is added:  $1 + 1 + 1 + 1 + 1 \rightarrow 0$  added to the input basis.

## 12. CONCLUSION

A modular algorithm for computing the congruence closure of ground equations expressed using multiple AC function symbols and uninterpreted symbols is presented; an AC symbol could also have additional semantic properties including idempotency, nilpotency, identity, cancelativity as well as being an Abelian group.

The algorithms are derived by generalizing the framework first presented in [Kap97] for generating the congruence closure of ground equations over uninterpreted symbols. The key insight from [Kap97]—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 possibly with additional semantic properties. Such decomposition and factoring enable using congruence closure algorithms for each of the pure subproblems on a single AC symbol and/or uninterpreted symbols independently, which propagate equalities on shared constants. Once the propagation of constant equalities stabilizes (reaches a fixed point), the result is (i) a unique reduced canonical rewrite system 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, nilpotency, cancelativity as well as their combination, as well as an AC symbol satisfying the properties of a group.

Unlike previous proposals based on the Knuth-Bendix completion procedure and its generalizations to AC theories and other equational theories, the proposed algorithms do not need to use extensions rules, AC/E unification or AC/E compatible termination ordering on arbitrary terms. Instead, the proposed framework provides immense flexibility in using different termination orderings for terms in different AC symbols and uninterpreted symbols.

The modularity of the algorithms leads to easier and simpler correctness and termination proofs in contrast to those in [BTV03, Mar96]. 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 [NO07]. 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.

By-products of this research are new insights into combination algorithms for ground canonical rewrite systems— (i) making orderings on ground terms more flexible and general in which constants are allowed to be greater than nonconstant terms as in orderings for Gröbner basis algorithms, as well as (ii) a new kind of superpositions to incorporate semantic properties such as cancelativity, which can be viewed as a superposition between the left side of a rule with the right side of another rule. An instantiation of the proposed combination framework to an AC symbol with identity and another AC symbol enriched with the properties of an Abelian group, leads to a new way to view a Gröbner basis algorithm for polynomial ideals over integers, as a combination congruence closure algorithm. Canonical bases of polynomial ideals over the integers (and more general rings with zero divisors) on extended signature can be computed for which rewriting (simplification) of a polynomial by another polynomial, when viewed on the original signature, need not terminate, opening possible new research directions.

Integration of additional semantic properties of AC symbols is done on a case by case basis by modifying critical pair constructions but avoiding extension rules or complex AC termination orderings compatible with these semantic properties. Using the proposed framework, other types of combinations of congruence closure algorithms in the presence of other semantic properties relating different AC symbols may be possible as well. A general

approach based on the properties of the semantic properties that avoids extension rules and sophisticated machinery due to the AC properties, needs further investigation.

**Acknowledgments:** Thanks to the FSCD conference and LMCS referees for their reports which substantially improved the presentation. Thanks to Christophe Ringeissen for detailed discussions which clarified the contributions of the proposed approach in contrast to other approaches, particularly [BT97, ABR09]. Thanks also to Jose Abel Castellanos Joo for help in formulating some of the algorithms.

## REFERENCES

- [ABRS09] Alessandro Armando, Maria Paola Bonacina, Silvio Ranise, and Stephan Schultz. New results on rewrite-based satisfiability procedures. *ACM Transactions on Computational Logic*, 10(1):4:1–4:51, 2009.
- [BD88] 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.
- [BK20] 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.
- [BL81] A. Michael Ballantyne and Dallas Lankford. New decision algorithms for finitely presented commutative semigroups. *Computers and Mathematics Applications*, 7:159–165, 1981.
- [BN98] Franz Baader and Tobias Nipkow. *Term rewriting and all that*. Cambridge University Press, 1998.
- [BRTV00] 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.
- [BT97] Franz Baader and Cesare Tinelli. A new approach for combining decision procedure for the word problem, and its connection to the Nelson-Oppen combination method. In William McCune, editor, *Automated Deduction - CADE-14, 14th International Conference on Automated Deduction, Townsville, North Queensland, Australia, July 13-17, 1997, Proceedings*, volume 1249 of *Lecture Notes in Computer Science*, pages 19–33. Springer, 1997. doi:10.1007/3-540-63104-6\_3.
- [BTV03] 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.
- [BWK93] Thomas Becker, Volker Weispfenning, and Heinz Kredel. *Gröbner bases - a computational approach to commutative algebra*, volume 141 of *Graduate texts in mathematics*. Springer, 1993.
- [DST80] 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.
- [Kap97] 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.
- [Kap19] 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.
- [Kap21] 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.
- [KB70] 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.

- [KC09] 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.
- [KK83] 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.
- [KK88] 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.
- [KKN85] Abdelilah Kandri-Rody, Deepak Kapur, and Paliath Narendran. An ideal-theoretic approach to word 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.
- [KKW89] 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.
- [KMN88] 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.
- [KN92] 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.
- [KZ95] Deepak Kapur and Hantao Zhang. An overview of Rewrite Rule Laboratory (RRL). *Computers and Mathematics Applications*, 29:91–114, 1995.
- [LeC83] Philippe LeChenadec. *Canonical forms in the finitely presented algebras*. Ph.D. Dissertation, U. of Paris 11, 1983.
- [Mar91] 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.
- [Mar96] 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.
- [MM82] 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.
- [NO80] 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.
- [NO07] 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.
- [NR91] Paliath Narendran and Michaël Rusinowitch. Any ground 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.
- [PS81] 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.
- [Sho78] Robert E. Shostak. An algorithm for reasoning about equality. *Commun. ACM*, 21(7):583–585, 1978. doi:10.1145/359545.359570.
- [Win84] 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.



- [WNW<sup>+</sup>21] Max Willsey, Chandrakana Nandi, Yisu Remy Wang, Oliver Flatt, Zachary Tatlock, and Pavel Panchekha. egg: Fast and extensible equality saturation. *Proc. ACM Program. Lang.*, 5(POPL):1–29, 2021. doi:10.1145/3434304.
- [Zha92] 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.