J Syst Sci Complex (2019) 32: 317–355

Conditional Congruence Closure over Uninterpreted and Interpreted Symbols

KAPUR Deepak

DOI: 10.1007/s11424-019-8377-8
Received: 11 October 2018 / Revised: 24 December 2018

Abstract A framework for generating congruence closure and conditional congruence closure of ground terms over uninterpreted as well as interpreted symbols satisfying various properties is proposed. It is based on some of the key concepts from Kapur's congruence closure algorithm (RTA97) for ground equations based on introducing new symbols for all nonconstant subterms appearing in the equation set and using ground completion on uninterpreted constants and purified equalities over interpreted symbols belonging to different theories. In the original signature, the resulting rewrite systems may be nonterminating but they still generate canonical forms. A byproduct of this framework is a constant Horn completion algorithm using which ground canonical Horn rewrite systems can be generated for conditional ground theories.

New efficient algorithms for generating congruence closure of conditional and unconditional equations on ground terms over uninterpreted symbols are presented. The complexity of the conditional congruence closure is shown to be $O(n*\log(n))$, which is the same as for unconditional ground equations. The proposed algorithm is motivated by our attempts to generate efficient and succinct interpolants for the quantifier-free theory of equality over uninterpreted function symbols which are often a conjunction of conditional equations and need additional simplification. A completion algorithm to generate a canonical conditional rewrite system from ground conditional equations is also presented. The framework is general and flexible and is used later to develop congruence closure algorithms for cases when function symbols satisfy simple properties such as commutativity, nilpotency, idempotency and identity as well as their combinations. Interesting outcomes include algorithms for canonical rewrite systems for ground equational and conditional theories on uninterpreted and interpreted symbols leading to generation of canonical forms for ground terms, constrained terms and Horn equations.

Keywords Completion, congruence closure, conditional congruence closure, intepreted symbols, rewriting, uninterpreted symbols.

KAPUR Deepak
Department of Computer Science, University of New Mexico, Albuquerque, NM, USA.
Email: kapur@cs.unm.edu.
◇ *This paper was recommended for publication by Editor LI Hongbo.*

1 Introduction

Equality reasoning arises in many applications including compiler optimization, functional languages, and reasoning about databases as well as, most importantly, reasoning about different aspects of software and hardware. Significance of congruence closure algorithms on ground equations in compiler optimization and verification applications has long been recognized. Particularly, in the mid 70's and early 80's, a number of algorithms for computing congruence closure were reported in the literature [1–3]. Congruence closure has also been used as a glue to combine different decision procedures for various theories with uninterpreted symbols, arising in program analysis and verification[3–8]. Whereas congruence closure algorithms were implemented in earlier verification systems, their role has become particularly significant in SMT solvers as a way to combine decision procedures.

Conditional equations arise in combining decision procedures especially if one of the component theory is the theory of ground equality over uninterpreted symbols. As an example, in a combination method for generating interpolants for linear arithmetic combined with the theory of ground equality over uninterpreted symbols (EUF)[9], uninterpreted symbols are eliminated by generating Horn clauses. Such conditional equations are often processed in SMT solvers by abstracting them as propositional formulas in which equations are abstracted as propositional variables.

In [10], Gallier combined Downey, et al.'s congruence closure algorithm[2] over ground equations with Dowling and Gallier's linear time algorithm[11] for testing satisfiability of propositional Horn formulas to propose a polynomial time algorithm for testing unsatisfiability of ground Horn clauses with equations. That algorithm seems to have been neglected by the SMT community perhaps because of a folklore that one gets the same effect with CDCL solvers when combined with ground congruence closure. This author is however unaware of any result (proof) showing such a relationship.

In this paper, we focus on the topic of conditional congruence closure on ground equations both in case of uninterpreted as well as interpreted symbols; unconditional congruence closure is merely a byproduct. We propose a new algorithm for computing ground conditional congruence closure which is in the same spirit as the ground congruence closure. This paper builds on two key ideas in our congruence closure algorithm proposed in [12]: (i) flattening (or abstracting) nonconstant ground terms using constants, (ii) unique signatures for nonconstant terms in the same equivalence classes, leading to two separate parts in the algorithm: (a) constant equivalence closure, and (b) generation of implied constant equalities used to update the constant equivalence closure from signature equalities. These ideas provide a flexible framework which is exploited to generate conditional congruence closures by manipulating Horn constant equations. Using these ideas, we give an $O(n * \log(n))$ complexity algorithm for conditional congruence closure; thus this algorithm is of the same complexity as that of (unconditional) congruence closure.

In a later part of the paper, we extend these ideas and the framework incorporating them, to

design congruence closure algorithms (both conditional and unconditional) for ground equalities and conditional equalities over function symbols satisfying properties such as commutativity, idempotency, nilpotency and identity as well as their combinations. The framework can also be used to design congruence closures for conditional ground equalities over interpreted function symbols with associativity and commutativity; this could not be included in this paper because of space limitations. An interested reader can request a draft of the working paper from the author. We present completion algorithms for ground Horn equations both for uninterpreted symbols as well as interpreted symbols. This is done without having to assume any ordering on nonconstant terms.

The key new idea introduced is that of **Horn closure** of rewrite rules — Both unconditional and conditional. This generates additional Horn equations by identifying conditions under which additional ground conditional equalities can be generated. These additional Horn equations can sometimes make the membership test in conditional congruence closure faster by not having to provisionally update conditional congruence closure to account for nonempty hypothesis in a conditional conjecture. As shown in the paper, it suffices to consider Horn equations on constants as nonconstant terms are replaced by introducing new symbols. The output of these algorithms is canonical on the extended signature including new symbols, but is locally confluent[13] on ground terms in the original signature in case the rewrite rules on the original signature are not terminating. Canonical rewrite systems on the extended signature are decision procedures for Horn equations but for the interpreted symbols case, this approach based on generating canonical rewrite systems using a general approach can be more expensive than specialized methods which could be designed exploiting the properties of interpreted symbols.

We propose an algorithm for generating a canonical rewrite system for conditional equations on ground terms (ground Horn equations) which also serves as a congruence closure algorithm for conditional congruence relation induced by ground Horn equations. In contrast to [10], the proposed algorithm is simpler, easier to present as well as to understand. Further, it generates canonical forms for terms constrained by a finite set of ground equations. It also constructs a canonical rewrite system for conditional equations and can thus generate canonical forms for conditional ground equations. Canonical rewrite systems of conditional ground equations can also be used for checking equivalence of conditional congruence closures generated by different sets of conditional ground equations. Like [10], the algorithm can be used to check for unsatisfiability of a finite set of ground Horn equations. It can be extended to allow ground atomic formulas with predicate symbols. The proposed algorithm, when specialized to ground equations, is the same as in [12].

Congruence closure algorithms presented in this paper use a variety of well-known algorithms as primitives, especially for the uninterpreted case. We do not provide many details about well-known algorithms such as Tarjan's[14] UNION-FIND algorithm for constant equivalence closure (see also Galler and Fisher[15]) and its extension by Downey, et al.[2] to ground

congruence closure using signatures and encoding of arbitrary arity function symbols by binary function symbols (called degree-2 graphs in [2]) for nonconstant ground terms (see also Cocke and Schwartz[16] and [17]). The complexity analysis of the presented algorithms is based on the complexity analysis of related constructions in [2, 10, 14]†. Proofs of relevant nontrivial results are given but many routine proofs of obvious and well known results employed are omitted.

1.1 Related Work

The only work directly related is that of [10]. However, it should be obvious that any first-order theorem proving strategy supporting equality when restricted to ground Horn equations (such as paramodulation, superposition, etc) can also solve this problem of satisfiability of ground Horn equations. But a canonical rewrite system can only be generated by superposition calculus like techniques based on clausal superposition, saturation and completion[18–21].

There has been extensive research on conditional (nonground) rewrite systems including testing confluence and generating canonical conditional rewrite systems. Those algorithms can be specialized to work on ground Horn equations but they can often get unnecessarily complicated. These comparisons could not be done in this paper because of space limitations; they are worth further investigation.

The proposed algorithm for conditional congruence closure is an extension of Kapur's algorithm[12]. It is in the spirit of a standard completion procedure[22]. This is in sharp constrast to a satisfiability checking algorithm given by Gallier[10] by merging ground congruence closure of Downey, et al.[2] and Dawling and Gallier's linear algorithm for checking satisfiability of Horn clauses[11]. Gallier's algorithm does not generate unique normal forms or a confluent rewrite system.

In [23], the authors adapted Kapur's congruence closure[12] using its key ideas to an abstract inference system; the Table data structure for new constant symbols defining flat terms introduced during flattening of nested nonconstant terms in $Make\_Rule$ in [12] were called D-rule for defining a flat term and C-rule for introducing a new constant symbol. They expressed various congruence closure algorithms in the literature using these inference steps. They also proposed an extension of this inference system to associative commutative functions, essentially integrating it with Peterson and Stickel's extension[18] of the Knuth-Bendix completion procedure adapted to ground equations; since extension rules used by Peterson and Stickel inherently involve variables, their approach becomes more like Peterson and Stickel's method, which is very different from Kapur's method[12].

In [24], a completion like algorithm for generating a canonical rewrite system from Horn equations is discussed; a Horn equation is made into a conditional rewrite rule with the condition that its left side in the conclusion is bigger than every other term in a reductive simplification ordering in the Horn clause. It is however unclear whether canonical systems generated using

† [2] discussed many variations of the congruence closure algorithm using different data structures and gave their complexity. All those results can be adapted for the algorithms in this paper also.

that approach generate canonical forms for Horn clauses. This work is extended in [25] using abstract canonical inference/presentation to consider ground Horn theories.

2 Preliminaries

Let F be a set of function symbols including constants and $GT(F)$ be the ground terms constructed from F. We will abuse the terminology somewhat by calling a k-ary function symbol as a function symbol (or equivalently, a nonconstant) if $k > 0$ and constant if $k = 0$. Similarly, by a function term, we mean a nonconstant term with a k-ary function, $k > 0$ as its outermost symbol. We assume an unsorted (single-sorted) language but the constructions and algorithms presented here can be extended to sorted languages including order-sorted languages. Every function symbol $f \in F$ is assumed to be uninterpreted; later in the paper we consider function symbols having properties such as commutativity, nilpotency, idempotency and/or identity.

2.1 Equations, Conditional Equations and Congruence Relations

Let S be a finite set $\{(\bigwedge h_1^i = h_2^i) \Rightarrow (c_1^i = c_2^i) | 1 \leq i \leq k\}$ of conditional (Horn) equations where h's and c's are ground terms from $T(F)$; the hypothesis in a Horn (conditional) equation could be empty (meaning `True` and hence an unconditional equation). In addition, a Horn equation of the form $(\bigwedge h_1^i = h_2^i) \Rightarrow$ `False` is allowed. A Horn equation $(\bigwedge h_1^i = h_2^i) \Rightarrow$ `True` is trivial; similarly, if the hypothesis of a Horn equation includes `False`, the Horn equation is trivial as well. Trivial Horn equations are deleted. `True` is deleted from the hypothesis of a Horn equation. `True` $\Rightarrow c_1 = c_2$ is the same as the unconditional Horn equation $c_1 = c_2$. Henceforth, it is assumed that this preprocessing is performed on the input as well as any Horn equation generated during the algorithms.

There are two goals: (i) generate congruence closure to decide whether another conditional equation follows from S or not, and (ii) generate canonical forms of constrained terms (terms under constraints expressed as a conjunction of ground equations) as well as conditional equations so that the algorithm can be used dynamically in an SMT solver. Canonical form computation can be used to decide membership in conditional congruence closure; however, it can be much more expensive if the cost of generating canonical forms is not amortized over numerous membership tests.

2.1.1 Congruence Closure of Equations and Conditional Equations

Definition 2.1 Given a finite set of $S = \{a_i = b_i | 1 \leq i \leq m\}$ of ground equations where $a_i, b_i \in GT(F)$, the congruence closure $CC(S)$ is inductively defined as follows: (i) $S \subseteq CC(S)$, (ii) for every $a \in GT(F)$, $a = a \in CC(S)$, (iii) if $a = b \in CC(S)$, $b = a \in CC(S)$, (iv) if $a = b$ and $b = c \in CC(S)$, $a = c \in CC(S)$, and (v) for every nonconstant $f \in F$ of arity $k > 0$, if $a_i = b_i \in CC(S), 1 \leq i \leq k$, then $f(a_1, \cdots, a_k) = f(b_1, \cdots, b_k) \in CC(S)$. Nothing else is in $CC(S)$.

$CC(S)$ is thus the smallest relation that includes S and is closed under reflexivity, symmetry, transitivity (an equivalence relation), and under function application, making it a congruence

relation. $CC(S)$ is also the equational theory of S.

It is easy to see that $CC(S)$ is in general infinite, but if F consists only of constants, then $CC(S)$ is finite. A remarkable insight of congruence closure algorithms is that even though $CC(S)$ is infinite, it suffices to consider only a finite set of ground terms that are congruent to subterms of S in $CC(S)$ and they suffice for checking membership in $CC(S)$.

Definition 2.2 Given a finite set $S = \{H_i \Rightarrow a_i = b_i | 1 \leq i \leq m\}$, where H_i is a finite (possibly empty) conjunction of ground equations, the congruence closure $C^*(S)$ induced by S is inductively defined as:

1) $C^0(S) = CC(\{a_i = b_i \mid a_i = b_i \in S\})$ where the CC operation gives the congruence closure of unconditional ground equations as defined above.

2) $C^{k+1}(S) = CC(C^k(S) \cup \{a_i = b_i \mid a^i_j = b^i_j \in C^k, \bigwedge_j (a^i_j = b^i_j) \Rightarrow a_i = b_i \in S\})$.

3) $C^*(S) = \bigcup C^i(S)$.

Let $CC(S)$ stand for the closure $C^*(S)$ as defined above; we call $CC(S)$ as the congruence closure of conditional equations in S[10]. As evident, $CC(S)$ is a set of pairs of ground terms and is the equational theory of S. $CC(S)$ in the conditional case is in general also infinite. However, when F has only constant symbols, then $CC(S)$ is finite just like the unconditional case (it is assumed that the trivial and redundant equalities in the hypothesis of a Horn equation are removed). It will be shown below that as in the unconditional case, it suffices to consider only a finite set of ground terms that are congruent to subterms in S in $CC(S)$ and they suffice to decide membership in $CC(S)$ even when S includes Horn equations.

2.1.2 Conditional Equational Closure of Conditional Equations

Using $CC(S)$, the conditional congruence closure $CCC(S)$ is defined as follows; it includes equations and Horn equations and is thus a generalization of $CC(S)$.

Definition 2.3 $H \Rightarrow a = b \in CCC(S)$ iff $a = b \in CC(S \cup H)$.

In this defintion, the hypotheses of a query are added to S and a provisional congruence closure is generated and $a = b$ is checked to be in it. It trivially follows from the definition that

Lemma 2.4 $CC(S) \subseteq CCC(S)$.

The above definition is equivalent to: $CCC(S \cup H \cup \{a = b \Rightarrow \texttt{False}\})$ includes `False`. In this case, membership in $CCC(S)$ is being decided by refutation. It is also easy to see that if $H \Rightarrow a = b \in CCC(S)$, then so is $H' \Rightarrow a = b$, where $CC(H) \subseteq CC(H')$; this capturing the obvious fact that if a and b are equivalent assuming H, then they remain equivalent under H' with additional equalities.

If S has no function symbols other than the constants appearing in them, the problem is simpler; we call it the conditional constant equivalence closure.

2.2 Revisiting Congruence Closure Algorithm from [12]

Below we review the key steps of the congruence closure algorithm proposed in [12]. Without any loss of generality, we may restrict the arity of function symbols to be ≤ 2 since function symbols of arbitrary arity can be replaced using binary tree representation as in [2, 17].

In [12], a quadratic complexity algorithm for congruence closure of ground equations was presented; it has two interacting components: (i) constant equivalence closure relation over constants in the input as well as new constant symbols introduced to flatten nonconstant ground terms (which amounts to introducing symbols for every nonconstant subterm in a DAG representation of a problem) and (ii) inference of additional constant equalities due to congruences using signatures of nonconstant terms. By imposing a total ordering on constant symbols and making all nonconstant function symbols bigger than constants, the algorithm also generates a complete/canonical rewrite system on ground terms built using original symbols and new symbols (called extended signature) in which rewrite rules on ground terms need not be terminating. The canonical form of a ground term could be a ground term properly containing it and further, the term universe need not be fixed a priori. The focus of that paper was to demystify Shostak's congruence closure algorithm which also generated canonical forms but in an ad hoc manner (see [3] and particularly [26] for a detailed analysis of Shostak's algorithm). Since Shostak's algorithm is of quadratic complexity, we did not bother to optimize the algorithm in [12] for its complexity.

It should be noted that the quadratic complexity is due to having to check for each pair of equations with flat terms on the left sides to possibly generate new constant equalities. If the arity of the function symbols is restricted to be less than some fixed arity, then this check requires constant time. Using data structures from [2], including the use of balanced trees for implementing UNION operation on equivalence clases, binary function symbols to encode arbitrary arity function symbols and signature table for function terms built using representatives of equivalence classes, Kapur's algorithm[12] can be modified to have the same complexity as that of [2] for various subcases, in particular $O(n*\log(n))$ (see also [17]). Equality check of flat terms with the same outermost function symbol is built into the data structure and consequently, does not have to be done explicitly. Most importantly, our algorithm is not only easier to undestand and prove correct but also easier to implement, and is in fact supported in Barcelogic SMT solver[17].

Let us revisit the key ideas of Kapur's algorithm[12].

2.2.1 Equivalence Closure of Constant Equations

Given a finite set CE of constant equations, the equivalence relation generated by CE is assumed to be efficiently computed in almost linear time using an equivalence generation algorithm such as Tarjan's UNION-FIND algorithm[14]. The output of the algorithm is a forest of trees, with each tree representing an equivalence class of constants, and the root of the tree being the representative of its equivalence class. Let $EC(CE)$ stand for this equivalence closure. This algorithm is incremental in the sense that equivalence checking as well as equivalence

extension can be done as needed.

Let $Find(c)$ compute the representative of the equivalence containing c; during its computation, path compression is performed to keep the heights of trees small. Let $Union(c, d)$ be the operation that merges the equivalence classes of c and d if they are different: This operation is implemented using $Find(c)$ and $Find(d)$; if c and d are in different equivalence classes, an edge from the root of the tree with the bigger height to the root of the tree with the smaller height is added to keep trees balanced ("modify the smaller half" heuristic (p. 759, [2]). The root of the tree of bigger height is the new representative of the merged equivalence classes. $Find$ takes $O(\log(n))$ steps in the worst case and $O(\alpha(n))$ steps as the amortized cost, where α is the inverse of unary Ackermann's function, which grows extremely slowly; it is almost constant after a certain value and n is the number of constants. In [12], equivalence closure was computed as a canonical rewrite system by incrementally defining an ordering on constants with rewrite rules from a bigger constant to a smaller constant[‡].

2.2.2 Flattening

In [12], a key insight for getting an efficient congruence closure algorithm on ground term equations as a completion procedure is that of *flattening* the input into two types of equations: (i) Constant equations $c = d$ and (ii) *flat equations* $f(c_1, \cdots, c_k) = d$, where $c_1, \cdots, c_k, c, d$ are constants. This is achieved by extending F to include new constant symbols not appearing in the input.

$Flatten(\{c = d\}) = \{c = d\}$,
$Flatten(\{f(c_1, \cdots, c_k) = d\}) = \{f(c_1, \cdots, c_k) = d\}$,
$Flatten(\{d = f(c_1, \cdots, c_k)\}) = \{f(c_1, \cdots, c_k) = d\}$,
$Flatten(\{f(t_1, \cdots, t_k) = c\}) = \{f(c_1, \cdots, c_k) = d, c = d\} \cup \bigcup_{i=1}^{k} Flatten(\{t_i = c_i\})$,
$Flatten(\{c = f(t_1, \cdots, t_k)\}) = Flatten(\{f(t_1, \cdots, t_k) = c\})$,
$Flatten(\{f(t_1, \cdots, t_k) = g(s_1, \cdots, s_{k'})\}) = \{c = d, f(c_1, \cdots, c_k) = c, g(d_1, \cdots, d_{k'}) = d\} \cup$
$\bigcup_{i=1}^{k} Flatten(\{t_i = c_i\}) \bigcup_{i=1}^{k'} Flatten(\{s_i = d_i\})$,

where new constant symbols c, d, c_1, c_2, d_1, d_2, etc., are introduced only for nonconstant subterms, but constant symbols are not replaced by new symbols (for instance, if t_i, s_i are constant symbols, no new symbols are introduced for them). Number of new symbols introduced by the above algorithm 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 dags whose each non-leaf node is assigned a distinct constant).

It is easy to see that the flattening step can be done in linear steps in the input size. All constant symbols in flat terms are representatives of equivalence classes from the first step, which is equivalent to updating their signatures as in [2].

[‡]Choice of an ordering on constants can be delayed until congruence closure is generated. A user/application may have preference for normal forms of terms. Kapur's algorithm is very flexible as it mimics Shostak's algorithm which allows arbitrary canonical forms including those in which the canonical form of a subterm can be a superterm containing it, something not allowed in classical term rewriting approach for generating congruence closure because such rewriting is nonterminating.

2.2.3 Implied Equalities Generation and Incremental Updating of Constant Equivalence

The steps in the algorithm are: (i) Replace constant symbols in f-equations by their representatives (canonical forms); (ii) if two f-equations have identical f-terms, then an implied equality on constants is generated; (iii) in that case, equate the two constant symbols on their right sides and update the equivalence closure by adding the implied equality. This process is repeated until convergence which is guaranteed since there are only finitely many constants and flat terms. Equivalence closure, implied equality generation and propagation steps interleave with each other.

This algorithm terminates when all f-equations have distinct f-terms. Flattening can be done in linear time in the size n of the input; equivalence closure takes $O(n * \log(n))$ steps to perform constant congruence using Tarjan's Union-Find algorithm[14], as in the worst case, there can be $O(n)$ constant symbols; the amortized complexity is $O(n*\alpha(n))$ as per[14]. The third step of generating implied equalities and propagating them is the most expensive; its complexity is determined by the maximum arity of a function symbol. In [12], careful complexity optimization was not done to check when f-terms in f-equations become equal leading to an upper-bound of $O(n^2)$. Using Downey, Sethi and Tarjan's function signature checking technique and encoding non-binary function symbols using binary function symbols, this check can be brought down to $O(n * \log(n))$[2, 17]. Implied equalities generated when two f-equations have identical left sides are used to incrementally update constant equivalence closure. It thus follows:

Theorem 2.5 *The overall complexity of the congruence closure of ground equations using completion is $O(n * \log(n))$, where n is the input size.*

Upon termination of the algorithm, the result is a finite set of distinct constant equations in which the right side of a constant equation is a representative of the congruence class containing its left side, and a finite set of distinct f-equations in which all constants are representatives of congruence classes. This set is also a canonical rewrite system and is reduced and unique given a total ordering on constant symbols, and thus serves as a canonical form of a congruence relation with respect to an ordering. The canonical form of an original constant term is the representative of its congruence class, which could stand for an original constant term or a function term in which all constants are in canonical forms. The canonical form of a nonconstant term is either a constant term in canonical form or a term in which all constants are in canonical forms. Further any two distinct terms have the same canonical form if and only if they are in the same congruence class.

It should be noted that given a flat term or a constant, the canonical rewrite system corresponding to a congruence relation generates its canonical form using original and new constant symbols in one step; the canonical form of a non-flat term is computed in steps no more than the number of function symbols appearing in the term.

Further, only subterms of S and others subterms equivalent to them using new symbols were

used in the above algorithm. Abusing the terminology somewhat, we called the output of the algorithm as the congruence closure $CC(S')$ of S', where S' is S in which subterms have been replaced by new symbols. Strictly speaking, the output is only the restriction of the infinite $CC(S')$ on a restricted finite subset of $GT(F \cup C')$, where C' is the finite set of new constant symbols introduced to stand for subterms in S. The following subsection discusses the basis of this abuse of terminology.

2.2.4 Membership Test

Given a finite set of ground equations S and a query $s = t$, $s = t$ is in the congruence closure of S, denoted by $CC(S)$, iff s and t have the same signature (i.e., identical normal forms). Ground terms s, t are arbitrary elements of $GT(F)$ and are thus not required to be in S or congruent to subterms of terms in S. To distinguish the output of the above algorithm from $CC(S)$, we will use $CCP(S)$, whenever there is possibility of confusion, to stand for the output which is a finite presentation for $CC(S)$. Thus, $CCP(S) = CC(S_C) \cup FE_S$, where $CC(S_C)$ is the forest of trees of congruence classes on constants (original as well as new constants) and FE_S is the finite set of f equations relating flat terms to constants.

Signature of s (which is not necessarily a subterm appearing in S) can be computed by innermost traversal of s replacing each node in the DAG representation by its signature: $sig(c) = Find(c)$ if c is a constant, replacing constants by the representatives of their equivalence classes; $sig(f(s_1, \cdots, s_k)) = sig(f(sig(s_1), \cdots, sig(s_k)))$; this is the same as normalizing s using f-equations by replacing their left sides by the respective right sides; a rewrite which replaces an f-term by a constant always reduces the size of the term until no more rewrites can be performed. This normalization step can be done in $O(k)$ steps, where k is the size of the query in addition to $O(n * \log(n))$ for generating $CC(S)$. Checking membership in $CC(S)$ amounts to the signature computation of both sides of the query. Thus,

Theorem 2.6 *Given a finite presentation $CCP(S)$ of the congruence closure $CC(S)$ of S, the complexity of membership test in $CC(S)$ using rewrite rules is $O(k)$, where k is the size of the query.*

Consider an example: $S = \{a = f(a)\}$. Without any loss of generality, let us introduce a new symbol c for $f(a)$. In the extended signature, the equations are: $\{a = c, f(a) = c\}$. Constant congruence generates a single equivalence class containing both a and c. If c is chosen its representative (canonical form), then we have $\{a \to c, f(c) \to c\}$, implying that the canonical form of both $a, f(c)$ is c on the extended signature; on the original signature, $a, f(f(a))$ have canonical forms $f(a)$. If we wish to check whether $f(f(f(a)) = a$ are in the congruence closure, then we compute canonical forms of $f(f(f(a)))$ which is c; the canonical form of a is also c, establishing their membership in $CC(S)$. The reader should note that the rewrite system representing congruence closure on the extended signature $\{a, f, c\}$ is both terminating and confluent. If new symbols are replaced by subterms for which they were introduced, the resulting rewriting system could be non terminating; for example, the rewrite rule $a \to f(a)$ is

nonterminating; however $f(f(a)) \rightarrow f(a)$ is terminating. Canonical forms of $a, f(a), f^i(a)$ are all $f(a)$.

If a had been chosen as the representative of the equivalence class containing a and c, then the rewrite system representing congruence closure is $\{c \rightarrow a, f(a) \rightarrow a\}$ which is terminating and confluent on the extended as well as original signature. And, the canonical forms of $a, f(a), f^i(a)$ are all a.

This framework provides considerable flexibility in deciding what canonical forms should be associated with various congruence classes. This was the intriguing feature of Shostak's algorithm for generating congruence closure motivating the author to develop the above framework proposed in [12].

3 Conditional Congruence Closure from Horn Equations

Using the above framework for congruence closure for ground equations, we first give an algorithm for computing congruence closure of Horn equations and then extend it to computing conditional congruence closure. In the first subsection, we extend Tarjan's Union-Find algorithm to Horn equations on constants. Afterwards, nonconstant function terms are considered. This extension mirrors the construction used in [12] based on flattening.

3.1 Computing Constant Equivalence Closure from Constant Horn Equations

Let S be the disjoint union of S_C, a finite set of constant equations and S_H, a finite set of constant Horn equations. Let $|S| = n, |S_C| = k, |S_H| = h$ be the sizes based on number of symbols.

1) Apply Tarjan's constant equivalence on the unconditional equations S_C. The resulting $CC(S_C)$ generates a forest of rooted trees in which the root of a tree serves as the representative of the equivalence class containing all constants labeling nodes in the tree. This step takes $O(k * \alpha(k))$[14]. Let $Find(c)$ be the root of the tree in which c appears; we will call $Find(c)$ as the signature of c and its amortized complexity is $\alpha(k)$.

2) **Preprocessing of Horn Equations** $Pre(S_H)$: This steps converts all constants appearing in Horn equations by their representatives. Each conditional equation $(\bigwedge h_1^i = h_2^i) \Rightarrow (c_1^i = c_2^i)$ is replaced by $(\bigwedge Find(h_1^i) = Find(h_2^i)) \Rightarrow Find(c_1^i) = Find(c_2^i)$ after deleting any trivial equality (in which $Find(h_1^j)$ and $Find(h_2^j)$ are identical). If $Find(c_1^i)$ and $Find(c_2^i)$ are identical, then the Horn equation is deleted since it is trivial. Keeping Horn equations on the representatives is one of the invariants of the subsequent steps.

   The amortized complexity of this step is $O(h * \alpha(k))$.

3) **Propagation of Horn Equations** $Prop(S_H)$: This step involves determining whether the hypothesis of any Horn equation(s) is `True` because all equalities appearing in it are `True`. If so, its concluding equality is propagated to be in the equivalence relation. Data structures

similar to those used in Dowling and Gallier's linear time algorithm for Horn satisfiability[11] are used to keep track of representatives occurring in various Horn equations so that if that representative changes due to new implied equalities, the update is reflected immediately. Further, only those Horn equations whose hypotheses become `True` are processed. The details about data structures used as well as the steps of the algorithm are given in the Appendix; they are not included to avoid duplication with similar discussion in [11].

Whenever an implied equality from a Horn equation is propagated, it reduces the number of representatives as well as the number of Horn equations left to be processed.

4) If no remaining Horn equation has its hypothesis in which each equality is valid, the algorithm terminates, generating congruence closure $CC(S)$ as well as conditional congruence closure $CCC(S)$ from the set S of Horn equations.

It should be noted that every equation and Horn equation in S is processed at most once. However, equalities appearing in their hypotheses are updated as many times as the number of new equalities propagated.

The complexity of propagation of implied equalities from Horn equations and updating equivalence closure is $O(h * \log(h))$ since the representative of a constant can change $\log(k')$ times due to tree balancing in Tarjan Union-Find, where $k' \leq h$ is the number of constants in S_H.

Lemma 3.1 *Given $CC(S_C)$, $Prop(S_H)$ is of $O(|S_H| * \log(|S|)$ complexity.*

Proof As discussed above, $Prop(S_H)$ involves computing $Find$ on all constants in the hypothesis of a conditional equation with path compression which is almost linear. In case a hypothesis is `True`, then the equality in its conclusion updates $CC(S_C)$ by merging their equivalence classes using tree balancing. Hence, the result follows. ∎

Theorem 3.2 *The above algorithm on S outputs a constant equivalence relation $CC(S)$ and a finite set H_S of Horn equations on the representatives in $CC(S)$.*

Thus, $CCP(S) = CC(S) \cup H_S$.

Theorem 3.3 *The complexity of computing $CC(S)$ and H_S from S is $O(k * \alpha(k) + |S_H| * \log(|S|))$, where k is the number of constants in S; the complexity of computing $CCC(S)$ is $O(|S| * \log(|S|))$.*

S is said to have no nontrivial Horn equations iff $CCC(S) = CC(S)$, i.e., H_S is the empty set; this happens if Horn equations in S are trivial by itself or are equations since their hypotheses are `True`.

3.1.1 Deciding Membership in Conditional Constant Equivalence Closure

Deciding whether a Horn equation $H \Rightarrow a = b$ is in $CCC(S)$ can be done in many ways as discussed in Subsection 2.1. The easiest but somewhat expensive way is to invoke the Deduction theorem and update $CCC(S)$ using constant equations in H and check whether a, b are in the

same equivalence class in the updated conditional equivalence closure: (i) For each equality $c = d$ in H, if $Find(c)$ and $Find(d)$ are identical, then that equality can be deleted; otherwise, $Find(c) = Find(d)$ is processed the same way as an implied equality updating the CCC data structure discussed earlier. After all equations in H have been processed, $Find(a)$ and $Find(b)$ are computed; if they are identical, then the conjecture Horn equation is in $CCC(S)$; otherwise, it is not.

The complexity of this operation is $O((n + k) * \log(n + k))$ where k is the size of the Horn equation being decided and n is the size of $CCC(S)$. Unlike in the unconditional case, where membership can be decided in almost constant time or $O(k)$ in the presence of nonconstant symbols, where k is the size of the query, congruence closure must be updated to include equalities in the hypothesis of the conjecture.

If S does not have any nontrivial Horn equations, i.e., $CCC(S) = CC(S)$, replacing constants by their representatives in $CC(S)$ is also a decision procedure for $CCC(S)$ without having to invoke the Deduction Theorem. To decide membership in $CCC(S)$ is merely replacing constants by their representatives in a conjecture and checking whether it conclusion follows directly from the equalities in its hypothesis.

The following lemma trivially follows from the definitions.

Lemma 3.4 *$(H \Rightarrow a = b) \in CCC(\{\})$ iff $a = b \in CC(H)$.*

Using the above lemma, the following result follows.

Theorem 3.5 *Given S with no nontrivial Horn equations, a Horn conjecture $(H \Rightarrow a = b)$, $C \in CCC(S)$ iff $C' = (H' \Rightarrow a' = b') \in CCC(\{\})$ where H' is computed by replacing constants in H by their representatives in $CC(S)$, and a', b' are respectively representatives of a, b in $CC(S)$.*

The above theorem gives a decision procedure for $CCC(S)$. The complexity in that case is $O(k)$, where k is the size of the conjecture.

The reason the above theorem does not hold if S has Horn equations which do not simplify to equations is because the hypothesis of a conditional conjecture can imply an equality which otherwise does not follow from $CC(S)$. Consider $S = \{a = b \Rightarrow c = d\}$; the conjecture $(a = c \wedge b = c) \Rightarrow a = d \in CCC(S)$ but $a = d$ does not directly follow from $\{a = c, b = c\}$. A similar situation arises if S has nonconstant equations even when it does not have any nontrivial Horn equation, as discussed later.

3.2 Redundancy in the Presentation of $CCC(S)$

Unlike the congruence closure algorithm for (unconditional) equations, the result of the above algorithm could still include redundant conditional equations including trivial conditional equations in which the conclusion is implied by the hypothesis (see Lemma above). As an example a Horn equation $(a = b \wedge b = c) \Rightarrow a = c$ can be in the output H_S even though the Horn equation is trivial since $a = c$ follows from the hypothesis. This is because constant equivalence closure cannot check whether the conclusion of a Horn equation follows from its

hypothesis without additional work. The next subsection discusses a heuristic to remove some obvious redundancies.

The output H_S can also include two different Horn equations $H_1 \Rightarrow a_1 = b_1$ and $H_2 \Rightarrow a_1 = b_1$ such that the equations in H_2 implies the equations in H_1. For example both $a = b \Rightarrow c = d$ and $(a = b \wedge a' = b') \Rightarrow c = d$ could be in H_S whereas the second Horn equation follows from the first. Rewriting in a later section will capture this kind of redundancy.

3.2.1 Normalized Horn Equations

To remove trivial Horn equations such as $(a = b \wedge b = c) \Rightarrow a = c$, the provisional equivalence relation induced by the equations in the hypothesis of a Horn equation is generated to check whether its conclusion is in this equivalence relation of the hypothesis. The conditional equation can then be deleted as being redundant. Otherwise, the conditional equation is brought into a normal form: The hypothesis is presented as equations specifying the induced partition in a unique fashion by using the total ordering on constants used in the congruence closure.

Let P_E stand for the partition (equivalence relation) induced by a finite set E of constant equations. Given a Horn equation on the representatives (roots) $(a_1^i = b_1^i \wedge \cdots \wedge a_{k_i}^i = b_{k_i}^i) \Rightarrow c = d$, a normalized Horn equation is thus: $\bigwedge_j (u_1^j = u_0^j \wedge \cdots \wedge u_{k_j}^j = u_0^j) \Rightarrow (u = v)$, where j ranges over equivalence classes in the partition P_H defined by $H = \{a_1^i = b_1^i, \cdots \wedge \cdots a_{k_i}^i = b_{k_i}^i\}$. A constant u_0^j is the representative of the j-th equivalence class containing other constants $u_1^j, \cdots, u_{k_j}^j$ different from the representative u_0^j, u, v are the distinct representatives of c and d in P_H, respectively.

It is easy to see that the normalized Horn equation H' of a given Horn equation H generates the same conditional congruence as H. Let the operator N stand for normalizing a Horn equation: $N((a_1^i = b_1^i \wedge \cdots \wedge a_{k_i}^i = b_{k_i}^i) \Rightarrow c = d) = \bigwedge_j (u_1^j = u_0^j \wedge \cdots \wedge u_{k_j}^j = u_0^j) \Rightarrow (u = v)$. For the special case where c, d are identical or u, v are identical, $N(H) = True$. It is easy to see that $N(N(H)) = N(H)$. This operation is of complexity $O(k * \log(k))$, where k is the size of a Horn equation.

As an example, consider $(a = b \wedge b = c \wedge d = e) \Rightarrow a = d$. The hypothesis defines the equivalence classes $\{a, b, c\}, \{d, e\}$. Using the ordering $a \succ b \succ c \succ d \succ e$, the representatives of the two equivalence classes are respectively c, e. The Horn equation is normalized to $(a = c \wedge b = c \wedge d = e) \Rightarrow (c = e)$. A Horn equation $(a = b \wedge b = c) \Rightarrow (a = c)$ normalizes to $(a = c \wedge b = c) \Rightarrow (c = c)$ which is trivial and hence is deleted.

A partition $P_E \subseteq P_{E'}$ iff the equivalence relation that induces P_E is a subset of the equivalence relation for $P_{E'}$, i.e., P_E is a refinement over $P_{E'}$ since more equalities hold in $P_{E'}$ than P_E. We abuse the notation by interchanging between writing a normalized Horn equation as $H \Rightarrow a = b$ or $P_H \Rightarrow a = b$.

To remove the second kind of redundancies discussed above, e.g., between two normalized Horn equations: $\{(a = c \wedge b = c) \Rightarrow d = e, a = b \Rightarrow d = e\}$, the second Horn equation implies the first making it redundant. This can be taken care by rewriting (or simplification) as discussed later in the section on generating a canonical Horn rewrite system using completion

or by additional processing of the hypotheses in Horn equations.

Lemma 3.6 *Given normalized $P_H \Rightarrow a = b$ and $P_{H'} \Rightarrow c = d$ where $P_H \subseteq P_{H'}$, the second Horn equation is implied by the first Horn equation (i.e., $P_{H'} \Rightarrow c = d \in CCC(\{P_H \Rightarrow a = b\})$) iff c, d are in the same equivalence class of the partition induced by $P_{H' \cup \{a=b\}}$.*

This not only deals with Horn equations discussed earlier but shows that $(a = c \wedge b = c) \Rightarrow c = e$ follows from $a = b \Rightarrow b = e$.

So far, only redundancies relating one Horn equation to another have been analyzed. A combination of Horn equations can also make another Horn equation redundant. Consider the following simple example consisting of four Horn equations: $\{1.\ a = b \Rightarrow b = c, \quad 2.\ b = c \Rightarrow c = d, \quad 3.\ a = b \Rightarrow c = d, \quad 4.\ a = b \Rightarrow b = d\}$, which are all normalized. Using the chain rule (transitivity), $a = b \Rightarrow c = d$ follows from 1 and 2; from 1 and 3, $a = b \Rightarrow b = d$. This suggests that many Horn equations are redundant but they may be needed for ensuring the local confluence of a conditional Horn rewrite system.

3.3 Extension to Nonconstant Function Symbols

The extension from the conditional constant equivalence closure of constant Horn equations to conditional congruence closure for the nonconstant case is essentially the same as that of extending Tarjan's Union-Find for constant equivalence closure to congruence closure for ground equations.

The reader needs to be reminded that congruence closure as well as conditional congruence closure are now infinite even though it suffices to consider the restrictions of them on subterms of terms appearing in the input S. We will continue to abuse the terminology of calling these restrictions to be $CC(S)$ and $CCC(S)$ since they suffice to decided membership in $CC(S)$ and $CCC(S)$ respectively.

The steps of the extension to ground Horn equations, emphasizing key points and any differences from congruence closure on constants, are:

Step 1 Flattening Same as in Subsection 2.2.2.

Step 2 Conditional constant equivalence closure This was discussed in Subsection 3.1.

Step 3 Implied constant equalities from f-equations This is the same as in Subsection 2.2.3. We emphasize that implied equalities generated from f-equations with identical left sides can generate additional implied equalities from constant Horn equations.

The result of the algorithm is a forest of trees representing congruence closure on constants, a finite set of constant Horn equations on the roots of the trees and a finite set of f-equations on the roots with distinct left sides. Thus, $CCCP(S) = CC(S_C) \cup H_S \cup FE_S$, where $CC(S_C)$ is the forest of trees representing congruence closure on constants (original as well as new), H_S are constant Horn equations and FE_S are f equations relating flat terms to constants. Recall that $CCP(S) = CC(S_C) \cup FE_S$.

The overall complexity of the algorithm remains the same as in the case of conditional congruence closure of constant Horn equations; including nonconstant symbols only increases

the complexity by the $\log(n)$ factor on the size of the input.

3.4 Deciding a Horn Conjecture

Given $CCCP(S)$ of a finite set S of Horn equations as computed above, the membership of a Horn conjecture $C = (\bigwedge t_1^i = t_2^i \Rightarrow s_1 = s_2)$ in $CCC(S)$ is decided as follows.

1) **Compute Canonical Forms of Terms in C:** Let $\overline{s}$ be the canonical form of a term s by $CCP(S)$. Check if $\overline{s_1}, \overline{s_2}$ are identical, then C follows from S. Otherwise for each equality $t_1^i = t_2^i$ in its hypothesis, compute their canonical forms $\overline{t_1}^i, \overline{t_2}^i$; if identical, that equality is deleted from C. Let $\overline{C} = (\bigwedge \overline{t_1}^i = \overline{t_2}^i \Rightarrow \overline{s_1} = \overline{s_2})$ stand for the normalized form of C on the roots of $CC(S_C)$.

   To avoid cumbersome notation, without any loss of generality, from now onwards, C is assumed to be already in canonical form.

2) **Flatten the normalized conjecture:** Let $Flatten(\{t_1^1 = t_2^1, \cdots, t_1^k = t_2^k\})$ give a set H of f-equations and constant equations, standing for the hypothesis. Let $Flatten(\{s_1 = s_2\})$ also give a set C of f-equations and constants. Then, the flattened form of the conjecture is: $(\bigwedge c_1^i = c_2^i) \Rightarrow d = e$, where a constant equality $c_1^i = c_2^i$ stands for $t_1^i = t_2^i$, and $d = e$ is the constant equality corresponding to $s_1 = s_2$ with $H \cup C$. It is only necessary to introduce new symbols for function terms whose subterms do not appear in S; this minimizes the number of new symbols. This step is linear in the size k, say, of the Horn conjecture; the size of $H \cup C$ of the flattened Horn conjecture is $O(k)$.

3) **Compute** $CCC(CCCP(S) \cup H \cup C \cup \{c_1^1 = c_2^1, \cdots c_1^k = c_2^k\})$: The provisional conditional congruence closure is computed using $CCCP(S)$ and new equations in $H \cup C \cup \{c_1^1 = c_2^1, \cdots, c_1^k = c_2^k\}$. Update constant congruence $CC(S_C)$ first with new constant equations; then updating the representatives of constants appearing in f-equations, possibly generating new implied equalities and also from Horn equations HE_S. The complexity of this step is $O((n+k) * \log(n+k))$ (which is the complexity of $CCCP$ on a problem of size $n+k$), but it can be amortized over various conjectures decided from the same C_S.

4) **Check** $d = e$**:** If $Find(d)$ and $Find(e)$ are the same, then the conjecture follows from S; otherwise, it does not. The complexity of this step is $\log(n+k)$; since this is an addition, it gets absorbed in the overall complexity.

It follows from the proofs of Subsections 2.2.4 and 3.1.1, that:

Theorem 3.7 *Membership in conditional congruence closure $CCC(S)$ can be decided in $O((n+k) * \log(n+k))$ steps where k is the size of the query and $|S| = n$.*

The above complexity is the same as that of computing membership in $CCC(S \cup H)$ because of the use of Deduction theorem, in contrast to $O(k)$ when the invocation of Deduction theorem is avoided.

To summarize, $CCP(S) = CC(S_C) \cup FE_S$; $CCCP(S) = CC(S_C) \cup FE_S \cup HE_S$.

As was proved above in Subsection 3.1, $CCP(S)$ is a decision procedure for $CCC(S)$ without having to invoke the Deduction theorem if S does not have any nontrivial Horn equations (since $CCC(S) = CC(S)$). However, in the presence of nonconstant function symbols, Deduction Theorem has to be invoked to decide membership in $CCC(S)$ even if S does not have any nontrivial Horn equations. Consider $S = \{f(a) = c, f(b) = d\}$. The Horn conjecture $a = b \Rightarrow c = d \in CCC(S)$; obviously, it is not in $CCP(S)$. The hypothesis $a = b$ causes an implied equality from the f-equations. Using Horn closure as defined below, this can be remedied somewhat if S has no nontrivial Horn equations but may have ground equations with nonconstant function symbols.

3.4.1 Horn Closure

Horn closure $HC(S)$ of $CCC(S)$ is defined as follows: (i) $CCC(S) \subseteq HC(S)$, and in addition, (ii) from every pair of distinct flat equations with the same outermost function symbol, say $h(a, b) = c,\ h(a', b') = c' \in HC(S)$, the Horn equation $(a = a' \wedge b = b') \Rightarrow c = c'$ is also in the Horn closure $HC(S)$. Horn equations so generated are on the representatives, are not redundant and further do not generate any implied equalities since all flat terms are using representatives as well[§]. To generate canonical forms for Horn equations, it becomes necessary to add such implied Horn equations from f-equations as will be shown in a later subsection.

4 Canonical Forms

Assume a total well-founded ordering $\succ$ on the extended signature (original and new constants and function symbols); this ordering can extend to a total wellfounded ordering on ground terms in many different ways[13].

A unique representative (canonical form) can be associated with constant, ground term, equation, constrained term as well as a Horn equation for a given S. We first consider the case when there are only constant symbols, and then we include function symbols.

4.1 Constant Equivalence, Tarjan's Union-Find and Canonical Forms

For a finite set S of constant equations, the canonical form of a constant a with respect to $\succ$ is the least constant equivalent, say $\overline{a}$, to a in $CC(S)$. Tarjan's Union Find algorithm generates canonical forms where the ordering is dynamically built based on the input and structure of equivalences; the root of a tree in a forest representing constant equivalence is the canonical form associated with the equivalence class of constants in the tree. Canonical forms (hence the ordering on constants) dynamically change depending upon some measure of the size of the equivalence classes during the Union (merging) step.

[§]Membership of such Horn equations in the conditional congruence closure can be decided invoking the Deduction Theorem, but cannot be decided by simplification.

4.1.1 Canonical Rewrite System Associated with Tarjan's Union-Find

Consider a forest of trees representing an equivalence relation on constants, as generated in Tarjan's Union-Find data structure. The rewrite system associated with the forest is: For every tree, if there is an up arrow (toward the root of its tree) from a node corresponding to a constant c (henceforth also called node c) to a node d, a rule $c \rightarrow d$ is included. From the construction of trees, there is at most one rule for every constant appearing on its left side since there is at most one up arrow from a node to another node.

$Find(c)$ computes the representative of c by traversing up from the node c to the root of its tree, keeping track of all the intermediate nodes. That is equivalent to rewriting c, keeping track of all intermediate results unless the result cannot be rewritten any more. The final result is the root, the canonical form of c and of all other nodes on the path from c to its root. The path compression step in Tarjan's algorithm connects all nodes encountered by up arrows to the root; that is equivalent to replacing all intermediate constants in rewriting directly to the root and deleting all rules used in rewriting. If $c_0 \rightarrow c_1 \rightarrow c_2 \cdots \rightarrow c_k$ (hence, $Find(c_0) = c_k$) because of rules $c_0 \rightarrow c_1, c_1 \rightarrow c_2, \cdots, c_{k-1} \rightarrow c_k$, they are all changed to $c_0 \rightarrow c_k, c_1 \rightarrow c_k, \cdots, c_{k-1} \rightarrow c_k$ in the path compression step.

Since $Union(c, d)$ involves computing $Find(c)$ and $Find(d)$, checking whether the results are different or not. In that case, the root with the "smaller" tree puts an up arrow to the root with the "larger" tree, where the ordering on trees could be based on their size, height or some combination. In the associated rewrite system, a new rule is added with the new root as its right side (since it is the canonical form of the merged tree) and the other root as its left side. Computation of $Find(c)$ and $Find(d)$ also results in processing of existing rewrite rules as discussed above.

It is easy to see the resulting rewrite system is indeed canonical and generates canonical forms which are the roots of the trees. However, it is not reduced as the left and right sides of a rule can be further reduced if the respective node on the left is not directly below a root and/or the respective node on the right is not a root.

Even when a prior total ordering $\succ$ on constants is given, the above construction can be employed ignoring the ordering except for an additional flag with every root indicating whether the root is the least element in its tree along with keeping track of the positions of the least elements. As a final step, for every root, if its flag indicates it is not the least element, then the least element in the equivalence class is identified using $Find$s on the least element and making it the root. This final step can be done without sacrificing efficiency. Upon termination, the least element in every equivalence class is the root of the corresponding tree.

Lemma 4.1 *For a constant equivalence relation $CC(S)$ with a total ordering $\succ$ on constants, the rewrite system R using the above construction represents a forest of trees (with each tree representing an equivalence class) in which the root is the least element in the equivalence class. Further, $\{c_i \rightarrow Find(c_i)\}$, for every non-root constant c_i, is canonical and the least among all rewrite systems representing $CC(S)$.*

Proof Suppose there is another rewrite system R' smaller than R. Consider the least rule $l \to r$ in R not in R'; similarly, $l' \to r'$ be the least rule in R' that is not in R. Either $l = l'$ implying that they differ in r, r', or $l \neq l'$. Consider the representatives of l and l' in $CC(S)$. If $l = l'$, then $Find(r) = Find(r')$ but $r \succ r'$ implies that $r \neq Find(r)$, a contradiction. Consider the case of $l \neq l'$; if $l' \succ l$, then $l' \to r' \succ l \to r$ but $l' \to r'$ is the least rule of R' not in R implying $l \to r \in R'$, which is a contradiction to $l \to r$ being the least rule in R not in R'. If $l \succ l'$, then R must include a rule to simplify l' which is smaller than $l \to r$ and its right side is $Find(l')$ by construction. $l' \to Find(l') \in R$ is smaller or equal to $l' \to r'$, which is a contradiction to the assumption that $l' \to r'$ is the least rule in R' not in R. ∎

4.2 Canonical Forms of Constant Horn Equations

Given $CC(S)$, which also induces a partition on constants in S, the canonical form of a Horn equation $H \Rightarrow a = b$ is either (i) `True`, (ii) an equation $a' = b'$ if each equality in H has canonical form `True` in $CC(S)$, and a', b' are respectively canonical forms of a, b in $CC(S)$, or (iii) a normalized Horn equation $H' \Rightarrow a' = b'$, i.e., $N'(H' \Rightarrow a' = b') = H' \Rightarrow a' = b'$, and H' is the conjunction of equalities in canonical forms including using equalities within H' to reduce each other and a', b' are canonical forms of a, b using $CC(S)$ as well as H'. For example $(a = b \wedge b = c) \Rightarrow a = d$, where each $a \succ b \succ c \succ d$ are in their own distinct equivalence classes in $CC(S)$, has its canonical form to be $(a = c \wedge b = c) \Rightarrow c = d$ obtained by using H to bring canonical forms to each of its equalities with respect to other equalities in H; for the concluding equality, the equalities in H are also used. Similarly, the canonical form of a constrained constant $H \Rightarrow a$ is $H' \Rightarrow a'$, where H' is the canonical form of H in $CC(S)$ and a' is the canonical form of a in $CC(S \cup H)$.

Given a set S of Horn equations, the canonical form of a constant a is the least constant in the ordering in the equivalence class of a in $CC(S)$. Similarly, the canonical form of an equality $a = b$ is `True` or $a' = b'$, where a', b' are the canonical forms of a, b, respectively in $CC(S)$.

The canonical form of a constrained constant $H \Rightarrow a$ is $H' \Rightarrow a'$, where a' is the canonical form a in of $CC(S \cup H)$ and for each $c = d \in H$, $c' = d'$ is the canonical form of $c = d$ in $CC(S \cup H - \{c = d\})$. This captures the condition that equalities in H are normalized with respect to each other as well as that the concluding equality is normalized with respect to H. The canonical form of $H \Rightarrow a = b$ is then $H' \Rightarrow a' = b'$ where $H' \Rightarrow a'$ and $H' \Rightarrow b'$ are respectively the canonical forms of $H \Rightarrow a$ and $H \Rightarrow b$.

For an S with constant Horn equations, we have:

Theorem 4.2 *$H \Rightarrow u = v \in CCC(S)$ iff the canonical form of $H \Rightarrow u$ and $H \Rightarrow v$ are identical, i.e., $u = v \in CCC(S \cup H)$.*

Two different Horn equation are thus equivalent even when they are not necessarily identical but they have the same canonical form. This could be because the same congruence relation can be presented in multiple ways. For example, $(a = b \wedge a = c) \Rightarrow (a = d)$ and $(a = b \wedge b = c) \Rightarrow b = d$ have the same canonical form $(a = c \wedge b = c) \Rightarrow c = d$ when $S = \emptyset$.

4.3 Extension to Horn Equations with Nonconstant Function Symbols

When S has uninterpreted symbols but no Horn equations, the congruence closure algorithm with extended signature outputs two components: (i) Constant equivalence represented as a forest of trees (equivalently a rewrite system as constructed in the previous section) and (ii) f-equations with distinct left side.

If the rewrite rules relating constants are reduced and used to rewrite those constants in f-equations and Horn equations, then the whole rewrite system is also reduced. Using a related Lemma 4.1 for constant equivalence, it follows that

Lemma 4.3 *Corresponding to a set S of ground equations and a total ordering $\succ$ on the extended signature, the above construction produces a canonical rewrite system that is reduced and the least among the presentations of the associated congruence closure relation.*

The canonical form of a flat term $f(c_1, \cdots, c_k)$ in $CCC(S)$ is the canonical form of the flat term $f(\overline{c_1}, \cdots, \overline{c_k})$, where $\overline{c_i}$ is the canonical form of c_i in $CCC(S)$; it is either a constant if the flat term is in its congruence class, otherwise, it is itself.

A simple example of a trivial Horn equation is: $(f(f(f(a)) = f(f(f(f(f(a))))) \wedge f(f(a)) = a) \Rightarrow f(f(f(f(a)))) = f(f(a))$ where S is $\emptyset$. Using flattening and introducing new symbols gives $A = \{f(a) = u_1, f(u_1) = u_2, f(u_2) = u_3, f(u_3) = u_4, f(u_4) = u_5\}$ and conjecture is the Horn equation $(u_3 = u_5 \wedge u_2 = a) \Rightarrow u_4 = u_2$. Its Horn closure $HC(A) = A \cup \{a = u_1 \Rightarrow u_1 = u_2, a = u_2 \Rightarrow u_1 = u_3, a = u_3 \Rightarrow u_1 = u_4, a = u_4 \Rightarrow u_1 = u_5, u_1 = u_2 \Rightarrow u_2 = u_3, u_1 = u_3 \Rightarrow u_2 = u_4, u_1 = u_4 \Rightarrow u_2 = u_5, u_2 = u_3 \Rightarrow u_3 = u_4, u_2 = u_4 \Rightarrow u_3 = u_5, u_3 = u_4 \Rightarrow u_4 = u_5\}$. It is easy to see that the condition $u_2 = a$ has a cascading effect making $u_1 = u_3 = u_5, u_2 = u_4 = a$ from which the conclusion of the conjecture follows.

For $CCC(S)$, the canonical form of a Horn equation $(\bigwedge_j s_1^j = s_2^j) \Rightarrow s = t$ is defined similar to that of a constant Horn equation discussed in the previous subsection. Let $H = \{s_1^j = s_2^j | 1 \leq j \leq k\}$ be the finite set of equalities in the hypothesis. The canonical form of an equality $s_1^j = s_2^j$ is $\overline{s_1^j} = \overline{s_2^j}$ where $\overline{s_1^j}, \overline{s_2^j}$ are canonical forms of $CCC(S \cup H - \{s_1^j = s_2^j\})$ of s_1^j, s_2^j, respectively. The canonical form of $H \Rightarrow s$ is $\{\bigwedge \overline{s_1}^j = \overline{s_2}^j\} \Rightarrow \overline{s}$, where $\overline{s}$ is the canonical form of s in $CCC(S \cup H)$. The canonical form of $(\bigwedge_j s_1^j = s_2^j) \Rightarrow s = t$ is `True` iff the canonical forms of $H \Rightarrow s$ and $H \Rightarrow t$ are identical; otherwise, it is $(\bigwedge \overline{s_1}^j = \overline{s_2}^j) \Rightarrow \overline{s} = \overline{t}$.

In the next subsection, rewriting of a constrained constant using Horn equations is defined so that it is terminating. A completion procedure is designed that checks whether Horn rules are locally confluent, and if not, add new Horn rules until a locally confluent system with the same $CCC(S)$ is generated. Such a canonical Horn rewrite system is a decision procedure for $CCC(S)$ by generating canonical forms for constrained constants as well as Horn equations.

5 Canonical Conditional Constant Rewrite System

We design a completion algorithm for generating a canonical conditional ground rewrite system from a finite set S of ground Horn equations. The canonical system serves as a decision procedure for membership of Horn equations in $CCC(S)$ as well as generating canonical forms for constrained terms and Horn equations. For this also, we first consider the case of constants only and later we extend the algorithm to the case when S also has function symbols, in the same way the above conditional congruence closure algorithm on ground Horn equations was generated from conditional constant closure algorithm.

There are multiple objectives for designing this algorithm beside generation of a canonical rewrite system, which is of independent interest: (i) Deleting redundant Horn equations, (ii) having a canonical representation for conditional congruence relation given a total ordering on function symbols and constants, and (iii) eliminating symbols, a construction that can be helpful in generating interpolants based on quantifier elimination[25, 27]. Further, associating a unique reduced canonical rewrite system with a conditional congruence closure of S for a fixed ordering would enable an easy linear time check for equivalence of conditional congruence relations presented by their respective reduced canonical rewrite systems.

5.1 Rewriting Using Conditional Rewrite Rules on Constants

We first define a rewrite relation on a constrained constant and then give an algorithm for generating a canonical rewrite system for $CCC(S)$ for a system of constant Horn equations.

Let S be a disjoint union of a finite set of equations, S_E, and Horn equations S_H on constants. We will start with a canonical rewrite system from S_E as defined in Subsection 4.1.1. This canonical system normalizes Horn equations in S_H by replacing each constant in it by the associated canonical form. Each Horn equation in S_H is also normalized after deleting `True, False` from a Horn equation; trivial Horn equations are deleted.

Recall that a normalized Horn equation is: $\bigwedge_j (u_1^j = v^j \wedge \cdots \wedge u_{k_j}^j = v^j) \Rightarrow (a = b)$, where j ranges over equivalence classes generated by the hypotheses, v^j is the least constant among all other constants in its equivalence class. a and b are distinct and are in canonical forms and are the least constants in equivalence classes generated after extending S_E with the constant equations in its hypothesis. The associated rewrite system with the above normalized Horn equation has its conclusion as a rewrite rule $a \to b$ if $a \succ b$ ($b \to a$ if $b \succ a$).

A conditional rewrite rule can apply at the term part of a constrained term and/or its constraints.

Definition 5.1 A normalized conditional rewrite rule $H_i \Rightarrow a_i \to b_i$ rewrites a normalized constrained term $H \Rightarrow a$ at a, denoted by $\to_m$, to $N(H \Rightarrow b_i) = H \Rightarrow b$ if (i) $P_{H_i} \subseteq P_H$ and (ii) $a_i = a, b = b_i \in P_H$.

A normalized conditional rewrite rule $H_i \Rightarrow a_i \to b_i$ rewrites normalized $H \Rightarrow a$ at H, denoted by $\to_h$, to $N(H' \Rightarrow a)$ if (i) $P_{H_i} \subseteq P_H$ and (ii) a_i is equivalent to some other constants different from itself in P_H. This ensures that $P_{H'} \neq P_H$ where $H' = H|_{a_i}^{b_i}$ modifies the equiv-

alence relation (partition) induced by H by replacing a everywhere by b thus capturing the equality of distinct constants. I.e., if a appears in H, then $H|_a^b$ does not have any occurrence of a and is an equivalence relation on fewer or smaller constants.

The rewrite relation $\rightarrow$ is then the union of $\rightarrow_m$ and $\rightarrow_h$.

Given that H is a conjunction of equalities, H on a single constant is the condition `True`, making $H \Rightarrow a = b$ to be unconditional equality $a = b$.

For example, consider the rewrite rules $1.a = b \Rightarrow c \rightarrow d$ and $2.c = d \Rightarrow u \rightarrow v$ where $c \succ d$ and $u \succ v$. A constrained term $(a = u \wedge c = d) \rightarrow u$ rewrites by the second rewrite rule $(\rightarrow_m)$ where $u \succ v$ to $(a = u \wedge c = d) \Rightarrow v$ at the conclusion. This can be further rewritten at the hypothesis to give $(a = v \wedge c = d) \Rightarrow v$.

A constrained term $(a = b \wedge c = d) \Rightarrow d$ is rewritten $(\rightarrow_h)$ by rule 1 to $a = b \Rightarrow d$ since c in the hypothesis reduces to d making $c = d$ to be trivial. A constrained term $(a = b \wedge c = e) \Rightarrow e$ assuming $c \succ d \succ e$, rewrites $(\rightarrow_h)$ by rule 1 to $(a = b \wedge d = e) \Rightarrow e$. However, if $c \succ e \succ d$, the result is $(a = b \wedge e = d) \Rightarrow d$.

The following properties about $H|_a^b$, where $a \succ b$, and H are used in the proofs later.

Lemma 5.2 *Let $H_1 = H|_a^b, a \succ b$, where H, H_1 are partitions induced by equivalence relations.*

1) *a does not appear in H_1.*

2) *$H - \{a\} \subseteq H$, where $H - \{a\}$ is the subpartition in which a has been deleted from its equivalence class (equivalently, the partition H on the set of constants excluding a).*

3) *If a does not appear in H_i, then $H_i \subseteq H \Rightarrow H_i \subseteq H_1$.*

4) *If $c = d \in H$ but $c = d \notin H_1$, then either c or d is a. In other words, if $c = d \in H$ and neither c or d is a, then $c = d \in H_1$ also.*

5) *$H_1|_a^d = H_1$. If $c \neq b, c \neq a$ $H_1|_c^d = (H|_c^d)|_a^b$. $(H|_a^b)|_b^d = (H|_a^d)|_b^d$.*

A total ordering $\succ$ on constants extends to constrained constants as well as finite sets of constant equations, Horn equations and the associated rewrite rules. Two multisets of constants $M_1 \succ M_2$ iff for each $y \in M_2 - M_1$, there is an $x \in M_1 - M_2$ such that $x \succ y$ $(\forall y \in M_2 - M_1 \exists x \in M_1 - M_2, x \succ y)$. An equation $a' = b' \succ a = b$ iff the multisets $\{\{a', b'\}\} \succ \{\{(a, b\}\}$. $H \Rightarrow a \succ H' \Rightarrow a'$ iff (i) $a \succ a'$ or (ii) $a = a' \wedge H \succ H'$, where H, H' are respectively equivalence classes represented as equations. Similarly, $H \Rightarrow a = b \succ H' \Rightarrow a' = b'$ iff $a = b \succ a' = b'$ or $(a = b) = (a' = b') \wedge H \succ H'$.

Lemma 5.3 *$\rightarrow$ is terminating.*

Proof $(H \Rightarrow a) \rightarrow (H' \Rightarrow a')$ using $H_i \Rightarrow a_i \rightarrow b_i$ gives that either (i) $H = H'$ or $H \succ H|_{a_i}^{b_i}$ and $a = a_i \in P_H \wedge a' = b_i \in P_{H'}$ implying $a \succ a'$, or (ii) $a = a' \wedge H \succ H|_{a_i}^{b_i}$. Since normalization of a constrained term and a Horn equation perserve $\succ$, $\rightarrow$ is terminating. ∎

Lemma 5.4 Soundness: *If $H_i \Rightarrow a_i \rightarrow b_i$ rewrites $H_j \Rightarrow a_j$ to $H'_j \Rightarrow b_j$, then $H_j \Rightarrow a_j = b_j$ as well as $H'_j \Rightarrow a_j = b_j$ are in $CCC(\{H_i \Rightarrow a_i \rightarrow b_i\})$.*

Proof Consider the case $H'_j \Rightarrow a_j = b_j \in CCC(\{H_i \Rightarrow a_i \Rightarrow b_i\})$; the proof of the other case is similar. $H'_j = N(H_j|^{b_j}_{a_j})$ is either $P_{H_j} = P_{H'_j}$ or $P_{H'_j} = P_{H_j|^{b_j}_{a_j}}$; we also have $P_{H_i} \subseteq P_{H_j}$ implying that $a_j = b_j \in CCC(\{H_i \Rightarrow a_i \rightarrow b_i\} \cup H'_j)$. ∎

The definition of rewriting $\rightarrow$ extends to a finite set S of conditional rules: $H \Rightarrow a \rightarrow H' \Rightarrow a'$ rewrites in one step by S iff there exists a rule $H_i \Rightarrow a_i \rightarrow b_i \in S$ such that $H \Rightarrow a \rightarrow H' \Rightarrow a'$ using the rule. Let the rewrite relations $\rightarrow^*, \rightarrow^+$ be the reflexive, transitive closure and transitive closure, respectively, of $\rightarrow$. $H \Rightarrow a$ is in **normal form** iff there is no rule in S which can rewrite it, i.e., for every rule of the form $H_i \Rightarrow a_i \rightarrow b_i \in S$, either (i) $P_{H_i} \not\subseteq P_H$, or (ii) $P_{H_i} \subseteq P_H$ but either $a_i = a \notin P_H$ or a_i does not appear in H.

For example, consider $S = \{1.\ a = b \Rightarrow c \rightarrow d,\ 2.\ c = d \Rightarrow u \rightarrow v\}$. $(a = b \wedge c = d) \Rightarrow u \rightarrow_h (a = b) \Rightarrow u$ using rule 1 (the normalization of $(a = b \wedge d = d) \Rightarrow u$); using rule 2, $(a = b \wedge c = d) \Rightarrow u \rightarrow_m (a = b \wedge c = d) \Rightarrow v$ which rewrites further using rule 1 to: $a = b \Rightarrow v$. Both $a = b \Rightarrow u$ as well as $a = b \Rightarrow v$ are in normal form. This establishes that $a = b \Rightarrow u = v \in CCC(S)$.

$H \Rightarrow a = b$ is rewritten in one step by S iff S rewrites either $H \Rightarrow a$ or $H \Rightarrow b$. $H \Rightarrow a = b$ is in normal form iff both $H \Rightarrow a$ and $H \Rightarrow b$ are in normal form. If $H \Rightarrow a = b$ is rewritten in many steps by S to a trivial Horn equation then $H \Rightarrow a = b$ is in $CCC(S)$.

Similarly, it easily follows that if $H_i \Rightarrow a_i \rightarrow b_i$ rewrites $H_j \Rightarrow a_j = b_j$ to $H_k \Rightarrow a_k = b_k$, then either a_j is rewritten to a_k with $b_j = b_k$ or b_j is rewritten to b_k with $a_j = a_k$. This is just to capture that the conclusion in an H-equation can be rewritten on its left side or right side.

Theorem 5.5 *The reflexive symmetric transitive closure of $\rightarrow$ induced by S is $CCC(S)$.*

Before presenting a proof, we illustrate using a simple example: Let $S = \{1.\ a = b \Rightarrow c \rightarrow d,\ 2.\ c = d \Rightarrow u \rightarrow v\}$. It is easy to see that $a = b \Rightarrow u = v \in CCC(S)$. A proof using $\leftrightarrow$ (the symmetric closure of $\rightarrow$) is: $(a = b \wedge c = d) \Rightarrow u \rightarrow a = b \Rightarrow u$ using rule 1 as well as to $(a = b \wedge c = d) \Rightarrow v$ using rule 2 which further rewrites using rule 1 to $a = b \Rightarrow v$. The crucial observation is the use of equalities in the hypotheses of constrained terms in a proof, they are rewritten using rules used in deriving additional equalities in $CCC(S \cup H)$, where H is the hypothesis of a conjecture being decided.

Proof In the forward direction, i.e., the reflexive symmetric transitive closure, $\leftrightarrow^*$ of $\rightarrow$ induced by S is contained in $CCC(S)$, is easier and follows from the soundness of rewriting and by induction in the number of steps relating two constrained terms in $\leftrightarrow^*$.

In the other direction, consider two constrained terms $H_1 \Rightarrow a_1,\ H_2 \Rightarrow a_2$ such that using $CC(S \cup H_1)$, $P_{H_1} = P_{H_2}$ as well as $a_1 = a_2$, a two sided rewriting sequence needs to be constructed using $\rightarrow$. H_1 makes hypotheses of conditional rewrite rules in S valid generating additional equalities in $CC(S \cup H_1)$ that are not in H_1 or $CC(S)$. If H_1 (or H_2) are extended to include such additional equalities, called H_a, rewrite rules needed in proofs $P_{H_1} = P_{H_2}$ as well

as $a_1 = a_2$ become applicable. Thus $H_1 \cup H_a \Rightarrow a_1$ can be rewritten using bi-directed rewriting at H_1, H_2, a_1, a_2 to $H_2 \cup H_a \Rightarrow a_2$ and then from both $H_1 \cup H_a \Rightarrow a_1$ as well as $H_2 \cup H_a \Rightarrow a_2$, H_a can be deleted using the rewrite rules that generated H_a from H_1, H_2, a_1, a_2. ∎

This is illustrated in an example above where $a = b \Rightarrow u \leftrightarrow^* a = b \Rightarrow v$ using $S = \{a = b \Rightarrow c = d,\ c = d \Rightarrow u = v\}$; equality $c = d$ is not in $CC(S)$ but is in $CC(S \cup \{a = b\})$. As shown above, $(a = b \wedge c = d) \Rightarrow u$ can be rewritten both to $a = b \Rightarrow u$ as well as $a = b \Rightarrow v$. From this, $a = b \Rightarrow u = v$ is deduced.

5.2 Local Confluence, Critical Pairs, and Completion

One way to ensure the confluence of a rewrite relation is to check whether it is locally confluent assuming it is terminating. The local confluence of a rewrite relation can be checked by generating superpositions among pairs of rewrite rules on which both rules are applicable and rewrite them in two possibly different ways (see [13] for more details).

There are three possible types of critical pairs among a pair of distinct normalized rules $H_i \Rightarrow a_i \rightarrow b_i$ and $H_j \Rightarrow a_j \rightarrow b_j$ in S due to interaction between $\rightarrow_h$ and $\rightarrow_m$.

A critical pair is called nontrivial iff it has a nontrivial Horn equation as a normal form; a critical pair is trivial if its normal form is a trivial Horn equation. A trivial critical pair is also called joinable.

If $(a_i, a_j) \in P_{H_i \cup H_j}$, then the superposition is: $N(H_i \cup H_j \Rightarrow \min(a_i, a_j))$ to enable the application of the two rules in possibly different ways: $\langle N((H_i \cup H_j) \Rightarrow b_i),\ N((H_i \cup H_j) \Rightarrow b_j)\rangle$ is a critical pair due to interaction between $\rightarrow_m$ and $\rightarrow_m$. For example, from $a = b \Rightarrow b \rightarrow u$ and $a = c \Rightarrow c \rightarrow v$, where $a \succ b \succ u, b \succ c, a \succ c \succ v$, the superposition is $N((a = b \wedge a = c) \Rightarrow \min(b, c))$ and the critical pair is: $(a = c \wedge b = c) \Rightarrow u$, $(a = c \wedge b = c) \Rightarrow v$ assuming $u \succ v$, $b \succ c$. The critical pair is not in normal form since both rules are applicable at their hypotheses.

If a_i occurs in H_j, then the superposition is: $N((H_j \cup H_i) \Rightarrow a_j)$ and the critical pair is: $\langle N((H_j|_{a_i}^{b_i} \cup H_i) \Rightarrow a_j,\ N((H_i \cup H_j) \Rightarrow b_j)\rangle$, due to interaction between $\rightarrow_h$ and $\rightarrow_m$. For example, from $a = b \Rightarrow c \rightarrow d$ and $c = d \Rightarrow u \rightarrow v$, we have: $N((a = b \wedge c = d) \Rightarrow u)$ as the superposition giving the critical pair: $\langle N((a = b \wedge d = d) \Rightarrow u),\ N((a = b \wedge c = d) \Rightarrow v)\rangle$. The second component can be further reduced using $a = b \Rightarrow c \rightarrow d$ to give $a = b \Rightarrow v$. It generates a new rule $a = b \rightarrow u \rightarrow v$.

The third case about interaction between $\rightarrow_h$ of $H_i \Rightarrow a_i \rightarrow b_i$ and $\rightarrow_h$ of $H_j \Rightarrow a_j \rightarrow b_j$ gives trivial critical pairs even when $i \neq j$. The superposition is: $N(H_i \cup H_j \Rightarrow a_i)$ and the critical pair is: $\langle N(H_i|_{a_i}^{b_i} \cup H_j \Rightarrow a_i),\ N(H_i \cup H_j|_{a_j}^{b_j} \Rightarrow a_i)\rangle$, which is trivial.

It is easy to check that the critical pairs are in the $CCC(S)$, which guarantees soundness.

Theorem 5.6 *A set S of Horn equations is locally confluent iff each pair of Horn equations has only trivial critical pairs.*

Proof Consider a constrained term $A \Rightarrow u$ which can possibly be rewritten in one step in two different ways: $A_1 \Rightarrow u_1$ using $H_i \Rightarrow a \rightarrow b$ as well as to $A_2 \Rightarrow u_2$ using $H_j \Rightarrow c \rightarrow d$.

There are multiple cases because of interactions between $\rightarrow_m$ and $\rightarrow_h$.

1) Both $H_i \Rightarrow a \rightarrow b$ and $H_j \Rightarrow c \rightarrow d$ rewrite u, which means $P_{H_i \cup H_j} \subseteq P_A = P_{A_1} = P_{A_2}$, $(a,u),(c,u),(u_1,b),(u_2,d) \in P_A$.

   The first way to construct a critical pair corresponding to this pair of rules is: $N(P_{H_i \cup H_j} \Rightarrow b = d)$ which is joinable. Rewrites used to show their joinability can be applied on $\langle A_1 \Rightarrow u_1, A_2 \Rightarrow u_2\rangle$ as they are for the joinability of the critical pair since $A = A_1 = A_2$ which includes $H_i \cup H_j$ and further $u_1 = b, u_2 = d \in A$.

2) $H_i \Rightarrow a \rightarrow b$ rewrites A whereas $H_j \Rightarrow c \rightarrow d$ rewrites u: We have $H_i \cup H_j \subseteq A$, $P_{A_1} = P_{A|_a^b}, P_{A_2} = P_A$ and $(u,u_1) \in P_{A_1}$ and $(u,c),(u_2,d) \in P_A$. $H_i \Rightarrow a \rightarrow b$ is applicable on $A_2 \Rightarrow u_2$.

   For $H_2 \Rightarrow c \rightarrow d$ to be applicable to $P_{A_1} \Rightarrow u_1$, it must be shown that $P_{H_j} \subseteq P_{A_1}$ as well as $(c,u_1) \in P_{A_1}$.

   There are two possibilities: (i) $a \notin H_j$: The joinability of the critical pair trivially follows since the second rule can still be applied on $A_1 \Rightarrow u_1$ and the first rule can be applied on $A_2 \Rightarrow u_2$ (using properties of the above lemma about relationship between A_1 and A). (ii) $a \in H_j$: Joinability of the second construction of the critical pair of these rules, $\langle N((H_j|_a^b \cup H_i) \Rightarrow c),\ \ N((H_i \cup H_j) \Rightarrow d)\rangle$ is used to show the joinability of $\langle A_1 \rightarrow u_1, A_2 \rightarrow u_2\rangle$ since $(H_j|_a^b \cup H_i) \subseteq A_1, H_i \cup H_j \subseteq A_2$; so all the rewrites can be repeated.

   Another way is to do case analysis on $H_j \subseteq A_1$. If $H_j \subseteq A_1$, then $H_j \Rightarrow c \rightarrow d$ is applicable if $(c,u_1) \in P_{A_1}$, in which case the joinability follows. If $(c,u_1) \notin P_{A_1}$, then by the lemma, $a = c \in P_A \vee u = c \in P_A$; in either case, the joinability follows from the joinability of the critical pairs.

   If $H_j \not\subseteq A_1$, then $a \in H_j$; the joinability follows from the joinability of critical pairs.

3) Both $H_i \Rightarrow a \rightarrow b$ and $H_j \Rightarrow c \rightarrow d$ rewrite A: $H_i \cup H_j \subseteq A$ and $a, c \in A$. Further $N(A|_a^b) = A_1, N(A|_c^d) = A_2$, $(u,u_1) \in A_1, (u,u_2) \in A_2$. u_1 is the same as u unless $(u,a) \in A_1$ in which case $u_1 = b$; similarly, u_2 is the same as u unless $(u,c) \in A_2$ in which case $u_2 = d$.

   A_1 does not have a so unless $c = a$, $c \in A_1$. The case of $c = a$ is considered later. If $a \notin H_j$, then $H_j \subseteq A_1$. $H_j \cup \{c \rightarrow d\}$ can be applied in that case on $A_1 \Rightarrow u_1$ in its hypothesis, giving $N(A_1|_c^d \Rightarrow u_1)$. Similarly, $H_i \cup \{a \rightarrow b\}$ can be applied on $A_2 \Rightarrow u_2$ assuming $a \neq c, c \notin H_i$ to give $A_2|_a^b \Rightarrow u_2$, which gives the joinability.

   If $a \in H_j$, then the joinability of critical pair gives the joinability of $\langle N(H_J|_a^b \cup H_i \Rightarrow c), N(H_i \cup H_j \Rightarrow d)\rangle$, from which joinability of the two sides follows. This argument works also if $c \in H_i$.

   In case $a = c$, then, the joinability of the critical pairs is used to show the joinability. ∎

If S is not locally confluent implying that there is at least one critical pair that has a non-trivial normal form, then its normal form is added as an additional Horn equation to S. This process continues until the resulting Horn rewrite rules have only trivial critical pairs. Upon termination, this completion algorithm generates a locally confluent Horn constant rewrite system. The completion procedure terminates since there are only finitely many constants and hence only finitely many new Horn equations can be generated. The result of the completion procedure is a finite set of locally normalized conditional rules which generate the same congruence relation as the input.

Theorem 5.7 *Given a canonical rewrite system R that is reduced (i.e., no left side of a rewrite rule can be further reduced using other rules), where $U \succ NU$, let R_{NU} be the subset of R which does not include any constants from U. Then, R_{NU} is a canonical rewrite system for $CCC(R_{NU})$, the subset of conditional congruence closure of $CCC(R)$ in which conditional equations do not have a symbol from U.*

Proof It is first proved that R_{NU} is canonical and then that R_{NU} is a decision procedure for $CCC(R)_{NU}$.

Since every rule in R_{NU} is from a terminating set R of rules, it is terminating. Every critical pair from pair of rules in R_{NU} is also a critical pair from the same pair of rules in R. Its joinability follows from the fact that all uncommon symbols in U are bigger in the ordering that other symbols, so no rule from $R - R_{NU}$ is used to show joinability using R.

To show that R_{NU} is a decision procedure for $CCC(R)_{NU}$, consider a Horn equation that cannot be decided in the same way using R_{NU} as using R. That would imply that there is a rule from $R - R_{NU}$ needed to show its decidability but such a rule cannot be applied since it includes a symbol from U which is bigger than all symbols appearing in the conjecture. Hence a contradiction. ∎

The above theorem is particularly useful in interpolant generation based on quantifier elimination since uncommon symbols needed to be eliminated[27, 31].

5.3 Examples

Consider, for example, two Horn equations 1. $a = b \Rightarrow c = d$ and 2. $c = d \Rightarrow a = b$. Consider an ordering $a \succ b \succ c \succ d$. Local normalization does not change the Horn equations which are oriented as 1. $a = b \Rightarrow c \rightarrow d$, 2. $c = d \Rightarrow a \rightarrow b$. Both 1 and 2 are already in normal forms. A possible critical pair between 1 and 2 that rewrites a in the hypothesis of rule 1 is $\langle N((c = d \wedge b = b) \Rightarrow c), N((a = b \wedge c = d) \Rightarrow d)\rangle$; its second component further reduces by rule 2 to $N((b = b \wedge c = d) \Rightarrow d)$ giving $N(c = d \Rightarrow c = d)$ which is trivial. Similarly, rule 1 rewrites c in the hypothesis of 2 also giving a trivial critical pair. Since all critical pairs are trivial, $1, 2$ constitutes a canonical rewrite system.

Suppose we slightly modify the second H-equation to 3. $b = d \Rightarrow a = b$. Local normalization gives $3'$. $b = d \Rightarrow a = d$ giving the rule $3'$. $b = d \Rightarrow a \rightarrow d$. The critical pair construction that rewrites a in 1 gives 4. $b = d \Rightarrow c = d$ which is oriented as 4. $b = d \Rightarrow c \rightarrow d$. It can be

proved that $b = d \Rightarrow c = d \in CCC(\{1, 3\})$. Critical pairs among $1, 3', 4$ are all trivial declaring $\{1', 3, 4\}$ as a canonical Horn rewrite system. A Horn-equation $b = d \Rightarrow a = c$ can be easily proved by rewriting it using 4 and $3'$.

Consider a Horn equation $b = c \Rightarrow a = b$ which locally normalizes and orients as a rewrite rule 5. $b = c \Rightarrow a \rightarrow c$. A critical pair between 5 and 1 gives 6. $b = c \Rightarrow c \rightarrow d$. 6 rewrites 5 to $5'$. $b = c \Rightarrow a \rightarrow d$. A critical pair between 1 and 5' gives $(b = c \wedge b = d) \Rightarrow c = d$ which is trivial. Thus $\{1, 5', 6\}$ is a canonical rewrite system.

The conjecture $a = c \Rightarrow b = d$ does not follow from $\{1, 5', 6\}$ (it is already in normal form).

5.4 Termination and Complexity

Termination of the completion algorithm trivially follows from the fact that there can be only be finitely many normalized Horn equations and hence Horn rewrite rules on a finite number of constants.

During the completion algorithm, all Horn rules are kept in normal form, which implies a new rule is added only after it has been normalized, i.e., for a new rule $H \Rightarrow a = b$ to be added, no other rule can have a as the left side of a conclusion unless H is a refined partition of existing rules with a as the left side of their conclusions. Thus, for any pair appearing as the left side of a Horn rule, other Horn rules with the same pair or a pair sharing constants in their conclusions, must have noncomparable (using subset ordering) or smaller partitions as conditions.

A very crude bound on the size of a canonical Horn rewrite system can be obtained by considering all possible Horn equations that can be constructed using n constants in the input to the completion algorithm. There cannot be more than $O(n^2)$ unconditional rewrite rules on n constants. However, this bound can be substantially improved by analyzing partitions on n and finding the one with the biggest size (i.e., number of equivalences classes) by optimizing the product of number of elements in an equivalence class times the number of equivalence classes.

Corresponding to an unconditional rewrite rule, there can be multiple nonredundant Horn rules with the same concluding equality but different hypotheses. The number of such rules can be bounded by the maximum number of partitions, the Bell numbers with the generating function $e^{(e^n-1)}$, where n is the number of nonequivalent constants on which Horn rules are being expressed. However, it suffices to consider partitions that cannot be compared with each other, i.e., for a set with $\{a, b, c\}$, for example, there are 5 possible partitions but noncomparable by subset ordering are only 3—$\{\{[a, b], [c]\}, \{[a, c], [b]\}, \{[a], [b, c]\}$; the other two $\{[a], [b], [c]\}$ and $\{[a, b, c]\}$ are respectively the subset and superset of each of the above partitions. Thus, only 3 nonredundant Horn rules are possible: $a = b \Rightarrow u \rightarrow v, a = c \Rightarrow u \rightarrow v, b = c \Rightarrow u \rightarrow v$, neither of which can rewrite the other. The worst case complexity of completion on Horn constant rewrite systems needs further detailed investigation.

5.5 Deciding Membership in $CCC(S)$

If S has an associated canonical rewrite system R, then $H \Rightarrow a = b$ can be decided by computing its normal form using R. If the canonical form $H \Rightarrow a = b$ normalizes to `True`, then it is in $CCC(S)$, i.e., follows from S; otherwise it does not. Recall that a single step rewrites either the constant pair (a, b) in the conclusion to smaller constants, or if the conclusion does not change, then H is rewritten to a smaller partition with a constant rewritten to a smaller constant. The canonical form can be computed in $O(k^2)$ steps, where k is the number of equivalence classes in $CC(S)$.

If a canonical rewrite system for S is not available, then deciding membership in $CCC(S)$ by generating a canonical system from S is expensive unless the cost of generating the canonical rewrite system is amortized over several membership queries. It is better in that case to avoid generating a canonical system and use the Deduction theorem to provisionally extend $CCC(S)$ with H and checking for equivalence of a and b in the extended system.

5.6 A Canonical System for Ground Horn Equations with Function Symbols

Given a set S of Horn equations expressed using function symbols, a completion procedure for generating a canonical rewrite system is in two stages: (i) Given a finite set S of Horn equations $H \Rightarrow s = t$, all ground terms in S are flattened using new constant symbols to stand for subterms appearing in S. This step gives a finite set FE_S of f-equations $f(c_1, \cdots, c_k) = d$, a finite set $CC(S_C)$ of constant equations on the extended signature, and a finite set HE_S of Horn equations purely expressed using constant symbols. FE_S is the only set of equations with nonconstant function symbols which are managed using function signatures as in [2].

The above completion algorithm is applied on $CC(S_C) \cup HE_S$ giving priority to constant equations since normalized Horn equations are expressed using canonical forms generated from constant equivalence closure. The result is a canonical Horn constant rewrite rules.

The f-equations in FE_S are rewritten, replacing constants in them by their canonical forms. If the left sides of two f-equations become identical, implied constant equalities are generated and a new rewrite rule corresponding to it is added. There is thus interplay between the two stages.

From two different f-equations but with identical outermost symbol, new constant Horn equations are also generated. A data structure keeping track of constants appearing in Horn equations updates them as canonical forms of constants dynamically change. If the hypothesis of a Horn equation becomes `True`, then an implied equality is generated which is eagerly processed to update the constant congruence relation.

The result of the algorithm is a finite set of R_S of rewrite rules which is partitioned as: (i) R_F, a finite set of distinct rules (equations) in which the left side is a flat term and the right side is a constant such that all constants are in canonical forms and no two left sides are identical, (ii) $CC(S_C)$, a finite set of constant rules abstractly specifying the forest of trees representing equivalence classes; there is a rule for every nonroot constant to its root, and (iii)

R_{HE}, a finite set of normalized Horn equations specified using root constants (canonical forms) which is locally confluent.

A completion procedure so designed is easier to understand, implement as well as efficient in contrast to the one directly on ground conditional Horn equations with function symbols. Furthermore, its termination follows from the termination of the completion of constant Horn rewrite rules. The interaction with f-equations is factored out; the equality of two f-equation with identical left sides is managed through the signature computation data structure that ensures uniqueness of the signature of all equivalent function terms.

The output of the completion procedure is on the extended signature and is a terminating, confluent rewrite system. If desired, new constants not in the input S can be replaced by the function terms they stand for, leading to a locally confluent Horn rewrite system since the result need not be terminating any more.

Theorem 5.8 *R_S is a canonical rewrite system such that $Orig(CCC(R_S)) = CCC(S)$, where Orig replaces all new constants by their corresponding flat terms to obtain Horn equations in the original function symbols of S.*

R_S is a decision procedure for S to determine whether a Horn equation is in $CCC(S)$.

Given a Horn query $H \Rightarrow s = t$, constants in it are replaced by their canonical forms first, then R_f is applied to generate a pure Horn equation in constants or a Horn equations with function symbols which cannot be eliminated; remaining function terms are flattened giving rise to the conjecture flattened to be a Horn constant equation. The Horn equation is then normalized using R_H; if its normal form is `True`, then it is in $CCC(S)$; otherwise it is not.

5.7 Example

We illustrate the above procedure first on a simple example on constants only.

Example 5.1 Let $S = \{1.\ a = e,\ \ 2.\ (a = b \wedge c = e) \Rightarrow c = d\}$; assume a total ordering $a \succ b \succ c \succ d \succ e$. Congruence relation induced by constant equations generates one nontrivial congruence class $\{a, e\}$ with e as its canonical form; all other constants are their own canonical forms. Normalization of H-equation generates $2'.\ (b = e \wedge c = e) \Rightarrow (d = e)$.

To check whether the rules from S are locally confluent, they are oriented as: $\{1.\ a \rightarrow e,\ 2'.\ (b = e \wedge c = e) \Rightarrow d \rightarrow e\}$. There is no critical pair since the left hand side of the unconditional rule does not appear in $2'$ and the left side of the conclusion in $2'$ does not appear in 1. This implies that the above two rules constitute a canonical rewrite system. Neither of the equations $d = e$ and $a = d$ follows from S. A Horn equation $a = b \Rightarrow b = e$ reduces using 1 to $b = e \Rightarrow b = e$ which normalizes to `True`.

Consider a Horn equation with function symbols, $(f(a) = f(b) \wedge g(f(d), c) = g(f(a), b)) \Rightarrow h(u, v) = h(v, u)$. Flattening it gives: $\{f(a) = c_1, f(b) = c_2, f(d) = c_3, g(c_3, c) = c_4, g(c_1, b) = c_5, h(u, v) = c_6, h(v, u) = c_7\}$ with subscripted c's as new constants. The original Horn equation becomes constant Horn equation $(c_1 = c_2 \wedge c_4 = c_5) \Rightarrow c_6 = c_7$. Consider a total ordering on constants: $a \succ b \succ c \succ d \succ u \succ v \succ c_1 \succ c_2 \succ c_3 \succ c_4 \succ c_5 \succ c_6 \succ c_7$. There are no constant

equivalences; so completion on Horn constant equations gives: $(c_1 = c_2 \wedge c_4 = c_5) \Rightarrow c_6 \rightarrow c_7$.

From f-equations, Horn equations are generated; from equations with f as the outermost symbol, $\{1.\ a = b \Rightarrow c_1 \rightarrow c_2,\ 2.\ a = d \Rightarrow c_1 \rightarrow c_3,\ 3.\ b = d \Rightarrow c_2 \rightarrow c_3\}$, and similarly from equations with the outermost symbol g, h: 4. $(c_1 = c_3 \wedge b = c) \Rightarrow c_4 \rightarrow c_5$, 5. $(u = v) \Rightarrow c_6 \rightarrow c_7$.

Completion is used to generate additional Horn rules. Rules 1 and 2 give a trivial critical pair: $a = b = d \Rightarrow c_2 = c_3$ since it rewrites to `True` by rule 3. But rules 1 and 2 with rule 4 give additional rules: $\{6.\ (a = b = c \wedge c_2 = c_3) \Rightarrow c_4 \rightarrow c_5,\ 7.\ (a = d \wedge b = c) \Rightarrow c_4 \rightarrow c_5\}$ whereas rules 3 and 6 generate a trivial critical pair. Rules 1–7 constitute a canonical Horn system.

The Horn equation $(c_1 = c_2 \wedge c_4 = c_5) \Rightarrow c_6 = c_7$ does not reduce to `True`, implies that the original Horn equation by itself is not valid. And, the above result is a canonical rewrite system generated from the input Horn equation.

6 Functions with Properties

In the preceding sections, all function symbols are assumed to be uninterpreted, i.e., they are assumed to have no properties or even if they have properties, such properties are not considered in computing conditional congruence closure or generating a canonical Horn rewrite system.

The proposed framework further generalizes to conditional equations with interpreted symbols with properties. Such properties may include a combination of commutativity, nilpotency, identity, idempotency, and other permutative properties. This framework also generalizes to a larger family of interpreted symbols including those having both associativity and commutativity properties as well as the case when interpreted symbols are characterized more generally by a first-order quantifier-free theory with a decision procedure; this will be subject of another paper.

A crucial property employed in the congruence closure proposed in [12] is that every ground term has a canonical form (this is equivalent to the requirement that ground terms have a unique signature which is common across all ground terms in the same congruence class). The unique signature requirement is adapted below to consider equivalences due to the properties of interpreted symbols.

The proposed framework elegantly factors out four separate parts of the algorithms: (i) Constant equivalence closure, (ii) Horn equations leading to implied equalities and their interplay with step (i) to generate the conditional constant equivalence closure, (iii) ground equations relating (a) flat uninterpreted term to a constant, and/or (b) interpreted terms belonging to a common theory that may involve a single function symbol or a collection of function symbols. The critical step in the generalization is generating canonical forms of signatures with respect to a finite set of equations relating interpreted terms. Unlike [23, 28], these extensions turn out to be quite simple, easy to understand and prove correct.

Let F_i be the subset of interpreted symbols in F. Let $T(F_i \cup C)$ be the ground terms generated using all constants in C, both from the original input as well as any new constants intro-

duced during purification and flattening both for uninterpreted as well as interpreted ground terms. A nonconstant term with an interpreted function symbols is called interpreted. A term with an uninterpreted symbol is called uninterpreted. Nonflat mixed terms are purified by introducing new constants for interpreted subterms belonging to a single theory.

The input to the algorithms below consists of (i) equations on constants, represented by a forest of trees implying that constant equivalence closure algorithm has already been applied, (ii) normalized Horn equations on canonical constants from (i), (iii) f-equations with flat terms for uninterpreted function symbols such that no two f-terms are identical thus assuming that implied equalities have already been generated from uninterpreted terms and processed, and (iv) f-equations relating interpreted terms belonging to each subtheory. Let the total size of the input (sum of the size of all terms in the input) be n.

As in the case of congruence closure, canonical forms and the associated rewrite systems are generated by imposing a total ordering on symbols. Nonconstant function symbols are bigger in the ordering than constant symbols. New constants are bigger in the ordering than original constants and interpreted constant symbols.

Below, we discuss interpreted symbols with simple theories for which signature computation can incorporate the properties of interpreted symbols. Richer theories including associative-commutativity, associativity and an arbitrary equational theory with an associated rewrite system will be subject of another paper because of lack of space.

6.1 Conditional Congruence Closure with Commutative Functions

Let FC be a nonempty subset of binary function symbols in F_i which are commutative: $f(x, y) = f(y, x), f \in FC$ for every x, y. Thus any two distinct flat terms with f as the outermost symbol but the same arguments in different order, say $f(c, d)$ and $f(d, c)$, are equivalent. Using a total ordering on constants, a unique signature is associated with flat terms with commutative functions by sorting its constants using the ordering: Signature of both $f(c, d)$ and $f(d, c)$ is $f(c, d)$ if $d \succ c$ is in the total ordering, for instance. While processing the input for flattening, interpreted subterms are put in normal form using the ordering on constant arguments, thus avoiding having to introduce unnecessary new constants. Canonical forms are generated as before except that for flat terms with commutative function symbol, constants are sorted. That is the only change in the signature computation. The rest of the algorithm does not change.

From complexity perspective, keeping signatures of interpreted terms in their normal form takes at worst n additional steps where n is the input size since the arguments to a commutative symbol may need swapping. Thus, conditional congruence closure with commutative function symbols is of the same complexity as that for uninterpreted symbols since flat terms are always kept in their normal forms with arguments in ascending order. If an argument to a flat term changes due to additional constant equalities, it needs to be brought to normal form again; however that does not add to the complexity since every such normalization takes constant time and the number of such operations is bounded by the input size. Further, this additional

cost gets absorbed in the overall complexity.

6.1.1 Horn Closure

For generating Horn equations from a pair of flat commutative terms in normal form: $f(c,d) = u, f(e,f) = v$, generate two Horn equations $(c = e \wedge d = f) \Rightarrow u = v$, $(c = f \wedge d = e) \Rightarrow u = v$. Equivalently an f-equation $f(c,d) = u$ with commutative f also gives another equivalent f-equation $f(d,c) = u$ on which the HC closure for the uninterpreted case gives equivalent results.

6.1.2 Generating a Canonical Rewrite System for $CCC(S)$

Completion algorithm for conditional congruence closure discussed in Subsection 5.2 extends in a straightforward way by keeping interpreted flat terms in normal forms.

6.2 Conditional Congruence Closure with Idempotent Functions

Let FI be a subset of binary function symbols in F_i that have idempotency property, i.e, a universal property $f(x,x) = x, f \in FI$ for every x. Signatures are normalized by replacing flat terms with identical arguments, e.g., replace $f(a,a)$ by a.

From complexity perspective, keeping signatures of interpreted terms in their normal form requires n additional steps, where n is the input size. So the overall complexity of the congruence closure algorithm does not change.

6.2.1 Horn Closure

For idempotent function symbols, the Horn closure generates an additional Horn equation from each interpreted flat term equation $f(c,d) = u$ with idempotent f: $c = d \Rightarrow d = u$. The rest of the construction is similar as in the uninterpreted case.

The above construction of Horn closure can be viewed as a special kind of critical pair generation from flat interpreted ground terms if the universal properties of an interpreted symbol(s) can be oriented into a canonical rewrite system. In general, given an interpreted ground equality: $g_1 = g_2$ oriented from left to right, it generate a Horn conjecture from a universal property $L \rightarrow R$ expressed as a rewrite rule iff **conditional matching**, expressed as constant equality conditions, of a subterm of L with g_1 instantiates all variables in R. The resulting Horn equation is: $H \Rightarrow \sigma(L)[g_1 \leftarrow g_2] = \sigma(R)$, where σ is a substitution for variables in L so that g_1 is a subterm in $\sigma(L)$ under the conditions in H.

To apply this construction on $f(c,d) \rightarrow u$ with $f(x,x) \rightarrow x$, $\sigma = \{x \leftarrow d\}, H = \{c = d\}$ leading to $c = d \Rightarrow d = u$.

6.2.2 Generating a Canonical Rewrite System for $CCC(S)$

Completion algorithm for conditional congruence closure discussed in Subsection 5.2 extends in a straightforward way by keeping interpreted flat terms in normal forms (equivalently rewriting using a meta nonground rule $f(x,x) \rightarrow x$ for idempotent symbol f).

Consider a simple example: $S = \{a = b \Rightarrow u = 0,\ f(a,b) = u\}$ with idempotent f. Without the second equation, the first Horn equation is a canonical rewrite system as there are no critical pairs.

Horn closure generates from the second equation: $a = b \Rightarrow u = b$ assuming $u \succ a \succ b \succ 0$. The result is $CCC(S) = \{a = b \Rightarrow u \rightarrow 0,\ f(a,b) \rightarrow u, a = b \Rightarrow u \rightarrow b\}$. A Horn equation $a = b \Rightarrow a = u$ in $CCC(S)$: It is proved directly by rewriting it using the third rule. The Horn equation $a = b \Rightarrow b = 0$ is also in $CCC(S)$ but it cannot be directly proved. However, using a canonical rewrite system generated from $CCC(S)$, it can be proved directly by rewriting. In fact, $a = b \Rightarrow b = 0$ is obtained by the critical pair between the first and third rules.

From $CCC(S)$: $\{1.\ a = b \Rightarrow u \rightarrow 0\ 2.\ f(a,b) \rightarrow u,\ 3.\ a = b \Rightarrow u \rightarrow b\}$, the third rule rewrites to give: $3'.\ a = b \Rightarrow b = 0$ which replaces 3. $\{1, 2, 3'\}$ is a canonical system.

6.3 Conditional Congruence Closure with Functions with $f(x,x) = e$

Let FN be a subset of binary function symbols in F that have nilpotency property, i.e., $f(x,x) = e, f \in FN$ for all x where e is the identity. Much like interpreted symbols with the idempotency property, signature computation of these interpreted function symbols is straightforward and efficient. An interpreted flat term with a nilpotent function symbol having identical arguments, say $f(c,c)$ is replaced by e. This step on a single f-equation can be performed in constant time (by maintaining a data structure in which the two arguments of f in different flat terms are maintained as a (hash) table). The overall complexity thus increases only by $O(n)$ steps, where n is the input size.

6.3.1 Horn Closure

From $f(x,x) \rightarrow e$ and a ground equality $f(c,d) \rightarrow u$, Horn closure construction using conditional matching gives a Horn equation $c = d \Rightarrow u = e$ using $\sigma(x) = \{x \leftarrow d\}, H = \{c = d\}$. This construction is applied on every f-equation with a nilpotent symbol.

6.3.2 Generating a Canonical Rewrite System for $CCC(S)$

Completion algorithm for conditional congruence closure also extends in a straightforward way by keeping interpreted flat terms in normal form and generating additional Horn equations as was done earlier for idempotent functions and functions with identities.

In the example in the previous subsection, assume f is also nilpotent along with being idempotent. An additional Horn equation is generated by Horn closure, giving $CCC(S) = \{a = b \Rightarrow u = 0,\ f(a,b) = u, a = b \Rightarrow u = b, a = b \Rightarrow b = 0\}$. A canonical rewrite system generated from it first rewrites rules using other rules giving: $S' = \{a = b \Rightarrow u \rightarrow 0,\ f(a,b) \rightarrow u, a = b \Rightarrow b \rightarrow 0\}$. Nilpotency property does not give any additional rule in the canonical system. It is easy to verify that $CCC(S) = CCC(S - \{a = b \Rightarrow u = 0\})$ if f is both nilpotent as well as idempotent.

6.4 Conditional Congruence Closure with Functions with Identity

Let $F0$ be a subset of binary function symbols in F_i with identities, i.e, $f(x, re) = x, f \in F0$ for all x in case of f having a right identity re. Similarly $f(le, x) = x, f \in F0$ in case f has a left identity le. As in the case of idempotent or nilpotent function symbols, signatures of interpreted terms are normalized using the above identities as rewrite-rules. The rest of the

algorithm remains the same. Keeping signatures of interpreted terms in their normal form takes at worst $O(n)$ additional operations, where n is the input size.

6.4.1 Horn Closure

Horn closure is constructed using conditional matching of interpreted terms with the rules for identities. From $f(c,d) \to u$ and an identity rule $f(x,re) \to x$, a Horn equation $d = re \Rightarrow c = u$ is generated; similarly, from an identity rule $f(le,x) \to x$, a Horn equation $c = le \Rightarrow d = u$.

6.4.2 Generating a Canonical Rewrite System for $CCC(S)$

Completion algorithm for conditional congruence closure also extends in a straightforward way by keeping interpreted flat terms in normal form and generating additional Horn equations. The complexity thus does not change.

Consider $S = \{a = b \Rightarrow u = 0,\ f(a,b) = u\}$, where 0 is both the left and right identity of f (without f being idempotent or nilpotent). Without the second equation, the first Horn equation is a canonical rewrite system as there are no critical pairs. Two Horn equations are generated from $f(a,b) = u$: $\{a = 0 \Rightarrow u = b,\ b = 0 \Rightarrow u = a\}$. Running completion on the three Horn rules gives trivial critical pairs $\{(a = b \wedge a = 0) \Rightarrow b = 0,\ (a = b \wedge b = 0) \Rightarrow a = 0\}$. So, the rewrite system consisting of three Horn rules and one unconditional rule is a canonical rewrite system: $R_S = \{a = b \Rightarrow u \to 0,\ f(a,b) \to u,\ a = 0 \Rightarrow u \to b,\ b = 0 \Rightarrow u \to a\}$ under the ordering $u \succ a$ as well as $u \succ b$.

Is $u = 0 \Rightarrow a = b$ in $CCC(S)$? The answer is no: Assuming the hypothesis $u = 0$ simplifies S to $\{f(a,b) = 0, u = 0\}$ from which $a = b$ does not follow. Adding the hypothesis to R_S also does not help: $\{a = b \Rightarrow 0 = 0,\ f(a,b) \to 0,\ a = 0 \Rightarrow b \to 0,\ b = 0 \Rightarrow a \to 0\}$ and $a = b$ cannot be proved unconditionally. Using R_S, it is easy to see that the conjecture is in normal form since it cannot be rewritten by any rule because of its condition $u = 0$.

If the second equation in S is replaced by $f(a,b) = b$ to give S', then, Horn equations $b = 0 \Rightarrow a = 0$ as well as $a = 0 \Rightarrow b = b$ are generated from $f(a,b) = b$ in its Horn closure; the second Horn equation is deleted since it is trivial. Critical pair computation on $a = b \Rightarrow u \to 0, b = 0 \Rightarrow a \to 0$ gives a new rule $b = 0 \Rightarrow u = 0$. The set $\{a = b \Rightarrow u \to 0, b = 0 \Rightarrow u \to 0, b = 0 \Rightarrow a \to 0, f(a,b) \to b\}$ is a canonical rewrite system. It is easy to double check that both $b = 0 \Rightarrow u = 0, b = 0 \Rightarrow a = 0$ are in $CCC(S')$. Further, $b = 0 \Rightarrow a = u$ is also in $CCC(S)$: Adding the hypothesis $b = 0$ to S' generates $a = 0 \Rightarrow u = 0, a = 0$ leading to $u = 0, a = 0$. By rewriting $b = 0 \Rightarrow a = u$ using the canonical rewrite system of S', both sides of the conclusion rewrite to 0.

As the reader would have noticed, a canonical rewrite system for interpreted terms with the above properties is expressed using Horn equations on existing constants. No additional new constants need to be introduced. This changes however for richer theories including interpreted symbols with both associative and commutative symbols where new constant symbols have to be introduced to stand for new interpreted terms generated.

To conclude, consider two f-equations $f(a,b) = c$ and $f(b,a) = d$ in which f is commutative, idempotent, nilpotent as well as has both left and right identity. Because of commutativity,

a constant equality $c = d$ is generated. Assuming $a \succ b$ and $c \succ d$ in the ordering, the above equations are brought into normal form resulting in $f(a, b) = d, c \rightarrow d$. Horn closure gives $a = b \Rightarrow b = d$ due to idempotency, $a = b \Rightarrow d = e$ due to nilpotency where e is the left and right identity of f, $a = e \Rightarrow b = d$ as well as $b = e \Rightarrow a = d$ due to the identity. The rewrite system is $\{f(a, b) \rightarrow d, c \rightarrow d, a = b \Rightarrow b \rightarrow e, a = b \Rightarrow d \rightarrow e, a = e \Rightarrow b \rightarrow d, b = e \Rightarrow a \rightarrow d\}$. Equivalent Horn equations with some redundancy would have been generated separately from $f(a, b) = c$ and $f(b, a) = d$ without considering the commutativity of f but after identifying $c = d$ due to the commutativity of f would result in simplifying those Horn equations to the set of Horn equations discussed earlier.

7 Conclusion

The framework first presented in [12] for generating congruence closure using flattening and abstracting congruence closure problems on constants is generalized to generating conditional congruence closure both in case of uninterpreted symbols as well as interpreted symbols with properties including commutativity, idempotency, identity and nilpotency and their combination. The conditional congruence closure algorithm for uninterpreted symbols is shown to be $O(n * \log(n))$, where n is the input size, the same complexity as the unconditional congruence closure. Further, for interpreted symbols that are commutative, idempotent, nilpotent, and with identities, the congruence closure algorithms are also of the same complexity.

A completion algorithm for generating a canonical (conditional) Horn rewrite system from a finite set of ground conditional equations is also presented. The complexity of generating a canonical rewrite system for conditional ground equations is however much higher than that of generating conditional congruence closure. This is in contrast to the congruence closure algorithm in [12] which also generates a canonical rewrite system on the extended signature for free. Consequently, deciding membership in a congruence closure is of the same complexity whether it is checked using normalization of a query (without having to do flattening) or flattening and extended signatures. In case the rewrite system on the original signature is also terminating, there is one to one correspondence between the two methods. If a total ordering on extended signature is already determined, then termination is not ensured as illustrated above, in which case, normalization must be performed using cycle checking which may be more expensive than using the extended signature.

Deciding membership in a conditional congruence closure is likely to be of much lower complexity if the hypotheses in a query are first used to extend the conditional congruence closure and then checking whether the conclusion follows in the provisional conditional congruence closure. In contrast, completion based approach for deciding membership is of higher complexity because a canonical rewrite system can be much larger than the original system, This additional complexity can be justified if numerous Horn equations need to be decided; this tradeoff is not different from trade off observed when using completion to generate a decision procedure versus directly deciding a few queries.

We believe the proposed framework which can deal with interpreted symbols, predicates as well as uninterpreted symbols has considerable promise for developing algorithms for other inference problems expressed using ground terms. The perspective presented in the paper is also likely to be useful in developing decision procedure for non-Horn ground clauses using instantiation based approaches to first-order theorem proving.

Acknowledgements I would like to thank Jose Castellanos Joo for comments and implementing parts of the algorithm in the context of interpolant generation. I also thank the referees for numerous suggestions for improving the presentation.

References

[1] Kozen D, *Complexity of Finitely Presented Algebras*, Technical Report TR 76–294, Dept. of Computer Science, Cornell Univ., Ithaca, NY, 1976.

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

[3] Shostak R E, An algorithm for reasoning about equality, *Communications of ACM*, 1978, **21**(7): 583–585.

[4] Nelson G and Oppen D C, Fast decision procedures based on congruence closure, *JACM*, 1980, **27**(2): 356–364.

[5] Craigen D, Kromodimoelijo S, Meisels I, et al., Eves system description, *Proc. Automated Deduction - CADE* 11, LNAI 607, Ed. Kapur, Springer Verlag, 1992, 771–775.

[6] Kapur D and Subramaniam M, Mechanically verifying a family of multiplier circuits, *Proc. Computer Aided Verification* (*CAV*), New Jersey, Springer LNCS 1102 (Eds. by Alur R and Henzinger T A), 1996, 135–146.

[7] Kapur D and Zhang H, An overview of rewrite rule laboratory (RRL), *Computers and Math. with Applications*, 1995, **29**(2): 91–114.

[8] Zhang H, Implementing contextual rewriting, *Proc. Third International Workshop on Conditional Term Rewriting Systems,* Springer LNCS 656 (Eds. by Remy J L and Rusinowitch M), 1992, 363–377.

[9] Rybalchenko A and Sofronie-Stokkermans V, Constraint solving for interpolation, *J. Symb. Comput.*, 2010, **45**(11): 1212–1233.

[10] Gallier J H, Fast algorithms for testing unatisfiability of ground Horn clauses with equations, *J. Symb. Comput.*, 1987, **4**(2): 233–254.

[11] Dowling W F and Gallier J H, Linear-time algorithms for testing the satisfiability of propositional Horn formulae, *J. Log. Program.*, 1984, **1**(3): 267–284.

[12] Kapur D, Shostak's congruence closure as completion, *Proc. Rewriting Techniques and Applications,* 8*th Intl. Conf.* (*RTA*-97), (Ed. by Comon H) Sitges, Spain, Springer LNCS 1231, June, 1997, 23–37.

[13] Baader F and Nipkow T, *Term Rewriting and All That*, Cambridge University Press, Cambridge, 1998.

[14] Tarjan R E, Efficiency of a good but not linear set union algorithm, *Journal of ACM*, 1975, **22**: 215–225.

[15] Galler B A and Fisher M J, An improved equivalence algorithm, *C. ACM*, 1964, **7**(5): 301–303.

[16] Cocke J and Schwartz J T, *Programming Languages and Their Compilers: Preliminary Notes*, Second Revised Version, Courant Institute of Mathematical Sciences, NY, 1970.

[17] Nieuwenhuis R and Oliveras A, Fast congruence closure and extensions, *Information and Computation*, 2007, **205**(4): 557–580.

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

[19] Zhang H and Kapur D, First order theorem proving using conditional rewrite rules, *Proc. 9th Intl. Conf. on Automated Deduction* (*CADE*), Springer LNCS 310, (Eds. by Lusk E W and Overbeek R A), Argonne, USA, May, 1988, 1–20.

[20] Bachmair L, Ganzinger H, Lynch C, et al., Basic paramodulation and superposition, *Proc. Automated Deduction — CADE* 12, LNAI 607 (Ed. by Kapur), Springer Verlag, 1992, 462–476.

[21] Jouannaud J P and Kirchner H, Completion of a set of rules modulo a set of equations, *SIAM J. of Computing,* 1986, **15**(4): 1155–1194.

[22] Knuth D and Bendix P, Simple word problems in universal algebras, *Computational Problems in Abstract Algebra* (Ed. by Leech), Pergamon Press, 1970, 263–297.

[23] Bachmair L, Tiwari A, and Vigneron L, *Abstract Congruence Closure*, Springer-Verlag, New York, 2003.

[24] Dershowitz N, Canonical sets of Horn clauses, *Proc. 18th ICALP*, LNCS 510, 1991, 267–278.

[25] Bonacina M P and Dershowitz N, Canonical ground Horn theories, Ganzinger Festchrift, LNCS 7797, 2013, 39–69.

[26] Cyrluk D, Lincoln P, and Shankar N, On Shostak's decision procedures for combination of theories, *Proc. Automated Deduction - CADE* 13, LNAI 1104 (Eds. by McRobbie and Slaney), Springer Verlag, 1996, 463–477.

[27] Kapur D, Efficient Interpolant generation algorithms based on quantifier elimination: EUF, Octagons, $\cdots$, *Proc. Dagstuhl Seminar* 17371–*Deduction beyond First-order Logic,* Wadern, Germany, Sep. 2017, A journal version is under preparation; a draft can be obtained from the author.

[28] Bachmair L, Ramakrishnan I V, Tiwari A, et al., Congruence closure modulo associativity and commutativity, *Proc. Frontiers of Combining Systems, Third International Workshop* (*FroCoS*), Nancy, France, 2000, 245–259.

[29] Narendran P and Rusinowitch M, Any ground associative-commutative theory has a finite canonical rewrite system, *Proc. 4th Intl. Conf. on Rewriting Techniques and Applications* (*RTA*), LNCS 488, Springer, 1991, 423–434.

[30] Kandri-Rody A, Kapur D, and Narendran P, An ideal-theoretic approach for word problems and unification problems over commutative algebras, *Proc. First International Conference on Rewriting Techniques and Applications* (*RTA*-85), Dijon, France (Eds. by Jouannaud and Musser), Springer LNCS 202, May 1985, 345–364.

[31] Bonacina M P and Johansson M, On interpolation in automated theorem proving, *J. Autom. Reasoning*, 2015, **54**(11): 69–97.

[32] Gulwani S and Musuvathi M, Cover algorithms and their combination, *Proc. 17th European Symposium on Programming, ESOP* 2008, Springer LNCS, 2008, 193–207.

[33] Le Chenadec P, Canonical forms in the finitely presented algebras, Ph.D. Thesis, U. of Paris 11,

1983.

[34] Ballantyne A M and Lankford D, New decision algorithms for finitely presented commutative semigroups, *Computers and Mathematics Applications,* 1981, **7**: 159–165.

Appendix

Details of the conditional constant congruence closure algorithm, particularly the data structures used, are provided below. Most data structures are adapted from [10, 11].

Given a set S' of constant Horn equations, let S_E be the subset of S including unconditional equations. Invoke Tarjan's Union-Find algorithm with path compression and tree size balancing. For every constant, there is a distinct node in a graph constructed from S_E. The output is a forest of trees whose roots are nodes corresponding to constants which serve as the canonical forms of constants corresponding to the nodes in the trees. Let k be the total number of constants in S. This step's amortized cost is $O(\max(k', |S_E|) * \alpha(k'))$, where $k' \leq k$ is the number of constants in S_E. Let $k'' \leq k'$ be the number of root nodes.

The above step is followed by processing Horn equations in S with nonempty hypotheses. As a Horn equation is processed, the data structures below are built using $Find$ on each atom in the Horn equation using path compression. Any trivial equality, after $Find$, in the hypothesis is deleted. If its concluding equality becomes trivial, then the Horn equation is deleted since it is implied by $CC(S_E)$. All Horn equations are expressed in terms of root nodes and any additional constants not in S_E.

Three graph structures are maintained and updated: (i) $Nodes(S)$, in which there is a node for every constant symbol appearing in the input S, (ii) $Atoms(S)$, in which there is a node for every atom (equality in this case but in general also predicate term), with each node having pointers labeled L to the node corresponding to the constant on its lhs and R to the node corresponding to the constant on its rhs, and (iii) $Horns(S)$, in which there is a node for every Horn equation in S. A node in $Horn(S)$ has three labeled pointers: (i) *counter*, indicating how many atoms in its hypothesis are proven `True` so far, (ii) *Hyp*, pointing to all the atoms, if any, in the hypothesis with each pointer having a label `True` or Unknown indicating the status about its validity, and (iii) *Conc*, pointing to the equality (or atom) in its conclusion with status label `True` or `Unknown`.

Initially, the counter is set to 0 for equations in the input and the number of equalities in the hypothesis of conditional equations. For every node c in $Nodes(S)$, there is also a list of nodes of equalities in $Atoms(S)$ in which c appears.

For every node e in $Atom(S)$, there are two lists, Hyp and $Conc$, of nodes from $Horn(S)$ with first list showing all Horn equations whose hypotheses includes e and second list all Horn equations in which e appears as a conclusion. These data structures allow fast detection and processing of implied equalities due to propagation. There is also a list (queue) EQ of unconditional equations.

The algorithm proceeds as follows:

Build the above data structures. Initially unmark all nodes in $Horn(S)$. The algorithm terminates if EQ becomes empty and there is no unmarked node in $Horn(S)$ whose counter is 0. The conditional congruence closure then consists of a forest of trees whose root nodes are canonical forms, and a finite set of unmarked Horn equations, with a nonzero counter, Hyp with Unknown status pointers to atoms in $Atoms(S)$ and Conc with an Unknown status pinter to the atom serving its conclusion.

For every node in EQ, the concluding equality is processed as follows after marking it: (i) If $Find(lhs) = Find(rhs)$, then mark the node and remove it from EQ. During the computation of $Find$, do path compression by linking all constant nodes during $Find$ computation directly to their root, as in [14]. Data structures are updated for every constant node $Node(S)$ whose representative changes. If $Find(lhs) = a \neq b = Find(rhs)$, then merge nodes a and b using tree balancing as in [14]; wlog, if b is chosen as the new representative, then a is now linked to b. From a, pointers to all equality atoms in which a appears, are changed to b, changing the status of the equality atom to `True` if it becomes trivial, For the equality atom whose status is changed, (a) if it appears as a conclusion in some Horn equation, the corresponding Horn node is deleted and marked as deleted; (b) otherwise, if it appears in the hypothesis of a Horn equation, its counter is decreased by 1. For every unmarked node in $Horn(s)$ whose counter is 0, put into EQ, its concluding equality.

Nodes in $Atoms(S)$ are marked as deleted if their canonical forms are identical as they are redundant. Otherwise, the canonical form of an undeleted equality in $Atoms(S)$ is the equality on its two distinct nodes.

In analyzing the complexity of various steps, the crucial additional step is (iii) where whenever canonical forms change (the height of a tree), equalities using that canonical forms must be checked. So if there are m equality atoms including those in conclusions, then at most $m * \log(m)$ additional steps are needed. Thus, the overall complexity of the algorithm is $k * \alpha(k) + m * \log(m) + n$ where k is the number of constants and m is the number of equality atoms and n is the size of input.