A New Algorithm for Computing (Strongest) Interpolants over Quantifier-Free Theory of Equality over Uninterpreted Symbols (EUF)*

Deepak Kapur

Dept. of Computer Science
University of New Mexico, Albuquerque, NM, USA
kapur@cs.unm.edu

– *Preliminary Draft* –

1 Introduction

An interpolant [4] for a pair of first order formulas (α, β) such that α implies β is a formula ψ such that (1) α implies ψ, (2) ψ implies β and (3) ψ is over the common language of α and β, that is, every non-logical symbol appearing in ψ appears both in α and in β. For first order logic, it is known that an interpolant always exists [4].

An alternate definition of an interpolant popularized by Mcmillan is: Given a pair of first order formulas (α, β') such that $\alpha \wedge \beta'$ is unsatisfiable, there exists ψ, called an inverse interpolant, such that (1) α implies ψ, (2) $\psi \wedge \beta'$ is unsatisfiable and (3) every non-logical symbol appearing in ψ appears both in α and in β' (an interpolant and reverse interpolant are related by making $\beta' = \neg\beta$). Yet another equivalent way to define an inverse interpolant is a formula ψ such that α implies ψ and β' implies $\neg\psi$.

Huang [9] might have been the first one to have published an inference system for generating an interpolant from propositional refutational proofs; he used OTTER to generate interpolants for first-order proofs by essentially working on their propositional structure. However, his work went unnoticed for quite sometime. Pudlack and Krajicek [19, 14] were credited for proposing interpolant generation algorithms in which they showed how interpolants can be constructed from refutational proofs of for propositional calculus as well as integer inequalities, a number of methods have been proposed for generating interpolants for various first-order theories from refutational proofs. The main idea seems to be to annotate nodes of a refutation proof tree $\alpha \wedge \beta'$ based on whether symbols are solely from α or β' or common.

In this paper, we take a radically different approach first indirectly suggested by our paper [12] in which an obvious relationship between interpolation and quantifier elimination was established. This approach is elaborated for two subtheories which have turned out to be quite useful in program analysis as well as

* Supported in part by an NSF award CCF-1248069 and DMS-1217054.

analysis of timed systems: (i) quantifier-free theory of equality of uninterpreted symbols is a subfragment of first order logic which has been found useful for both hardware and software analysis; more on this later. (ii) octagonal formulas of the form $\pm x \pm y \leq c, c \in \mathbb{Z}$, which have been found so useful in program analysis that it has become the main engine (abstract domain) in the ASTREE system based on the abstract interpretation framework.

Uninterpreted symbols can be used for modeling memory access, arrays, as well as other function symbols, such as $*$, exp on numbers, whose semantics is difficult (or even impossible) to capture without making those subtheories undecidable. In [13], we proposed a reduction approach in which we demonstrated how satisfiability problem over quantifier-free theories over container data structures such as finite arrays, finite sets, finite multisets as well as recursively defined dats structures can be reduced to a combination of quantifier-free Presburger arithmetic (PA) and quantifier-free theory of equality over uninterpreted function symbols (EUF). Further, we also showed in [12] interpolants for container theories can be generated using interpolants over EUF and PA. There we also an intimate relationship between interpolants and quantifier elimination. The main idea in that paper is simple: an interpolant, I_α can be generated by elimination/projecting out uncommon symbols from α (equivalently, a different interpolant I_β can be generated by eliminating uncommon symbols form β).

In this paper, we continue this line of research and give methods for generating interpolants for these two subtheories and their combination based on quantifier elimination. Our approach thus does not require the need to generate refutational proofs; in fact, in contrast, no such proof is needed; instead we work directly on α (equivalently, β). Formulas α and β are needed only to compute common symbols appearing in them. Interpolants can be generated both from α as well as β only as long as common symbols among the two are known. Our approach generates the strongest interpolant I_α implied by α, which not only serves as the strongest interpolant for β used to identify common symbols between α and β but also a whole family fo β's which are implied by α and which have the same set of common symbols with α as used for computing the interpolant. In the same spirit, dual of the above also holds in the sense that an interpolant I_β can als be generated from β which is the weakest interpolant implying β and it serves as the weakest interpolant for a whole family of α's which imply β and have the same set of common symbols as an α used to generate I_β.

As we prove later, all formulas in the lattice of formulas of common symbols with I_α as the bottom and I_β as the top using the implication ordering are also interpolants thus giving us interpolants of different strengthes found useful in program verification. This is in contrast to generating an interpolant from a refutational proof which is specific to (α, β) as well as its strength and quality depend upon the particular refutational proof as demonstrated later. It is our contention that most program analysis techniques based on model checking, IC^3 and its various generalization, bounded model checking, predicate abstraction etc., which use interpolants from a trace/execution of a counter-example differ in which interpolants in the lattice of interpolants of a pair (α, β) are selected;

in some sense, the performance of these algorithms can be compared by position in the lattice of interpolnats used by these techniques.

2 Preliminaries

We start with standard definitions of many-sorted logics, theories, and interpolation.

3 Quantifier Elimination based Interpolant Generation Algorithms

Given α, β such that α implies β, an interpolant ψ over the common symbols of α and β, such that α implies ψ and $\psi \implies \beta$ can be generated by eliminating from α all symbols that are exclusively in α (equivalently, all nonlogical symbols that are exclusively in β).

Here is a general algorithm for computing a family of interpolants for (α, β) under the assumption that a method exists to eliminate ab symbol x from a formula δ such that $\forall x \delta \equiv \psi$ where ψ does not include x. If α (respectively, β) is a tautology, then the constant T, standing for a valid formula, is I_α (respectively, I_β); similarly iff α (respectively, β)) is unsatisfiable, then the constant F, standing for an unsatisfiable formula, is I_α (respectively, I_β). Below, we assume that neither α nor β a tautology or unsatisfiable. Further, let C be the common symbols in α and β and $\alpha \wedge \neg(\beta)$ is unsatisfiable. Without any loss of generality, we can assume that α is a quantifier-free conjunction of literals.

Let I_α be a formula obtained from α by eliminating all symbols $\bar{x}$ which are in α but not in β (henceforth called uncommon symbols in α), such that $\forall \bar{x} \alpha$ is equivalent I_α.[1] Similarly, let I_β be a formula obtained from β by eliminating all symbols $\bar{y}$ which are β but not in α, such that $\forall \bar{x} \beta$ is equivalent I_β.

It can be easily shown that $I_\alpha \implies I_\beta$. Further, any formula ψ on common symbols of α and β such that $I_\alpha \Rightarrow \psi$ or $\psi \Rightarrow I_\beta$ is an interpolant for (α, β). In this way, a whole family of interpolants can be obtained for (α, β), which has a lattice structure with I_α as the bottom (0) and I_β as the top (1) of the lattice ordered by implication relation. Since I_α is assumed to be the strongest formula implied by α only on the common symbols of α and β, it follows that any interpolant of α and β is implied by I_α.

In case a theory does not admit quantifier elimination then incomplete heuristics for quantifier elimination can be employed. When all uncommon symbols

[1] It is not essential to require that I_α be equivalent to $\exists \bar{x} \alpha$; it suffices to have I_α to be a strong enough formula implied by $\exists \bar{x} \alpha$; further it is not necessary that the theory under consideration admit quantifier elimination. As we shall see below, the theory of EUF does not admit quantifier elimination because of function symbols appearing in it, however, it is possible to eliminate uncommon function symbols for generating interpolants.

from α have been eliminated (or if all of them cannot be eliminated), then from the result, consider only those subformulas which involve common symbols; their conjunction is I_α. If there are none, then $I_\alpha = T$, suggesting that β must be T as well, thus contradicting our assumption, and implying that there must be at least one subformula involving only common symbols.

An interpolant I_α computed using a common set of symbols CS serves as an interpolant for all (α, β) pairs such that their common symbols are precisely CS (even a super set of CS). In this sense, it is an interpolant for a possibly infinite family of β's.

4 Lattice of Interpolants

It will also be useful to study properties of interpolants: namely if I and I' are distinct interpolants of (α, β), then so is $I \wedge I'$ as well as $I \vee I'$. This follows from $\alpha \implies I$ as well as $\alpha \implies I'$, and $I \implies \beta$ as well as $I' \implies \beta$: $\alpha \implies (I \wedge I')$ as well as $\alpha \implies (I \vee I')$; it is also easy to see that $(I \vee I') \implies \beta$, and further, trivially $(I \wedge I') implies \beta$. Thus interpolants are closed under $\wedge$ and $\vee$. Of course, they are not closed under implication but the lattice of interpolants can be ordered using implication: $I > I'\ iff\ I \implies I'$.

Given a nontrivial α (to mean neither valid nor unsatisfiable), I_α can be a valid formula only if there is no β with common symbols CS with α such that $\alpha \implies \beta$. Consider for an example α to be $(p \wedge q) \vee (\not p \wedge \not q)$ with p (or q) as the only common symbol in CS.

Given a pair (α, β), the lattice of interpolants for this pair is all interpolants I such that $\alpha \implies I \wedge I \implies \beta$. The interpolant generated from β after eliminating uncommon symbols using a complete quantifier elimination method is the weakest interpolant, thus the bottom of this lattice, whereas the interpolant generated from α is the strongest interpolant in the lattice, thus the top of the lattice.

It is easy to see that given any other β that is implied by α such that the common symbols of α and $\beta;'$ are the precisely those of α and β, then I_α also serves the strongest interpolant of (α, β'), Similarly given any α' that implies β such that the common symbols of α' and β are precisely those of α and β, then any interpolant of (α', β) implies I_β. There is this duality relationship. In fact, it suffices to consider β' with a subset of common symbols and dually α' with a superset of common symbols.

4.1 Variety of Interpolants from Refutation Proofs

As stated above, almost all algorithms reported in the literature for interpolant generation are guided by refutation proofs of (α, β). That would suggest that all interpolants in the lattice of (α, β) (at least their basis) should be obtainable from all refutation proofs; while seeming true for many first-order theories, however there is no such general result to our knowledge. It is not difficult to see that different refutation proofs can generate nonequivalent interpolants by interpolant generation systems.

It appears also the case that different traversal of a same proof using the same interpolant generation algorithm would lead to different interpolations; further, one refutation proof could be transformed to another refutation proof leading to a different traversal giving different interpolants, particularly of different strengths. Such observations are discussed below for propositional logic. It is our contention it is difficult using such an approach to efficiently generate interpolants of desired strengths.

We will assume that every atomic formula in α, β are essential to prove their unsatisfiability. In the absence of this assumption, it is easy to see that quantifier elimination of the whole α may be extremely inefficient if only a small part of α is likely to participate in establishing unsatisfiability with β and thus may be difficult (almost impossible) to guess in the absence of a refutation proof of $\alpha \wedge \beta$.

In contrast, our approach generates interpolants either from α or from β even though our preference is always α since it generates the strongest interpolant. It is possible that it might be faster in certain cases to generate an interpolant from a refutation proof than performing full quantifier elimination, but we conjecture that is not the case. If a refutation proof of $\alpha \wedge \beta$ is available for free, an unsatisfiable core can be extracted and the portion of α used in the proof can be used to generate interpolants using the proposed approach instead of α.

5 Propositional Calculus

We start with the simplest of the theories where it is easy to see how a variety of interpolants can be generated from a direct proof of $\alpha \implies \beta$, a refutational proof of $\alpha \wedge \neg\beta$, directly from α or from β and how they might different from each other in terms of their quality. Another motivation for discussing this theory is that D'Silva et al [5] were the first one to our knowledge to have studied the issue of quality of interpolants generated from refutational proofs. Otherwise, the topic, which we consider extremely important, has largely gotten ignored (see however McMillan's papers on beautiful interpolants and little interpolants, Reummer's tutorial etc.).

Since $I_\alpha = \exists UC\ \alpha$, an easy way to compute the strongest interpolant from $I_\alpha = \bigvee_\sigma \sigma(\alpha)$, for every truth assignment of uncommon symbols in UC_α. Similarly, the weakest interpolant from $\neg\beta$ can be constructed as: $I_\beta = \neg(\bigvee \delta(\beta)$ for every truth assignment δ of uncommon symbols in UC_β. If the objective is just to get an interpolant, then it can be optimized based on the number of uncommon symbols in α or β. The worst-case complexity is 2^l, where l is the number of uncommon symbols in α.

Theorem 1. *Given a mutually contradictory (α, β) pair of propositional formulas, $I_\alpha = \bigvee \sigma(\alpha)$, where σ is a truth assignment of uncommon propositional variables, UC, in α. is their strongest interpolant.*

Proof. We first show that I_α is an interpolant of (α, β). $\alpha \implies I_\alpha$: Every satisfying truth assignment of α also makes I_α true by construction. Further, for any satisfying assignment of I_α, it can be extended to include assignments of symbols in UC based on a satisfying disjunct in I_α which makes satisfies α.

I_α and β are mutually contradictory since any satisfying assignment of $I_\alpha \wedge \beta$ can be extended to a satisfying assignment of α. making $\alpha \wedge \beta$ satisfiable, which violates the assumption.

To show that I_α is the strongest interpolant, we show that any interpolant I of (α, β) is implied by I_α: by definition, $\alpha \implies I$; I cannot be stronger than I_α to imply that there is an assignment that makes I_α true but I false; this satisfying assignment is extended to include the assignment to uncommon symbols in UC which make α true but that contradicts the assumption that I is an interpolant. Further, $I_\alpha \wedge \neg I$ cannot be satisfiable either as any falsifying assignment of I makes α false irrespective of any assignment of uncommon symbols in UC, which also makes I_α false. None

The reader should keep the above argument in mind when we show gives such proofs about interpolant generation for other theories as well.

Duality gives a proof that $I_\beta = \neg(\bigvee \sigma(\beta)$ is also an interpolant (α, β) and is in fact, their weakest interpolant.

Consider the following examples from [5]. The first example has $\alpha = (a_1 \vee \neg a_2) \wedge (\neg a_1 \vee \neg a_3) \wedge a_2$ whereas $\beta = (\neg a_2 \vee a_3) \wedge (a_2 \vee a_4) \wedge \neg a_4)$. The common symbols are a_2, a_3. $I_\alpha = (\neg a_3 \wedge a_2) \vee (\neg a_2 \wedge a_2)$ which simplifies to $\neg a_3 \wedge a_2$. From β, $I_\beta = \neg((\neg a_2 \vee a_3) \wedge a_2)$ which simplifies to $\neg a_2 \vee \neg a_3$. Clearly, $I_\alpha \implies I_\beta$; in fact, the implication is strict giving a nontrivial lattice of interpolants. Thus, any formula that is implied by I_α and implies I_β is also an interpolant, e.g., $\neg a_3$; however, a_2 is not an interpolant since it does not imply I_β; it can be easily verified that $a_2 \wedge \beta$ is satisfiable. Notice that I_α is also an interpolant of $\beta_1 = \neg a_2, \wedge a_4, a_3 \wedge a_4, (\neg a_2 \vee a_3) \wedge a_4$ as well as many more formulas involving symbols other than a_2, a_3.

The second example in citeDSilva10 is trivial since there is no uncommon symbol: $\alpha = \neg a_1 \wedge (a_1 \vee \neg a_2)$, $\beta = (\neg a_1 \vee (a_2) \wedge a_1$. Both α and β are interpolants, but so are $\neg a_1$ as well as $\neg a_2$. Unlike the discussion in [5], note that there was no need to have access to a refutational proof of $\alpha \wedge \beta$.

Here is another example from Bonacina and Johanson's survey paper [1] : (example 2): $\alpha = (a \vee e) \wedge (\neg a \vee b) \wedge (\neg a \vee c), \beta = (\neg b \vee \neg c \vee d) \wedge \neg d \wedge \neg e)$. In α, the uncommon symbol is a, so $I_\alpha = (b \wedge c) \vee e$; the uncommon symbol in β is d; $I_\beta = \neg((\neg b \vee \neg c) \wedge \neg e)$. $I_\alpha = I_\beta$ implying the lattice of interpolants has only 1 element. Again I_α serves as an interpolant for many formulas mutually contradictory with α which include a subset of $\{b, c, e\}$ and any other symbols different from a.

In a typical interpolant generation algorithm based on a refutation proof of $\alpha \wedge \beta$, a unsatisfiable core of subformulas participating in a refutational proof is identified; smaller cores are preferred over larger ones since an unsatisfiable core need not be unique in general as it is expected that they may give smaller interpolants. It is desired that there are no unnecessary intermediate formulas even though they may appear in a proof tree. Of course the role of lemmas needed in a proof affecting interpolant generation is totally unclear as well. Given a proof, there can be numerous ways to traverse the same proof giving rise to different interpolants of different strengths; see discussion below. So far we have focussed on a single proof; in general there can be many proofs, perhaps even infinitely many, of the unsatisfiability of $\alpha \wedge \beta$; what proof method to be preferred over others for interpolant generation is an open question. A succinct proof may not produce a "good" interpolant for an application.

Interpolant generation algorithms differ also in the way a proof tree is used to generate an interpolant, particularly, where as $\alpha \wedge \beta$ as well as $\beta \wedge \alpha$ are unsatisfiable, an interpolant generated from (α, β) could be different from the one from (β, α). At least two different interpolation generation inference systems are discussed there for generating interpolants from a refutation proof: *Symmetric System* and *McMillan's System*; And, there could be many more, especially for richer theories. Let us now consider the discussion in [5] with regards to the above two examples. While Huang's system is symmetric, producing an interpolant for the first example to be: $\neg a_3$, whereas McMillan's asymmetric system generates $a_2 \wedge \neg a_3$. For the second example, the symmetric system as well as McMillan's system generates $\neg a_1 \wedge \neg a_2$ as an interpolant; of course there are $\neg a_)$ as well as $\neg a_2$ are each interpolants as well but they cannot be generated from either from Huang's system or McMillan's system.

Jhala and McMillan [10] reported how different interpolants can affect the convergence of software model checkers based on predicate abstraction. Particularly, rearranging the same refutation proof can lead to different interpolants which may be preferred over other interpolants. Proof transformations/restructuring has been extensively studied; see [21].

We however are able to side step this whole issue by considering a totally different approach; our algorithm is not only simpler but generates the strongest interpolant which can thus work for a large family of β's insofar as each is mutually contradictory with α.

6 Interpolant Generation for Octagonal Formulas

This section discusses a simple as well as efficient algorithm for generating the strongest interpolant of (α, β) in the theory of octagonal formulas. The algorithm is based on quantifier-elimination and in fact an adaption of Fourier-Motzkin's algoritm to octagonal formulas.

Consider α, a finite conjunctions of $ax_i + by_i \leq c$, where $a, b \in \{-1, 0, -1\}$ but $c \in \mathbb{Z}$. Given a set of symbols x_i's and y_j's declared to be local to α, our goal is to eliminate them from α to get a projection which is a formula purely in the remaining symbols. This is done by eliminating one symbol at a time and generating a new set of octagons formulas as follows:

for every pair of an octagon atoms $ax_i + by_j \leq c$ and $a'x_i + b'y_k \leq d$ where the signs of a and a' are opposite of each other (implying one is -1 and the other is 1), generate a new octagon

Elim: $by_I + b'y_j \leq c + d$.

This is done for all such pairs. In the special case when $y_i = y_j$, the above can give $2y_i \leq c+d$ which must be normalized by dividing $c+d$ by 2 and taking the integer floor (in case of computing a projection over the reals or rationals, simple divison is performed).

Before applying the above rules, preprocessing can be performed to simplify α: $x - x \leq c$ is T if c is nonnegative, and F otherwise; $x - y \leq c \wedge x - y \leq d$ is simplified to $x - y \leq min(c, d)$.

Repeat this process until all local symbols are eliminated generating an interpolant I_α by taking the conjunction of all octagonal atoms which do not have any uncommon symbol in α and other octagonal formulas generated using **Elim**.

The following properties of the above algorithm are easy to prove. f

Theorem 2. *Given a mutually contradictory pair (α, β), where α, β are finite conjunctions of octagonal atoms, the above algorithm terminates with an interpolant that is a finite conjunction of octagonal atoms.* *None*

Proof. The termination follows from the fact that every step eliminates a symbol. Given that there are only finitely many uncommon symbols in α to be eliminated, termination is obvious.

Elim generates $by_I + b'y_j \leq c + d$ from $ax_i + by_j \leq c$ and $a'x_i + b'y_k \leq d$, where $a = -a'$. Any satisfying assignment of variables x_i, y_j, y_k for $ax_i + by_j \leq c$ and $a'x_i + b'y_k \leq d$ is also a satisfying assignment of $by_I + b'y_j \leq c + d$. In case $y_i = y_j$, its coefficient has to be made 1 by dividing the bound by 2.

Given a satisfying assignment of $by_I + b'y_j \leq c + d$, it is possible to extend it to x_i satisfying $ax_i + by_j \leq c$ and $a'x_i + b'y_k \leq d$ as follows: $ax_i \leq c - by_j$ as well as $a'x_i \leq d - b'y_k$, pick value of x_i satisfying the upper and lower bounds based on the values of y_i, y_j in the satisfying assignment. However, such constraints on x_i must be generated from all pairs of octagonal atoms in which x_i appears. More simply, plug the satisfying assignment of I_α into common symbols in α generating constraints on uncommon symbols and extend the satisfying assignment to include values of uncommon symbols satisfying these constraints. The existence of the extended assignment is guaranteed since α is satisfiable.

Let I the subset of all octagonal atoms in common symbols from α and octagonal atoms generated by **Elim**. $\alpha \implies I$ since every satisfying assignment of α is also a satisfying assignment of I. Given that α and β are mutually contradictory, I and β are also mutually contradictory since otherwise any satisfying assignment of $I \wedge \beta$ can be extended to a satisfying assignment of α.

If n is the size of α which puts an upper bound on the total number of variables in it, then $O(n^2)$ new octagon formulas are generated, which is also an upper bound on the number of octagon formulas generated with $O(n)$ variables.

The worst case complexity of computing the interpolant I_α is $O(n^3)$ since for each variable to be eliminated, $O(n^2)$ formulas may need to be analyzed in intermediate steps. Typically the number k of variables to be eliminated is much smaller than n, in which case the complexity is $O(n * k^2)$.

Below, we illustrate the algorithm on examples mostly taken from Griggio's thesis [7]. in his thesis, Griggio does a detailed analysis of an unsatisfiable set of octagon formulas and considers different partitions of it into α and β to illustrate the intricacies of his graph based algorithm. Following Mine who introduced two variables x^+ and x^- for every variables x to stand for $+x$ and $-x$, to respectively and transforming every octagonal constraint into two difference constraints of the form $u - v \leq c$, a set of octagonal constraints can be represented as a difference graph. If this graph is a negative cycle, then the constraint set is unsatisfiable. Even when the graph has a 0 weight cycle, then sometimes constraints can be unsatisfiable.

Some characteristics of Griggio's example are (i) when represented as a graph of difference constraints, it is unsatisfiable but with a 0 weight cycle and (ii) his algorithm generates a conditional interpolant for some partitions even though the same refutation proof is used. Griggio showed the behavior of his algorithm for different ways to generate interpolants based on the common symbols of various partitions.

Consider the last case of two formulas $\alpha = \{x_1 - x_2 \geq -4, \quad - x_2 - x_3 \geq 5, \quad x_2 + x_6 \geq 4, \quad x_2 + x_5 \geq -3\}$ and $\beta = \{ -x_1 + x_3 \geq -2, \quad - x_4 - x_6 \geq 0, \quad - x_5 + x_4 \geq 0\}$

Griggio's algorithm generated a conditional interpolant $x_6 + x_5 \geq 0 \implies x_3 - x_1 \geq -3$.

In contrast, in our approach, we identify x_2 as the variable to be eliminated from α, since it is local to α only. Using the *Elim* rule to eliminate x_2 from complimentary literals, we get $\{-x_3 + x_5 \geq 2, x_1 + x_6 \geq 0, x_1 + x_5 \geq -7, \quad -x_3 + x_6 \geq 9\}$, leaving no other pairs of octagon formulas with a positive x_2 and a negative x_2. Since every literals of α includes x_2, nothing from α is included in the interpolant, with the result being $I_\alpha = \{-x_3 + x_5 \geq 2, x_1 + x_6 \geq 0, x_1 + x_5 \geq -7, -x_3 + x_6 \geq 9\}$. It can be checked that I_α is implied by α and is inconsistent with β,

Griggio's conditional interpolant $(-x_6 - x_5 \geq 0) \Rightarrow (x_1 x3 \geq 3)$ is implied by I_α since $-x_6 - x_5 \geq 0$ with $x_1 + x_5 \geq -7 \wedge x_1 + x_6 \geq 0$ with tightening gives $x_1 \geq -3$; similarly, $-x_6 - x_5 \geq 0$ with $-x_3 + x_5 \geq 2 \wedge -x_3 + x_6 \geq 9$ with tightening gives $-x_3 \geq 6$; from $x_1 \geq -3 \wedge -x_3 \geq 6$, we get $x_1 - x_3 \geq 3$. Further, The interpolant generated by the proposed algorithm is strictly stronger than Griggio's interpolant.

6.1 Other Examples

Consider example 1 in [3] which can be done over the rationals because of a negative weight cycle, and thus does not require any complex analysis of the negative cycle.

$\alpha = \{-x_2 - x_1 + 3 \geq 0, x_1 + x_3 + 1 \geq 0, -x_3 - x_4 - 6, x_5 + x_4 + 1 \geq 0\}$,
$\beta = \{x_2 + x_3 + 3 \geq 0, x_6 - x_5 - 1 \geq 0, x_4 - x_6 + 4 \geq 0\}$.

The uncommon symbol to be eliminated from α is x_1: its elimination gives: $I_\alpha = \{-x_2 + x_3 + 4, -x_3 - x_4 - 6, x_5 + x_4 + 1 \geq 0\}$. If we eliminate the uncommon symbol x_6 from β: $I_\beta = \neg(x_2 + x_3 + 3 \geq 0 \wedge x_4 - x_5 + 3 \geq 0)$. In contrast, Cimatti et al. reported $(-x_2 - x_4 - 2 \geq 0 \wedge x_5 - x_3 - 5 \geq 0)$, which is strictly implied by I_α and hence weaker than I_α, and which implies I_β.

Let us now consider the examples used by Griggio to illustrate different case analysis of 0-weight cycles. The case above in our view, illustrates the most complex case. The other three cases, which are relatively easy, are illustrated below.

Example 2: $\alpha = \{x_3 - x_1 \geq -2, -x_6 - x_4 \geq 0, x_4 - x_5 \geq 0\}$, and $\beta = \{x_1 - x_2 \geq -4, \ -x_2 - x_3 \geq 5, \ x_2 + x_6 \geq 4, \ x_2 + x_5 \geq -3\}$, reversing α and β from the running example. x_4 is eliminated from α to give: $x_3 - x_1 \geq -2, -x_5 - x_6 \geq 0$ as I_α; this result is the same as in [3].

Example 3: $\alpha = \{x_1 - x_2 \geq -4, \ x_2 + x_6 \geq 4, \ -x_4 - x_6 \geq 0, \ -x_1 + x_3 \geq -2\}$ and $\beta = \{ \ -x_2 - x_3 \geq 5, \ x_2 + x_5 \geq -3, \ x_4 - x_5 \geq 0\}$. The symbols local to α are x_1, x_6. Eliminating them gives: $I_\alpha = \{x_3 - x_2 \geq -6, x_2 - x_4 \geq 4\}$, again the same as reported by [3].

Example 4: $\alpha = \{x_1 - x_2 \geq -4, \ x_2 + x_6 \geq 4, \ -x_1 + x_3 \geq -2, \ -x_2 - x_3 \geq 5\}$ and $\beta = \{ \ x_2 + x_5 \geq -3, \ -x_5 + x_4 \geq 0, \ -x_4 - x_6 \geq 0\}$. The symbols local to α are x_1, x_3. Eliminating them gives: $I_\alpha = \{-x_2 \geq 0, x_2 + x_6 \geq 4\}$, again the same as reported by [3].

6.2 Termination, Correctness and Properties of Interpolants

Theorem 3. *Given an (α, β) such that $\alpha \wedge \beta$ is unsatisfiable, the algorithm generates an interpolant I_α to be a conjunction of octagonal literals. Further, I_α is the strongest interpolant of (α, β).* *None*

Proof. The Elim step generates an octagonal literal from a pair of octagonal literals. The Normalize step generates a bound on a variable. The algorithm picks among all the octagonal literals so generated, a subset only in common symbols, implying that the generated interpolant is indeed a conjunction of octagonal literals

Consider any other interpolant I of (α, β) such that either I strictly implies I_α or noncomparable with I_α in the strict implication ordering. Wlog, I is also a conjunction of octagonal atoms (in general I is any boolean combination of octagonal atoms, but it can be converted to a disjunctive normal form in which each disjunct is a conjunction of octagonal atoms and further some nonempty subset of the disjuncts is an interpolant). Further the symbols of I are a subset of symbols of I_α as I_α includes all the symbols of α except those in UC.

If I strictly implies I_α, then there is a satisfying assignment of common variables of I_α that falsifies I This satisfying assignment of I_α can be extended as shown in the termination proof above to include assignments of uncommon symbols eliminated from I_α which makes α true, which contradicts the assumption that it falsifies I.

It also cannot be that $I_\alpha \wedge \neg I$ is satisfiable: a satisfying assignment of I_α can be extended to a satisfying assignment of α which implies that I cannot be false on that assignment. None

I_α as generated above is the strongest interpolant for every pair $(\alpha, \beta'$ such that I_α and β' are mutually contradictory. Further, I_α is the strongest interpolant for any pair mutually contradictory pair (α, β) in so-far as no symbol in UC appears in β, thus serving as the strongest interpolant for not one single β but rather a family of β's.

A similar proof establishes that

Theorem 4. *Given an (α, β) such that $\alpha \wedge \beta$ is unsatisfiable, the algorithm generates an interpolant I_β to be a disjunction of octagonal literals from β by applying the above algorithm on β and taking the negation of the interpolant generated. Further, I_β is the weakest interpolant of (α, β).* *None*

6.3 Extensions: Difference Atoms and Transitive Relations

Difference logic is a strict subset of octagonal logic in which only literals allowed are of the form $x - y \geq c$ for some constant c. It is easy to see that interpolants for this logic can be easily generated using merely the *Elim* rule along with simplification rule.

If lower and upper bounds on a variable are also included as literals, even then *Elim* rule suffices along with simplification rule. In particular, the normalization step is never needed since it is applicable only on two literals in which two distinct variables have the same sign in one of the literals, which is not allowed in difference logic.

Transitive Relations In [15], Kroening et al discuss the use of transitive relations for expressing properties of bounded arithmetic on bounded integers as implemented in programming languages. They gave an interpolating decision procedure for the quantifier-free theory of uninterpreted function symbols with transitive relations $>, \geq, =, \neq$. Their procedure is graph based, very much in the spirits of [3] for the theory of transitive relations and of [6] for EUF. Below we first show how transitive relations can be handled in our approach. Then in the next section, we show how to combine interpolation generation for transitive relations with that on EUF.

Interpolants for Transitive Relations Consider a formula pair (α, β), where each formula is a conjunction of atoms of the form $x > y$, $x \geq y$, $x = y$ and $x \neq y$. These relation symbols have standard interpretation; $x > y \Leftrightarrow ((x \geq$

$y) \wedge (x \neq y))$; $\geq$ is a partial ordering which is reflexive, antisymmetric, and transitive.

The simplification step is similar to the octagonal case: $x = x$ as well as $x \geq x$ is T, and can thus be eliminated from any conjunction. $x \neq x$ and $x > x$ are false, simplifying any conjunction to be false. Idempotency property of $\wedge$ is used to eliminate any duplicate atoms.

Given α which is in simplified form, a variable x can be eliminated from it by considering all pairs of the form $x \geq y$ (respectively, $x > y$) and $z \geq x$ (respectively, $z > x$) and deriving $z \geq y$ (respectively, $z > y$) from the pair; when there are mixed strict inequality and inequality, then the derived atom is a strict inequality also. A symbol appearing in an equality literal is eliminated by replacing it by the symbol it is equal to: $y = x$ and $x = z$, derive $y = z$ by eliminating x. Mixed cases are also similar: from $y = x$ and $x > z$ (respectively, $x \geq z$), derive $y > z$ ($y \geq z$); from $y = x$ and $x \neq z$, derive $y \neq z$. All these computations can be done efficiently using the UNION-FIND data structure of Tarjan, as noted in [15].

An interpolant can be generated in $O(n^2)$ (consider α including $\{x \geq u_1, x \geq u_2, \ldots, x \geq u_k, v_1 \geq x, \ldots, v_j \geq x\}$) where n is the size of α, or even in $O(n\alpha(n))$ if only equalities and disequalities are in literals.

7 Theory of Equality over Uninterpreted Function Symbols -EUF

We discuss a new algorithm for the quantifier-free theory of equality over uninterpreted symbols also known as EUF which we have developed using approximate quantifier elimination approach. This theory is interesting not only because it is supported in almost every SMT solver and further it is extensively used in our method for generating interpolants for quantifier-free theories of container data structures [12]. From a technical stand point, an interpolation algorithm must eliminate uncommon function symbols implying that complete quantifier elimination is not possible in this theory. A simple example is $\exists x f(x) = y$. It is easy to see however that if formulas involve only constants, then this subclass does admit quantifier elimination even though it is not convex any more ($\exists x x \neq y$ is equivalent to $y = a_1 \vee y = a_2 \vee \ldots$, a disjunction of all constants that are in the equivalence class of y since there are only finitely many constants in a formula after Skolemization). Another interesting aspect of the our approach is that even such conditions conditions, an algorithm can be formulated using approximate quantifier elimination methods.

The proposed algorithm is much simpler and produces stronger interpolants compared to various algorithms proposed in the literature [7, 16, 17, 6]. Most of these algorithms assume a priori the existence of an unsatisfiability proof of $\alpha \wedge \neg\beta$. [6] uses a (color) graph based congruence closure algorithm; [16, 17] uses inference system specifying the properties of equality. An exception is an algorithm implicit in the interpolant generating algorithm of [20] for the combined theory of quantifier-free linear arithmetic over the reals (or rationals) and EUF which does not assume an existence of a refutational proof. Instead it reduces the problem of interpolant generation for the combined theory to that of linear arithmetic by flattening terms over uninterpreted symbols by introducing new constant symbols as well as adding Horn clauses expressing specific congruence closures properties leading to nonconstant terms appearing in α and β.

As we show later using examples, algorithms by McMillan [16, 17] as well as by Tinneli et al [6] are unnecessarily complex in their presentation, thus making them difficult to implement as well as the interpolants generated by them are complex. Further, we are unaware of any complexity analysis of any of the interpolation algorithms. In contrast, the algorithm presented below is simpler and often generates simpler interpolants, and most importantly, its complexity can be easily analyzed. The proposed algorithm is illustrated below on a collection of examples discussed in [6] where they contrast the output of their algorithm with those of McMillan's algorithm.

The key ideas of the algorithm are quite simple and they are summarized: the algorithm is built on top of the congruence closure algorithm in which symbols to be eliminated can never be a representative of a congruence class of members which only have common symbols. There is one additional operation needed to approximate quantifier-elimination in case (i) a function symbol must be eliminated or (ii) a constant symbol can only be eliminated by eliminating a function symbol to which it is an argument even though that function symbol does not have to be eliminated. In this step, approximation in quantifier elimination is performed. The algorithm can generate compact interpolants and is efficient if new symbols are allowed to serve as placeholders for terms in which no symbol needs to be eliminated; an alternative to such a presentation is the use of directed acyclic graphs for representing an interpolant.

We use an approach similar to the one presented above for UTVI formulas and focus only on a formula α and a set U of symbols from α which need to be eliminated. The output interpolant will serve as an interpolant for all (α, β) pairs in which β does not have any symbol from U insofar as $\alpha \implies \beta$. In this sense, the output is an interpolant for a possibly infinite family of such β's.

The input to the algorithm is a satisfiable α assumed to be a conjunction of equations and disequations on ground terms and a finite set U of symbols in α which must be eliminated. As above, symbols not be eliminated from α are common to (α, β) whereas symbols to be eliminated from α are uncommon to β's (simply uncommon). We abuse the terminology below and call only nonconstant function symbols as function symbols even though constants are function symbols of arity 0; they would instead be called just constants. Throughout the algorithm, terms containing symbols which do not need to be eliminated will also be called *common*. Any new symbols introduced to stand for such function terms will also be called common.

There are three phases in the algorithm. The first phase is to generate congruence closure much like in [11], using flattening by introducing new symbols to make *flat* terms, or as in [18] where dag representation with labeled nodes (pointers) in which case flattening is built into the data structure. All uncommon constants found equivalent to common constants are eliminated by substitution. At the end of this phase, equations and disequations are divided into two parts: (i) those containing only common symbols, and (ii) those containing at least one uncommon symbol that cannot be eliminated just by substitution.

The second phase is the most interesting as it involves eliminating function symbols in U as well as outside U; this quantifier elimination step is approximate unless nothing is discarded. A function symbol g not in U may have to be eliminated if a symbol in U is *hiding* as an argument of g and there is no other way to eliminate it unless g is eliminated. The result of this phase is a finite set of Horn clauses in which an uncommon symbol can be eliminated only under some conditions.

The final phase is the generation of an interpolant by retaining from the output, formulas with common symbols only.

Below we give more details of the algorithm patterned after Kapur's congruence closure algorithm [11]. In the next subsection, the algorithm will be illustrated using two examples from the literature, comparing the results of our algorithm with the outputs of other well-known algorithms given in [7, 16, 17, 6].

1. **Flattening:** Flatten function terms by introducing new constants so that all equations and disequations in α are of the form: $f(c_1, \cdots c_k) = d,\ c = d$ or $c \neq d$, where $c_1, \cdots c_k, c, d$ are constants. The first type of an equation is called an f-equation to distinguish from a constant equation. All the new constant symbols introduced to stand for flattened subtermsare classified as either common or uncommon based on symbols appearing in them. With innermost traversal of nonconstant subterms in α, a new constant is uncommon iff it stands for a flat term which has an uncommon symbol.
   If an f-equation $f(c_1, \cdots c_k) = d$, where d as an uncommon symbol and $f(c_1, \cdots c_k)$ purely in common symbols, is ever generated, it is split into two equations using a new common symbol, say e: $f(c_1, \cdots c_k) = e$ purely in common symbols and $d = e$. This invariant is throughout the execution of the program (i.e., an uncommon constant is never equated to a term with common symbols).[2] It is easy to see that every f-equation either has only common symbols or at least one of $f, c_1, \cdots c_k$ in the function term is an uncommon symbol. Further all disequations, if any, only express that certain constant symbols are not equal.
2. **Phase I: Elimination of uncommon constants:** Generate equivalence classes induced by constant equations using the UNION-FIND data structure with the requirement that the representative of an equivalence class can only be an uncommon symbol if all symbols in it are also uncommon. Further, an original common symbol is to be used if possible.[3] This can be ensured during

[2] This is purely a technical condition. It is needed if Downey et al's algorithm [18] is used which does not use explicit flattening.

[3] This can be done without sacrificing complexity by appropriately swapping a uncommon symbol by an original common symbol in the UNION function.

intermediate steps or it can be done at the end by swapping a uncommon root representative common symbol in its equivalence class, without affecting the complexity of the equivalence closure algorithm.

Congruence Relation: If two f-equations become identical when constants appearing in them are (conceptually) replaced by their representatives,[4], then update the equivalence relation by equating their constants on the right side. To get the best complexity, the "modify the smaller half" idea [18] is employed to keep a tree representation of a congruence class to be balanced. Any new symbol standing for a function terms, previously marked as uncommon, is marked common if all its symbols become common. After the congruence closure is generated, all uncommon constants which have common constants as representatives, are eliminated.

This is where Kapur's congruence closure algorithm concludes, producing a forrest of DAGs representing congruence classes and a list of f-equations and disequations on constants possibly involving uncommon symbols which do not have any common constant(as a representative. Further no two distinct f-equations have the same function term.

For interpolation generation, all uncommon symbols that has a common symbol as representative can be deleted from the problem since they will not appear in the final output of an interpolant.

Further, any uncommon symbol remaining at this stage cannot be eliminated directly.

3. **Phase II: Elimination of uncommon function symbols:** If a function symbol g is uncommon, then pick a distinct pair of f-equations with outermost g: $g(c_1, \ldots, c_k) = d$ and $g(e_1, \ldots, e_k) = h$, generate the Horn clause: $(c_1 = e_1 \wedge \cdots \wedge c_k = e_k) \implies d = h$. This process is repeated until all distinct pairs of f-equations have been used to generate Horn clauses. Each of these Horn clauses is normalized: (i) delete all trivial equalities in the hypotheses as well as the conclusion, (ii) check whether the conclusion equality is in the local congruence closure generated from the equalities in the hypotheses, in which case, that Horn clause is deleted.[5]

   A Horn clause represents conditional congruence.

   This is a step where approximate quantifier elimination is being performed to generate a quantifier-free formula implied by $\exists g \quad g(c_1, \ldots, c_k) = d$ and

[4] In [11], no attempt was made to optimize this step since the goal of that paper was primarily to mimic Shostak's algorithm, particularly the kind of canonical forms generated by the canonizer function in Shostak's algorithm which allowed the canonical form of a to be $f(f(a))$ from a congruence relation including the equation $f(f(a)) = a$. This step can be optimized using techniques such as signatures as in [18].

[5] It might be a good idea to remember the constraints on such function symbols for building extended interpretation as needed in showing that the proposed algorithm generates the strongest interpolant. These constraints are $g(c_1, \ldots, c_k) = d$ and $g(e_1, \ldots, e_k) = h$ where c_i;'s and e_j's may need to be replaced by common constants/terms in case they become equal to them.

$g(e_1, \ldots, e_k) = h$ if f-equations are discarded since Horn clauses only express relationship under conditions. This is proved in a later section.

4. **Exposing uncommon constants underneath common function symbols:** Uncommon constants appearing as arguments in an f-equation are eliminated by exposing them even when f is a common symbol. For any pair of f-equations with the same outermost function symbol, if at least one of them has as arguments uncommon constants, generate a Horn clause as was done for an uncommon function symbol.[6]

   In principle, this step can be performed irrespective whether any uncommon constant is an argument in an f-equation or not but we apply it only if there is an uncommon constant hiding under f as it can unnecessarily generate more complex interpolants, particularly Horn clause interpolants, which are not the strongest.[7] In fact, many algorithms supported in SMT solvers implement this step irrespective of whether f is common or not.

   Convert a disequation if any to a Horn clauses as well: $c \neq d$ is replaced bt $c = d \implies \mathbf{F}$. In this way, there is no need to treat disequations in any special way.

   The result is equations and Horn clauses purely in common symbols, and hence a part of an interpolant, and equations and Horn clauses with at least one uncommon symbol such that no two distinct f-equations have the same function term.

5. **Phase III: Eliminating Uncommon symbols conditionally**

   At this step, uncommon symbols can only be eliminated conditionally. Such elimination can significantly contribute to the complexity of computing an interpolant including its size, especially if many uncommon symbol can be conditionally eliminated in multiple ways.[8]

   If a Horn clause has only common constants in the hypothesis but the conclusion only has a nonconstant symbol u on one side, then it can be used to conditionally eliminate u from other Horn clauses. After this use, the Horn clause can be deleted from consideration to generate an interpolant. Hopefully this elimination step would result in Horn clauses with the same property. Similarly, if the conclusion of a Horn clause is $u_1 = u_2$ with u_1, u_2 being uncommon, then also $u_1 = u_2$ can be conditionally eliminated in other Horn clauses by replacing them with $\mathbf{T}$. This is repeated until it is not possible to even conditionally eliminate an uncommon symbol.

6. **Generating an interpolant:**

   The conjunction of equations and Horn clauses only with common symbols is the interpolant I_α.

[6] Especially in this step, it is useful to remember constraints on such common function symbols for building interpretations of I_α and β.

[7] Since this step forgets properties of a common function symbol.

[8] This situation is similar to the one in syntactic unification algorithms where a most general unifier substitution can grow exponentially in size if it is not represented using a DAG.

This is the eager approach and its worst case complexity can be exponential in the input size as well as the interpolant so generated can also be exponential size.
Consider an example from [8] in which $\alpha = \{f(z_1, v) = s_1, f(z_2, v) = s_2, f(f(y_1, v), f(y_2, v)) = t\}$ in which v is the only uncommon symbol and f as well as constants $z_1, z_2, y_1, y_2, s_1, s_2, t$ are common. We first show all the steps of the above algorithm until this step.
After flattening, $\alpha = \{f(z_1, v) = s_1, f(z_2, v) = s_2, f(y_1, v) = n_1, f(y_2, v) = n_2, f(n_1, n_2) = t\}$ with n_1, n_2 being the new uncommon symbols along with v from the input. Since there are no constant equations, every equivalence class is a single element which serves as its own representative.
f is a common function symbol, uncommon symbols are hiding under f as its arguments in f-equations. f needs to be eliminated generating Horn clauses from f-equations. Wlog, $f(z_1, v) = s_1$ is retained whereas other f-equations can be discarded.
$\{1.\ z_1 = z_2 \implies s_1 = s_2,\ 2.\ z_1 = y_1 \implies n_1 = s_1,\ 3.\ z_1 = y_2 \implies n_2 = s_1,\ 4.\ (z_1 = n_1 \wedge v = n_2) \implies s_1 = t,\ 5.\ z_2 = y_1 \implies n_1 = s_2,\ 6.\ z_2 = y_2 \implies n_2 = s_2,\ 7.\ (z_2 = n_1 \wedge v = n_2) \implies s_2 = t,\ 8.\ y_1 = y_2 \implies n_2 = n_1,\ 9.\ (y_1 = n_1 \wedge v = n_2) \implies n_1 = t,\ 10.\ (y_2 = n_1 \wedge v = n_2) \implies n_2 = t\}$.
We are now in a situation where no uncommon symbol can be unconditionally eliminated. However, Horn clauses 2, 3, 5 and 6 have one uncommon symbol in their conclusions. They can used to conditionally eliminate these symbols giving among others:
$\{11.(z_2 = y_1 \wedge z_1 = y_1) \implies (s_1 = s_2),\ 12.(y_1 = y_2 \wedge z_1 = y_1) \implies n_2 = s_1,\ 13.(y_1 = y_2 \wedge z_2 = y_1) \implies n_2 = s_2,\ 14.(z_2 = y_2 \wedge z_1 = y_2) \implies s_1 = s_2,\ 15.(y_1 = y_2 \wedge z_1 = y_1 \wedge z_1 = y_2) \implies s_1 = s_1,\ 16.(y_1 = y_2 \wedge z_2 = y_1 \wedge z_1 = y_2) \implies s_1 = s_2,\ 17.(y_1 = y_2 \wedge z_1 = y_1 \wedge z_2 = y_2) \implies s_2 = s_1,\ 18.(y_1 = y_2 \wedge z_2 = y_1) \wedge z_2 = y_2 \implies s_2 = s_2\}$.
The rest of the Horn clauses would also have v in their hypotheses and are not shown to save space. Substitution for n_1, n_2 would generate 4 Horn clauses each corresponding to $7, 10$ which do not play any role in the generation of an interpolant.
The interpolant is: $\{(z_1 = y_1 \wedge z_1 = y_2) \implies f(s_1, s_1) = t,\ (z_1 = y_1 \wedge z_2 = y_2) \implies f(s_1, s_2) = t,\ (z_2 = y_1 \wedge z_1 = y_2) \implies f(s_2, s_1) = t,\ (z_2 = y_1 \wedge z_2 = y_2) \implies f(s_2, s_2) = t,\ 1.z_1 = z_2 \implies s_1 = s_2,\ 11.(z_2 = y_1 \wedge z_1 = y_1) \implies (s_1 = s_2),\ 14.(z_2 = y_2 \wedge z_1 = y_2) \implies s_1 = s_2,\ 15.(y_1 = y_2 \wedge z_1 = y_1 \wedge z_1 = y_2) \implies s_1 = s_1,\ 16.(y_1 = y_2 \wedge z_2 = y_1 \wedge z_1 = y_2) \implies s_1 = s_2,\ 17.(y_1 = y_2 \wedge z_1 = y_1 \wedge z_2 = y_2) \implies s_2 = s_1,\ 18.(y_1 = y_2 \wedge z_2 = y_1) \wedge z_2 = y_2 \implies s_2 = s_2\}$. It is easy to see that 1 implies $11, 14, 16, 17$. Further, $z_1 = y_1 = y_2) \implies \mathbf{T}, z_2 = y_1 = y_2) \implies \mathbf{T}$,
The above process of successive back substitution is similar to the one in Gauss's method for solving linear equations (and syntactic unification to generate a substitution as a unifier, in which no variable being substituted appears in substitutions of other variables).
The reader would notice that this example can be easily generalized so that the interpolant output by the above algorithm is exponential in size of the

input; consequently, the interpolant generation procedure would also take in the worst case, exponential time.
Let $\alpha = \{f(z_1, v) = s_1, f(z_2, v) = s_2, \cdots, f(z_k, v) = s_k,$
$g(f(y_1, v), f(y_2, v), \cdots, f(y_k, v)) = t\}$ with the only uncommon symbol v and $z_1, \cdots, z_k, y_1, \cdots, y_k, s_1, s_2, \cdots s_k, t$ are constant common symbols. α is of size $O(k)$. A pure interpolant would be of exponential size and is of the form $\bigwedge(z_{j_1} = y_{i_1} \wedge z_{j_2} = y_{i_2} \cdots \wedge \cdots z_{j_k} = y_{i_k}) \implies g(s_{u_1}, \cdots, s_{u_k}) = t$ for various possible sets of $s_1, \cdots, s_k$.
In the next subsection, we propose a lazy approach which identifies at the outset which Horn clauses would not help in generating interpolants, leading to a polynomial time algorithm as well as a polynomial size presentation of an interpolant.

7. **Role of Disequations in Interpolant Generation:**
It can be noted that once disequations are transformed to Horn clauses, they do not have to be treated in any special way in the proposed interpolant generation algorithm,
Consider the following illustrative example: $\alpha = \{f(x_1) \neq f(x_2)\}$ in which f is an uncommon symbol. After flattening and introducing two new symbols a_1, a_2 $\{f(x_1) = a_1, f(x_2) = a_2, a_1 \neq a_2$. To eliminate the uncommon symbol f, the Horn equation $x_1 = x_2 \implies a_1 = a_2$ is generated. The disequation $a_1 \neq a_2$ is also turned to the Horn clause: $a_1 = a_2 \implies \mathbf{F}$. a_1 can be conditionally replaced to generate $(x_1 = x_2 \wedge a_2 = a_2) \implies \mathbf{F}$, which normalizes to $x_1 = x_2 \implies \mathbf{F}$, which is the interpolant I_α from α.
Consider two Horn clauses: $x_1 = y_1 \implies a_1 = a_2$ and $(a_1 = a_2 \wedge x_2 = y_2) \implies x_3 = y_3$ in which a_1, a_2 are the only uncommon symbols. The first Horn clause can be used to conditionally replace $a_1 = a_2$ by $\mathbf{T}$ in the second Horn clause to generate $(x_1 = y_1 \wedge x_2 = y_2) \implies x_3 = y_3$.
Here is another example from (Example 3 in [1]. Let $\alpha = \{f(x_1) + x_2 = x_3, f(y_1) + y_2 = y_3, x_2 = x_1, y_1 = x_2, x_3 \neq y_3\}$ in which $+, f, x_1, y_1$ are uncommon. Flattening would yield $f(x_1) = a_1, f(y_1) = a_2, a_1 + x_2 = x_3, a_2 + y_2 = y_3, x_2 = x_1, y_1 = x_2, x_3 \neq y_3$. Constant congruence would generate one nontrivial equivalence class: $\{x_1, x_2, y_1\}$ with representative x_2; all other constants have equivalence classes containing themselves and serve as their own representatives. Congruence by f makes $a_1 = a_2$ generating another equivalence class $\{a_1, a_2\}$; let a_1 be its representative.
A Horn clause is generated: $x_2 = y_2 \implies x_3 = y_3$ The disequation becomes: $x_3 = y_3 \implies \mathbf{F}$. Since x_1, y_1 cannot be eliminated, the interpolant generated is: $\{x_2 = y_2 \implies x_3 = y_3, x_3 = y_3 \implies \mathbf{F}\}$.

7.1 Lazy Approach: Pseudo-Interpolants

we are working on two ideas: (i) treat new common symbols introduced in flattening as place holders for terms in common symbols even though they may not appear in any β.[9] This way, it is possible to generate compact

[9] Recall that this definition is recursive.

interpolants. Using topological sorting, this can be done by presenting an interpolant as a finite ordered sequence of conditional replacement rules for new symbols using which, an interpolant can be generated when needed, along with equations and Horn clauses purely using common symbols. (ii) Otherwise conditionally eliminate all uncommon symbols successively starting with a minimal Horn clause to a maximum, generating an interpolant purely in terms of the original common symbols of α.

We will use the above example to illustrate the key ideas of the lazy approach.

An analysis of the above Horn clauses reveals that while there are Horn clauses to eliminate uncommon symbols n_1, n_2 but there is none for even conditionally eliminating v (since there is no Horn clause in which $v = x$ or $x = v$ for some constant x in its conclusion). Because of this observation, all equations and Horn clauses in which v appear can be deleted since they will not be used in further to generate an interpolant.

We are left with f-equations $f(n_1, n_2) = t$ and Horn clauses $1, 2, 3, 5, 6, 8$. n_1 can be eliminated conditionally using $2, 5$; n_2 can be eliminated conditionally using $3, 6$. After conditional elimination, an interpolant is generated that would not have any uncommon symbol, as illustrated above.

A lazy approach is to observe that both n_1 and n_2 can be expressed purely in common symbols of α since their respective Horn clauses do not upon other uncommon symbols. So they can be marked "common" now serving as place holders for expressions purely in common symbols. This labeling would make $f(n_1, n_2)$ to be a term in common symbols as well. Thus, $1, 2, 3, 5, 6, 8$ can be declared as an interpolant. We call such an interpolant expressed using place holders for expressions expressing using original common symbols of α as *pseudo interpolants.*

In general, given a finite set of Horn clauses in which every clause has at least one uncommon symbol, these Horn clauses can be presented in solved form depending upon an order in which elimination of uncommon symbols can be successively done generating a result only in common symbols, This order can be determined using dependencies of the use of uncommon symbols on each other. This is very similar to solved form for substitutions in unification problems. The situation here is more complex because of conditional substitutions whereas in the case of standard unification (over the empty theory), complete subterm sharing using a dag representation of fully shared subterms suffices to avoid exponential blow-up.

A pseudo-interpolant is a finite set of equations and conditional equations purely in common symbols along with a finite set of conditional substitutions and a total (could be partial) ordering on place holders (which are symbols not originally common to α but stand for terms purely in terms of original common symbols and other place holders lower than them in the ordering relation). The ordering captures the conditional dependency of uncommon symbols among themselves. The idea here is to determine the order in which uncommon symbols would be conditionally eliminated by Horn clauses in the conditional elimination step above.

Pseudo interpolants for the example discussed in the previous subsection would be the rules:$\{1, 2, 3, 5, 6, 8\}$ and $\{f(n_1, n_2) = t\}$ with the ordering $n_2 > n_1$ in which $2, 5$ are used to eliminate n_1 followed by $3, 6$ to eliminate n_2; the ordering $n_1 > n_2$ also works. The result would be the same as the interpolant given in the previous subsection.
A pseudo-interpolant is thus an intermediate form where some uncommon symbols are eliminated only if needed in an application where interpolants are needed.
Associate a function *rank* on uncommon symbols which determines an order in which uncommon symbols should be eliminated. The rank of a Horn clause is then the maximum over the ranks of all the uncommon symbols appearing in the clause. Common symbols will have the rank 0 thus making the rank of uncommon symbols to be > 0. Given a Horn clause $(c_1 = e_1 \wedge \cdots \wedge c_k = e_k) \implies d = h$, rank constraints are generated on uncommon symbols as follows: if any of d, h is an uncommon symbol, then its rank is bigger than the rank of every uncommon symbol in its hypotheses. Rank constraints of a Horn clause are identified using the Horn clause as a superscript. If there is no uncommon symbol in the conclusion of a Horn clause, no rank constraint is generated. If a Horn clause has only common symbols in its hypotheses, then rank constraints generated on uncommon symbols in its conclusion, if any, is > 0. A minimal subset of rank constraints that are contradictory gives a cycle of dependencies among Horn clauses which generated those rank constraints. These Horn clauses and the associated rank constraints can be deleted for further consideration and in generating an interpolant. In the above example, it would delete Horn clauses $9, 10$ from further consideration. If there is an uncommon symbol with no rank constraint (orphaned uncommon symbol), then all clauses and equations in which it appears can be eliminated from further consideration, e.g, the uncommon symbol v becomes orphaned after $9, 10$ are deleted. A satisfiable (cycle-free) subset can be assigned numeral ranks starting from 1 onwards. This however does not consider the case in which the equality on uncommon symbols in the conclusion of a Horn clause can be used to replace it in other Horn clauses. That can be done also deriving rank constraints on equalities or by performing this step first.
The remaining set of Horn clauses and equations constitute an interpolant I_α in which uncommon symbols with an ordering are viewed as place holders.
The result of the above steps after repeated applications is: (i) the set of equations, disequations and conditional equations purely in common symbols, (ii) conditional equations each of which has an uncommon symbol in its hypothesis and has been eliminated elsewhere.

7.2 Complexity Analysis

The flattening step can be done in $O(n)$ where n is the size of the DAG representation of the input terms (with full sharing). In general there are $O(n)$ constant symbols after flattening, corresponding to a constant for every node in the graph representation. The constant congruence step and associated processing of replacing constants by their representatives can be done easily in $O(n*log(n))$ using balanced trees (but its amortized complexity is $O(n*\alpha(n))$, almost linear assuming path compression is employed). Congruences using signature tables or any other means can take $O(m)$, where $m*log(m)$ is the number of edges in the DAG since in the worst case, every directed edge may have to be checked for equality against the corresponding edge.

Horn clauses of size k, where k is the maximum arity of a function symbol, (or constant size if all nonunary function symbols are encoded using binary function symbols as proposed in [18]), are generated from every pair of f-equations with the same function symbol.

The most expensive step is that of conditional elimination primarily because for a single uncommon symbol, there can be multiple Horn clauses equating the uncommon symbol under different conditions. This can result in an exponential blow-up if uncommon symbols are completely eliminated; the above example illustrates this.

In contrast, a pseudo-interpolant can be generated in $O(n^2)$. Perhaps this complexity can be improved substantially by exploiting the structure of Horn clauses. Ranking constraints and deletion of Horn clauses not useful for interpolant generation can be done in less than $O(n^2)$ time.

To our knowledge, this is the first complexity analysis of interpolant generation algorithms in the literature. Furthermore, the above complexity results are for generating the strongest possible interpolants from α without having access to a proof of $\alpha \implies \beta$ (which can in general be of higher complexity than interpolant generation if the complexity of proof generation is also included in the complexity analysis of interpolant generation).

Comparison with other interpolant generation algorithms We illustrate differences between our algorithm and those in [7, 16, 17, 6]. which generate interpolants from refutation proofs of $\alpha \wedge \neg\beta$. To compare the interpolants generated from our algorithm with those in the literature (particularly McMillan's [17] and Tinelli et. al's [6]), consider Example 3.1 in [6]: $\alpha = \{z_1 = x_1, x_1 = z_2, z_2 = x_2, x_2 = f(z_3), f(z_3) = x_3, x_3 = z_4, f(z_2) = x_2, x_2 = z_3\}$ and $\beta = \{z_1 = y_1, y_1 = f(z_2), f(z_2) = y_2, y_2 = z_3, z_3 = y_3, z_2 = y_2, y_2 = f(z_3), y_3 \neq z_4\}$. Common symbols are $\{f, z_1, z_2, z_3, z_4\}$. $\{x_1, x_2, x_3\}$ are to be eliminated from α. The input is already flattened. The constant congruence gives equivalence classes: $\{z_1, x_1, z_2, x_2, z_3\}$ and $\{x_3, z_4\}$; two f-equations $x_2 = f(z_3), f(z_3) = x_3$ have identical terms, equating $x_3 = x_2$ causing the merger of the above two equivalence classes; all constant symbols become equivalent. Wlog, z_1 can be chosen the representative of the equivalence class. The interpolant is: $\{z_2 = z_1, f(z_1) = z_1, z_4 = z_1, z_3 = z_1\}$, whereas McMillan's algorithm produces $\{z_1 = z_2, z_2 = f(z_3), f(z_3) = z_4\}$ (as reported in [6]), and Tinelli et al's algorithm gives $\{z_1 = z_4\}$. The interpolant generated by our algorithm implies the interpolants generated by both McMillan's as well as Tinelli et al's algorithms and is stronger.

Example 4.3 in [6] is also illustrative of differences among algorithms: $\alpha = \{x_1 = z_1, z_2 = x_2, z_3 = f(x_1), f(x_2) = z_4, x_3 = z_5, z_6 = x_4, z_7 = f(x_3), f(x_4) = z_8\}$ and $\beta = \{z_1 = z_2, z_5 = f(z_3), f(z_4) = z_6, y_1 = z_7, z_8 = y_2, y_1 \neq y_2\}$. Commons symbols are $\{f, z_1, z_2, z_3, z_4, z_5, z_6, z_7, z_8\}$ and $\{x_1, x_2, x_3, x_4\}$ are uncommon symbols to be eliminated from α. Using our algorithm, the constant equivalence relation is: $\{\{x_1, z_1\}, \{x_2, z_2\}, \{x_3, z_5\}, \{x_4, z_6\}, \{z_3\}\{z_4\}, \{z_7\}, \{z_8\}\}$, let $z_1, z_2, z_5, z_6, z_3, z_4, z_7, z_8$ respectively be the representatives of the equivalence classes. The f-equations are: $\{f(z_5) = z_7, f(z_6) = z_8, f(z_1) = z_3, f(z_2) = z_4\}$. No Horn clauses are generated as no f-equation is hiding any uncommon symbol. The interpolant is: $\{f(z_1) = z_3, f(z_2) = z_4, f(z_5) = z_7, f(z_6) = z_8\}$. The interpolant reported for McMillan's algorithm in [6] is: $(z_1 = z_2 \wedge (z_3 = z_4 \implies z_5 = z_6)) \implies (z_3 = z_4 \wedge z_7 = z_8)$ and for Tinelli et al's algorithm, it is $(z_1 = z_2 \implies z_3 = z_4) \wedge (z_5 = z_6 \implies z_7 = z_8)$. The interpolant by our algorithm is stronger and much simpler. If an application prefers interpolants expressed in constants, they can also be generated by eliminating common function symbols.

Our interpolant uses f since it is a common symbol as well. Suppose we wanted an interpolant without f, then we will generates Horn clauses thus eliminating f to give: $(z_1 = z_2 \implies z_3 = z_4) \wedge (z_1 = z_5 \implies z_3 = z_7) \wedge (z_1 = z_6 \implies z_3 = z_8) \wedge (z_2 = z_5 \implies z_4 = z_7) \wedge (z_2 = z_6 \implies z_4 = z_8) \wedge (z_5 = z_6 \implies z_7 = z_8)$. However, this unnecessarily increases the size of the interpolant as well as generating a weaker interpolant whereas this rule is necessary if the function symbol is uncommon and must be eliminated.

7.3 Properties of Interpolants Generated

First of all, it is easy to see that the above algorithm terminates. A proof of $\alpha \implies I_\alpha$ follows from the properties of various steps in the above algorithm, particularly that every step generates new formulas which are equisatisfiable with α or are implied by α.

Flattening generates a formula with new symbols that is equisatisfiable with α. Any interpolant generated from this formula is also an interpolant of α. Until Phase II, all steps of the algorithm are equivalence preserving and generate congruence classes induced by α.

The only interesting step is that of Horn clause generation.

Lemma 1. *$f(x_1, x_2) = y_1 \wedge f(x_3, x_4) = y_2$ implies $(x_1 = x_3 \wedge x_2 = x_4) \implies y_1 = y_2$. Further, $(f(x_1, x_2) = y_1 \wedge (x_1 = x_3 \wedge x_2 = x_4) \implies y_1 = y_2) \implies (x_1 = x_3 \wedge x_2 = x_4) \implies f(x_3, x_4) = y_2$ but it need not imply $f(x_3, x_4) = y_2$.* *None*

Proof. The only interesting part of the lemma is $(f(x_1, x_2) = y_1 \wedge (x_1 = x_3 \wedge x_2 = x_4) \implies y_1 = y_2)$ need not imply $f(x_3, x_4) = y_2$. Consider an interpretation in which $x_1 = a, x_3 = a, x_2 = a, a \neq b, x_4 = b, y_1 = a, y_2 = a$ and $f(a, a) =$

$a, f(a,b) = b$. We have: $(x_1 = x_3 \wedge x_2 = x_4) \implies y_1 = y_2 \wedge f(x_1, x_2) = y_1$ is true but $f(a,b) = a$ is false. None

Conditional elimination is validity preserving. At the end of Phase III and just before generating an interpolant, all equations and Horn clauses are derived from α.

The final interpolant generation step drops all equations and Horn clauses that have uncommon symbols. This symbol elimination step is approximate when f-equations are dropped due to uncommon symbols appearing in them because of the above lemma. Thus I_α is not equivalent to $\exists UC \alpha$ unless UC consists of only constant symbols none of which appears in function terms equivalent to subterms in α.

It must also be proved that $I_\alpha \wedge \beta$ is unsatisfiable. Let us attempt a proof by contradiction implying that $I_\alpha \wedge \beta$ are satisfiable, i.e., there exists an interpretation for all constants as well as function symbols in I_α and β so that I_α as well as β is true for that interpretation. We need to extend this interpretation to include interpretation of uncommon symbols eliminated from α to get I_α such that α is true in that interpretation which will contradict the assumption that $\alpha \wedge \beta$ is unsatisfiable.[10]

Any uncommon constant is given the same interpretation as that of a common constant or a common function term it became equivalent to in the algorithm; otherwise, it can be given any interpretation. Any function symbol (common or uncommon) f that does not appear in I_α must be given interpretation so that for all common terms $g_1, \cdots, g_k$ such that $f(g_1, \cdots, g_k)$ is equivalent to some other common term, the interpretation satisfies these properties. This is again feasible because α is satisfiable and these relations are implied by α. For a function symbol that is partially eliminated (i.e. Horn clause generation from f-equations with that outermost function symbol is done only if necessitated), it must also satisfy the constraints for all common terms serving as its argument in α. The fact that such an extended interpretation from I_α can be built with only constraints necessitated by the algorithm that makes α true is the crucial aspect of the construction which also established that that I_α generated by the above algorithm is indeed the strongest expressed in terms of a boolean combination of equalities over common symbols.

Any interpolant in EUF is in general a boolean combination of equalities on ground equalities. Without any loss of generality, it is a disjunction of conjunctions of such equalities and disequalities). Some of these disjuncts are interpolants since interpolants are closed under disjunctions. Consider any such disjunct that is an interpolant. Wlog, it can be assumed that any interpolant I is a conjunction of equalities and disequalities and that $\alpha \implies I$.

Showing that I_α is the strongest can be established similar to proofs above for propositional calculus and UTVI theory. For any interpolant I that is not

[10] There might be a close relationship between extending interpretation and amalgamation property extensively discussed in [2, 22] in the context of interpolation generation.

implied by I_α, we need to construct an interpretation for I_α which falsifies I, extend it so that it satisfies α and derive a contradiction since $\alpha \implies I$. We thus have:

Theorem 5. *Any interpolant I of a mutually contradictory (α, β) is implied by I_α.* *None*

Proof. Assume that there is indeed an interpolant I of (α, β) such that I_α does not imply I. This means $I_| alpha \wedge \neg I$ is satisfiable implying the existence of an interpretation of common symbols in α in which I_α is true but I is false. This interpretation is minimally extended to give interpretation to uncommon symbols in UC as well as some common function symbols which may have been completely eliminated or partially eliminated, so that it satisfies α. Since $\alpha \implies I$, the assumption that I is false in this interpretation gives a contradiction.None

The proof relies on the property of the above algorithm that a common function symbol is either not eliminated or if partially eliminated, its partial interpretation can be appropriately extended. As we show later for an example, the above algorithm does not eliminate common function symbols unlikes McMillan's and Tinelli's algorithm.

7.4 Comparison with McMillan's and Tinelli et. al.'s algorithms

Let us revisit Example 3.1 in Tinelli et al's paper. $\alpha = \{z_1 = x_1, x_1 = z_2, z_2 = x_2, x_2 = f(z_3), f(z_3) = x_3, x_3 = z_4, f(z_2) = x_2, x_2 = z_3\}$. The interpolant generated by the proposed algorithm is: $\{z_2 = z_1, f(z_1) = z_1, z_4 = z_1, f(z_2) = z_1, z_3 = z_1\}$. McMillan's algorithm (as reported in Tinelli et al's paper) produces $\{z_1 = z_2, z_2 = f(z_3), f(z_3) = z_4\}$, and Tinelli et al's algorithm gives $z_1 = z_4$. The interpolant generated by our algorithm implies the one generated by McMillan's algorithm since z_1, z_2, z_3, z_4 are all equivalent. It also implies the interpolant of Tinelli et al's algorithm. It is also easy to see that the interpolant from our algorithm is strictly stronger than both the interpolants as they do not imply our interpolant.

Revisiting Example 4.3 in Tinelli et al's paper, where $\alpha = \{x_1 = z_1, z_2 = x_2, z_3 = f(x_1), f(x_2) = z_4, x_3 = z_5, z_6 = x_4, z_7 = f(x_3), f(x_4) = z_8\}$. The uncommon symbols are: $\{x_1, x_2, x_3, x_4\}$ The interpolant generated by our algorithm is: $\{f(z_1) = z_3, f(z_2) = z_4, f(z_5) = z_7, f(z_6) = z_8\}$. The interpolant without f can also be generated as shown above: $(z_1 = z_2 \implies z_3 = z_4) \wedge (z_1 = z_5 \implies z_3 = z_7)\, amd (z_1 = z_6) \implies z_3 = z_8\, and (z_2 = z_5) \implies z_4 = z_7 \wedge (z_2 = z_6) \implies z_4 = z_8) \wedge (z_5 = z_6) \implies z_7 = z_8$

The interpolant generated by Tinelli et al's algorithm is $(z_1 = z_2 \implies z_3 = z_4) \wedge (z_5 = z_6 \implies z_7 = z_8)$, and is implied by our algorithm's interpolant and is strictly stronger. The interpolant reported for McMillan is: $(z_1 = z_2 \wedge (z_3 = z_4 \implies z_5 = z_6)) \implies (z_3 = z4 \wedge z_7 = z_8)$ is just too complicated, however it is also implied by our algorithm's interpolant which is strictly stronger.

Also consider Example 3.3 in the same paper:
$\alpha = \{x = z_1, x\dot{z}_2 = z_3\}, \beta = \{y = z_2, z_1.y \neq z_3\}$.
Common symbols are $\{;z_1, z_2, z_3\}$.
The rewrite system for α is $\{x \to z_1, z_1\dot{z}_2 = z_3\}$.
The interpolant is thus $z_1\dot{z}_2 = z_3$.
Example in Figure 4 of that paper:
$\alpha = \{x_1 = z_1, z_3 = f(x_1), f(z_2) = x_2, x_2 = z_4\}$.
$\beta = \{z_1 = y_1, y_1 = z_2, y_2 = z_3, z_4 = y_3, f(y_2) \neq f(y_3)\}$.
Common symbols are: $\{f, z_1, z_3, z_3, z_4\}$.
The rewrite system for generating congruence closure of α is:
$\{x_1 \to z_1, f(z_1) \to z_3, x_2 \to z_4, f(z_2) \to z_4\}$.
I_α is thus $\{f(z_1) = z_3, f(z_2) = z_4\}$, the same as the one produced by McMillan's algorithm; in contrast, Tinelli et al produced $z_1 = z_2 \implies z_3 = z_4$ which would be generated by our algorithm if an interpolant without f is desired.

8 Concluding Remarks and Future Work

Acknowledgments: The relationship between interpolation and quantifier elimination, although quite obvious in hindsight, was first explicitly pointed out by my former post doc Dr. Calogero Zarba which also led to our joint FSE paper [12].

References

1. M. P. Bonacina and M. Johansson. On interpolation in automated theorem proving. *J. Autom. Reasoning*, 54(1):69–97, 2015.
2. R. Bruttomesso, S. Ghilardi, and S. Ranise. From strong amalgamability to modularity of quantifier-free interpolation. In B. Gramlich, D. Miller, and U. Sattler, editors, *Automated Reasoning - 6th International Joint Conference, IJCAR 2012, Manchester, UK, June 26-29, 2012. Proceedings*, volume 7364 of *Lecture Notes in Computer Science*, pages 118–133. Springer, 2012.
3. A. Cimatti, A. Griggio, and R. Sebastiani. Interpolant generation for UTVPI. In R. A. Schmidt, editor, *Automated Deduction - CADE-22, 22nd International Conference on Automated Deduction, Montreal, Canada, August 2-7, 2009. Proceedings*, volume 5663 of *Lecture Notes in Computer Science*, pages 167–182. Springer, 2009.
4. W. Craig. Three uses of the herbrand-gentzen theorem in relating model theory and proof theory. *Journal of Symbolic Logic*, 22(3):269–285, 1957.
5. V. D'Silva, D. Kroening, M. Purandare, and G. Weissenbacher. Interpolant strength. In G. Barthe and M. V. Hermenegildo, editors, *Verification, Model Checking, and Abstract Interpretation, 11th International Conference, VMCAI 2010, Madrid, Spain, January 17-19, 2010. Proceedings*, volume 5944 of *Lecture Notes in Computer Science*, pages 129–145. Springer, 2010.

6. A. Fuchs, A. Goel, J. Grundy, S. Krstic, and C. Tinelli. Ground interpolation for the theory of equality. *Logical Methods in Computer Science*, 8(1), 2012.
7. A. Griggio. An effective smt engine for formal verification. *Ph.D. Thesis, University of Trento*, December, 2009.
8. S. Gulwani and M. Musuvathi. Cover algorithms and their combination. In *Programming Languages and Systems, 17th European Symposium on Programming, ESOP 2008, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2008, Budapest, Hungary, March 29-April 6, 2008. Proceedings*, pages 193–207, 2008.
9. G. Huang. Constructing craig interpolation formulas. In *Computing and Combinatorics* COCOON, pages 181–190. LNCS, 959, 1995.
10. R. Jhala and K. L. McMillan. A practical and complete approach to predicate refinement. In H. Hermanns and J. Palsberg, editors, *Tools and Algorithms for the Construction and Analysis of Systems, 12th International Conference, TACAS 2006 Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2006, Vienna, Austria, March 25 - April 2, 2006, Proceedings*, volume 3920 of *Lecture Notes in Computer Science*, pages 459–473. Springer, 2006.
11. D. Kapur. Shostak's congruence closure as completion. In H. Comon, editor, *Proc. Rewriting Techniques and Applications, 8th Intl. Conf., RTA-97*, pages 23–37, Sitges, Spain, June 1997. Springer LNCS 1231.
12. D. Kapur, R. Majumdar, and C. Zarba. Interpolation for data structures. In *Proceedings of the 14th ACM SIGSOFT Symp. on Foundations of Software Engineering*, 2006.
13. D. Kapur and C. Zarba. A Reduction Approach to Decison Procedures. *Technical Report, Department of Computer Science, UNM,*, Dec. 2006.
14. J. Krajicek. Interpolation theorems, lower bounds for proof systems and independence results for bounded arithmetic. *Journal of Symbolic Logic*, 62(2):457–486, 1997.
15. D. Kroening and G. Weissenbacher. An interpolating decision procedure for transitive relations with uninterpreted functions. In K. S. Namjoshi, A. Zeller, and A. Ziv, editors, *Hardware and Software: Verification and Testing - 5th International Haifa Verification Conference, HVC 2009, Haifa, Israel, October 19-22, 2009, Revised Selected Papers*, volume 6405 of *Lecture Notes in Computer Science*, pages 150–168. Springer, 2009.
16. K. L. McMillan. Applications of craig interpolants in model checking. In N. Halbwachs and L. D. Zuck, editors, *TACAS*, volume 3440 of *Lecture Notes in Computer Science*, pages 1–12. Springer, 2005.
17. K. L. McMillan. An interpolating theorem prover. *Theor. Comput. Sci.*, 345(1):101–121, 2005.
18. R. S. P.J. Downey and E. Tarjan. Variations on the common subexpression problem. *J. of the ACM*, pages 758–771, 1980.
19. P. Pudlack. Lower bounds for resolution and cutting plane proofs and monotone computations. *Journal of Symbolic Logic*, 62(3):981–988, 1997.
20. A. Rybalchenko and V. Sofronie-Stokkermans. Constraint solving for interpolation. *J. Symb. Comput.*, 45(11):1212–1233, 2010.
21. R. S.F., S. O., and S. N. Leveraging interpolant strength in model checking. In *Proc. Computer Aided Verification (CAV)*. Springer LNCS 7358, 2012.
22. V. Sofronie-Stokkermans. On interpolation and symbol elimination in theory extensions. In N. Olivetti and A. Tiwari, editors, *Automated Reasoning - 8th International Joint Conference, IJCAR 2016, Coimbra, Portugal, June 27 - July 2, 2016,*

Proceedings, volume 9706 of *Lecture Notes in Computer Science*, pages 273–289. Springer, 2016.