Duality-Based Interpolation for Quantifier-Free Equalities and Uninterpreted Functions

Leonardo Alt, Antti E. J. Hyvärinen, Sepideh Asadi, and Natasha Sharygina
Università della Svizzera italiana, Lugano, Switzerland
Email:firstname.lastname@usi.ch

***Abstract*—Interpolating, i.e., computing safe over-approximations for a system represented by a logical formula, is at the core of symbolic model-checking. One of the central tools in modeling programs is the use of the equality logic and uninterpreted functions (EUF), but certain aspects of its interpolation, such as size and the logical strength, are still relatively little studied. In this paper we present a solid framework for building compact, strength-controlled interpolants, prove its strength and size properties on EUF, implement and combine it with a propositional interpolation system and integrate the implementation into a model checker. We report encouraging results on using the interpolants both in a controlled setting and in the model checker. Based on the experimentation the presented techniques have potentially a big impact on the final interpolant size and the number of counter-example-guided refinements.**

I. Introduction

An important skill in constructing mathematical proofs is to identify the aspects of the problem that are relevant. When applied to formal reasoning about the correctness of software this means ignoring the parts of the system that play no role in its correctness. One such approach that works well in automated software verification based on satisfiability modulo theories (SMT) engines (see, e.g, [1]) is to employ the Equality Logic and Uninterpreted Functions (EUF) when applicable: in some cases it suffices to assume that a given function returns the same value when invoked with the same arguments. This technique is particularly useful, for example, when modeling memory or arrays [2], proving program equivalence [3], or as a technique for avoiding flattening in solving bit-vector problems [4], [5].

Generalizing a formula over the states reachable by a program is a natural subtask when summarizing the behavior of a procedure [6], or computing a fixed-point of a transition function [7], [8]. These techniques are now popular in software model-checking [9], [10], and together with the theory-based abstraction result in a growing interest in an over-approximation technique known as *interpolation*.

In this paper we present the *EUF-interpolation system* which aims at specializing and tailoring interpolants for the needs of interpolation-based model-checking. The paper contributes to the state-of-the-art by (i) providing the first approach for controlling the strengths of EUF interpolants; (ii) identifying a strength lattice of interpolation algorithms; and (iii) proving under certain assumptions the size order for the interpolants produced by the system. In addition we (iv) provide an implementation of the system; (v) integrate and experiment with the system on a model checker; and (vi) study the combination of labeled interpolation systems for EUF and propostitional logic. The EUF-interpolation operates on the proof of unsatisfiability in EUF based on a recursive algorithm for building a final interpolant from partial interpolants and uses *duality* of interpolants, a logical relation between an interpolant and its negation discussed below, to control the strength of the constructed partial and final interpolants.

The system is implemented in the SMT solver OpenSMT2 [11], and used in a model-checking algorithm based on the interpolating incremental C verifier HiFrog [6]. This gives us the advantage of making a direct connection between the theoretical contributions and practice. We evaluate the efficiency of the EUF-interpolation system with two major experiments. In the first experiment we verify a set of C software verification problems produced by HiFrog, and in the second experiment we study different combinations of propositional and EUF interpolation algorithms on a set of instances from the SMT-LIB benchmark collection. Based on the results the system has a big impact on the generated interpolants, and the interpolants seem to be very useful in our application to model-checking. To the best of our knowledge our work is the first to consider the duality of interpolants in constructing EUF interpolants recursively, and to report experiments with EUF interpolation together with incremental verification.

a) Related work : Recent work on *labeled interpolation systems* (LIS) addresses interpolation in propositional logic [12], [13], [14], [15] by providing control over fitting the interpolant strength and size to particular model-checking applications. Our approach extends the work on propositional interpolation to SMT theories and in particular to EUF. Interpolation procedures for EUF have been introduced in [16], [17]. The interpolation procedure given in [16] provides a way of computing a single interpolant from a given proof. The technique is extended in [17] to allow construction of several interpolants through the coloring of congruence graphs edges. Our work differs fundamentally from both these approaches by using duality for controlling the interpolant strength, a feature not available in earlier formalizations.

The parametric interpolation frameworks presented in [18] and [19] generalize first-order interpolation procedures. The former provides labeled interpolation systems for hyper-resolution proofs which are then extended to first order interpolation systems for local proofs; the latter generalizes the former further to non-local proofs. Both of these techniques provide control on the propositional level. Unlike ours, they are not specialized and optimized for EUF and, to the best of our knowledge, have not been implemented.

Other orthogonal procedures exist for the quantifier-free fragments of the theories of linear integer arithmetics [20], [21], linear real arithmetics [16], [22], [23], and Arrays [24], while [25] provides a labeled interpolation system for Non-linear Real Arithmetics. On a high level, we believe that the duality-based approach followed in this work can be applied also in these fields.

This paper is organized as follows: Sec. II presents a general algorithmic framework for interpolation as a preliminary for the EUF-interpolation system. The main result on the EUF-interpolation system is presented in Sec. III The experiments are reported in Sec. IV, and the paper concludes in Sec. V. For lack of space we present most of the proofs of the theorems in Appendix A. The implementation and more experimental results are available at http://verify.inf.usi.ch/euf-interpolation.

II. Preliminaries

This paper considers the extension of propositional logic to Boolean variables that are interpreted as equalities over uninterpreted functions. Following [26], we call the extension the theory of equality logic and uninterpreted functions (EUF) For example $\neg(a = b) \vee f(a) = f(b)$ is an EUF formula containing the uninterpreted functions a, b, and f, embedded in a Boolean structure. Given an EUF formula F, we call the equality ($=$), and the Boolean connectives (e.g. $\neg, \wedge, \vee$) the *logical symbols*, while the Boolean variables and uninterpreted functions are its *non-logical symbols*, denoted by $\mathit{Vars}(F)$.

Given an unsatisfiable conjunction $A \wedge B$ of EUF formulas A and B, an *interpolation instance* is a pair (A, B), and an *interpolant* for (A, B) is a formula $I(A, B)$ such that *(i)* $A \rightarrow I(A, B)$, *(ii)* $I(A, B) \wedge B$ is unsatisfiable, and *(iii)* $\mathit{Vars}(I(A, B)) \subseteq \mathit{Vars}(A) \cap \mathit{Vars}(B)$. In general several interpolants can be computed for an instance (A, B). We denote an algorithm computing an interpolant $I(A, B)$ by $\mathit{Itp}(A, B)$, and, with a slight abuse of the notation, use $\mathit{Itp}(A, B)$ to denote the interpolant $I(A, B)$ when the interpolation algorithm needs to be specified. A concept central to this paper is the duality between interpolation algorithms: Given an interpolation algorithm $\mathit{Itp}(A, B)$, also the algorithm $\mathit{Itp}^-(A, B)$ returning $\neg \mathit{Itp}(B, A)$ computes an interpolant for (A, B), as can be seen from the following reasoning: By definition, $\mathit{Itp}^-(A, B) = \neg \mathit{Itp}(B, A)$. $\mathit{Itp}(B, A)$ satisfies (i) $B \rightarrow \mathit{Itp}(B, A)$; (ii) $\mathit{Itp}(B, A) \rightarrow \neg A$; and (iii) $\mathit{Vars}(\mathit{Itp}(B, A)) \subseteq \mathit{Vars}(A) \cap \mathit{Vars}(B)$. By rewriting, from (ii) follows that (iv) $A \rightarrow \neg \mathit{Itp}(B, A)$, and from (i) that (v) $\neg \mathit{Itp}(B, A) \rightarrow B$. From (iii), commutativity of intersection, and definition of non-logical symbols, follows (vi) $\mathit{Vars}(\neg \mathit{Itp}(B, A)) \subseteq \mathit{Vars}(B) \cap \mathit{Vars}(A)$.

In this work we consider algorithms that build interpolants based on the unsatisfiability proof of $A \wedge B$. We make this explicit by denoting the interpolation algorithm (and the resulting interpolant) by $\mathit{Itp}(A, B, R)$, where R is the *refutation* representing the proof of unsatisfiability. In this work we are particularly interested in ordering interpolation algorithms with respect to the strength of the interpolants they compute. An interpolant I is *stronger* than an interpolant I' if $I \rightarrow I'$. We extend the strength relation to interpolation algorithms: if $\mathit{Itp}^s(A, B, R) \rightarrow \mathit{Itp}^w(A, B, R)$ for algorithms Itp^s and Itp^w for all interpolation instances (A, B) with a fixed refutation R for $A \wedge B$, then Itp^s is stronger than Itp^w. If the strength relation can be established between the algorithms Itp and Itp^-, we call the algorithm computing the stronger interpolant the *base* and the weaker the *dual interpolation algorithm* and denote them by Itp and Itp', respectively.

Algorithm 1 Congruence closure

1: **procedure** CongruenceClosure(T, Eq)
2: Initialize $E \leftarrow \emptyset$ and $G \leftarrow (T, E)$
3: **repeat** pick $x, y \in T$ such that $(x \not\sim y)$
4: **if** (a) $(x = y) \in \mathit{Eq}$ or
5: (b) x is $f(x_1, \ldots, x_k)$, y is $f(y_1, \ldots, y_k)$, and
6: $(x_1 \sim y_1), \ldots, (x_k \sim y_k)$ **then**
7: $E \leftarrow E \cup \{(x, y)\}$
8: **until** no such x, y can be chosen so that E would grow
9: **return** G

A. EUF Preliminaries

This section describes our interpolation system for EUF. The presentation is based on [17] and uses the congruence graph as the refutation.

Many EUF solvers rely on the *congruence closure* algorithm [27] to decide the satisfiability of a set of equalities and disequalities. The algorithm, described in Alg. 1, takes as input a finite set Eq of equalities, and the set T of subterm-closed terms over which Eq is defined. During the execution the algorithm builds an undirected *congruence graph* G using the set T as nodes. We write $(x \sim y)$ if there is a path in G connecting x and y and denote this path by $\overline{xy}$.

Theorem 1 (c.f. [27]): Let S be a set of EUF disequalities $x \neq y$ over the terms T. The set $S \cup \mathit{Eq}$ is satisfiable if and only if the congruence graph G constructed by CongruenceClosure(T, Eq) has no path $(x \sim y)$ such that $(x \neq y) \in S$.

During the creation of G, an edge (x, y) is added only if $(x \sim y)$ does not hold, which ensures that G is acyclic. Therefore, for any pair of terms x and y such that $(x \sim y)$ holds in G, the path $\overline{xy}$ connecting these terms is unique. Empty paths are represented by $\overline{xx}$. For an arbitrary path π, we use the notation $[\![\pi]\!]$ to represent the equality of the terms that π connects. If, for example, $\pi = \overline{xy}$, then $[\![\pi]\!] := (x = y)$. We also extend this notation over sets of paths P so that $[\![P]\!] := \bigwedge_{\sigma \in P} [\![\sigma]\!]$.

An edge may be added to a congruence graph G because of two different reasons in Alg. 1 at line 7. Edges added because of Condition (a) are called *basic*, while edges added because of Condition (b) are called *derived*. Let e be a derived edge

$(f(x_1, \ldots, x_k), f(y_1, \ldots, y_k))$. The k *parent paths* of e are $\overline{x_1 y_1}, \ldots, \overline{x_k y_k}$. Given a congruence graph G and two terms x, y such that $x \sim y$ we denote by $G[\overline{xy}]$ the congruence graph obtained from the graph G by including the edges and terms that appear on the path $\overline{xy}$ and recursively all its parent paths.

Let A and B be two sets of equalities and disequalities. A *coloring* C of a congruence graph $G = (E, T)$ created over the equalities and terms of $A \cup B$ is a function $C : E \to \{a, b\}$, that is, C assigns a color a or b to each edge, considering two restrictions: *(i)* basic edges e must be colored with a if $e \in A$ and with b if $e \in B$; and *(ii)* if an edge has color $\kappa \in \{a, b\}$, both its endpoints must be κ-colorable. ab-colorable derived edges can be colored arbitrarily. To compute the interpolant the congruence graph needs to be annotated with the information on which equalities and formulas belong to which partition. Equalities and formulas are *a-colorable* if all their non-logical symbols occur in A, and *b-colorable* if all their non-logical symbols occur in B. They are *colorable* if they are a or b-colorable, and *ab-colorable* if both. An edge (x, y) of a congruence graph has the same color as the equality $(x = y)$. A path in a congruence graph is colorable if all its edges are colorable, and a congruence graph is colorable if all its edges are colorable.

While it is possible to construct a non-colorable congruence graph, the following lemma and its constructive proof in [17] state that we may assume without loss of generality that congruence graphs are colorable.

Lemma 1 (c.f. [17]): Let (A, B) be an interpolation instance over EUF. If x and y are colorable terms and if $A, B \models (x = y)$, then there exist a term set T and a colorable congruence graph over the equalities contained in $A \cup B \cup T$ in which $(x \sim y)$.

We denote a congruence graph G colored with a function C by G^C. A path is called an *a-path* if all its edges are colored a, and a *b-path* if all its edges are colored b. A *factor* of a path in G^C is a maximal subpath such that all its edges have the same color. Notice that every path is uniquely represented as a concatenation of the consecutive factors of opposite colors.

Example 1: Let $A := \{(v_1 = f(y_1)), (f(y_2) = v_2), (y_1 = t_1), (t_2 = y_2), (s_1 = f(r_1)), (f(r_2) = s_2), (r_1 = u_1), (u_2 = r_2)\}$ and $B := \{(x_1 = v_1), (v_2 = x_2), (t_1 = f(z_1)), (f(z_2) = t_2), (z_1 = s_1), (s_2 = z_2), (u_1 = u_2), (x_1 \neq x_2)\}$. Figure 1 shows a colored congruence graph G^C built while proving the unsatisfiability of A and B with Alg. 1. The curvy edges with the labels *s* or *w* in G^C are not relevant for this example and are used later in Section III. The congruence graph G^C demonstrates the joint unsatisfiability of A and B, since it proves $(x_1 = x_2)$ and $(x_1 \neq x_2)$ is an original term. Edges are represented by thick lines, and dotted arrows point to the parents of derived edges. We present a-colorable nodes (terms) and a-colored edges by black circles and solid lines, b-colorable nodes and b-colored edges by white circles and dashed lines, and ab-colorable nodes by gray circles. In the first (top) path of G^C, we see that basic edges (original equalities from $A \cup B$) are used to prove $(r_1 = r_2)$. This fact is used to infer $(f(r_1) = f(r_2))$, which is in turn used as a derived edge in the path below, proving $(z_1 = z_2)$. The equality $(f(z_1) = f(z_2))$ is then inferred and used to prove $(y_1 = y_2)$ in the path below. In the last (bottom) path of G^C, the derived edge representing $(f(y_1) = f(y_2))$ is created and finally $(x_1 = x_2)$ is proved.

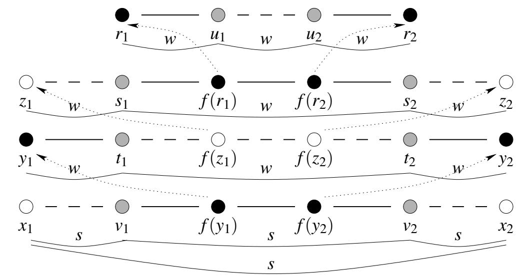

Figure 1. Congruence graph G^C that proves the unsatisfiability of $A \cup B$

III. The EUF Interpolation System

In this section we present the EUF-interpolation system which extends the approach described in [17] with a modular use of dual interpolants. The main novelty of the system the control over the interpolant strength. Due to lack of space all the proofs of the theorems in this section are presented in Appendix A.

Intuitively, the approach computes partial interpolants with either a base or a dual interpolation algorithm using the structure of a congruence graph. We show that while interpolating on a fixed congruence graph the liberty in choosing between the two interpolation algorithms allows computing several interpolants that can be partially ordered with respect to their strength. To make this choice explicit we introduce the *labeling functions* L for the EUF-interpolation interpolation system, and the algorithm Itp_L for computing the interpolants.

Definition 1 (Labeling function): Let $G[\overline{xy}]^C$ be a colored congruence graph and W its factors. A *labeling function* $L : W \cup \{\overline{xy}\} \to \{s, w\}$ labels the factors and the path corresponding to the conflict $x \neq y$ as s or w.

We want to emphasize that colors, described in Sec. II-A, and labels are different concepts. The colors a, b tell if a node or edge in a congruence graph belongs to A or B, whereas labels s, w are used while deciding whether to use the standard or the dual interpolant.

Given an (unsatisfiable) interpolation instance (A, B), an EUF interpolation algorithm $Itp_L(A, B, G[\overline{xy}]^C)$ computes an interpolant for (A, B); $G[\overline{xy}]^C$ is a congruence graph with coloring C; $\overline{xy}$ a path such that $(x \sim y)$ is in G and the disequality $(x \neq y)$ exists in $A \cup B$; and L is a labeling function. We omit A, B, G^C and L when they are clear from the context, referring to the interpolation algorithm and the corresponding interpolant as $Itp(\overline{xy})$. We define separately two constant labeling functions $L_s(\sigma) = L_s = s$ and $L_w(\sigma) = L_w = w$ that will be useful in the following analysis.

The interpolation algorithms in [16] and [17] essentially compute an interpolant by collecting the A-factors that prove $(x = y)$ in G^C. To maintain the unsatisfiability with the B

part of the problem, the A factors will then be implied by their *B-premise set*. A *premise set* for a color is the set of equalities of the opposite color justifying the existence of a parent edge. More technically, the B-premise set $\mathcal{B}$ for a path π is

$$\mathcal{B}(\pi) := \begin{cases} \bigcup\{\mathcal{B}(\sigma) | \sigma \text{ is a factor of } \pi\}, \text{if } \pi \text{ has} \geq 2 \text{ factors;} \\ \{\pi\}, \text{if } \pi \text{ is a } B\text{-path; and} \\ \bigcup\{\mathcal{B}(\sigma) | \sigma \text{ is a parent path of an edge of } \pi\}, \\ \quad \text{if } \pi \text{ is an } A\text{-path.} \end{cases} \tag{1}$$

As stated in Sec. II, it is also possible to compute a dual interpolant for A as the negation of an interpolant for B. To compute the dual interpolant we will need similarly to collect the B-factors that prove $(x = y)$ in G^C, implied by their A-premise set. The A-premise set $\mathcal{A}$ for a path π is defined as

$$\mathcal{A}(\pi) := \begin{cases} \bigcup\{\mathcal{A}(\sigma) | \sigma \text{ is a factor of } \pi\}, \text{if } \pi \text{ has} \geq 2 \text{ factors;} \\ \{\pi\}, \text{if } \pi \text{ is an } A\text{-path; and} \\ \bigcup\{\mathcal{A}(\sigma) | \sigma \text{ is a parent path of an edge of } \pi\}, \\ \quad \text{if } \pi \text{ is a } B\text{-path.} \end{cases} \tag{2}$$

We extend the notation of $\mathcal{A}$ and $\mathcal{B}$ over a set S of paths as $\mathcal{A}(S) := \bigcup_{\sigma \in S} \mathcal{A}(\sigma)$ and $\mathcal{B}(S) := \bigcup_{\sigma \in S} \mathcal{B}(\sigma)$. For any two operators $\mathcal{O}, \mathcal{O}'$ such that domain of $\mathcal{O}$ contains the range of $\mathcal{O}'$ we define the composite operator $\mathcal{O}\mathcal{O}'(\sigma) := \mathcal{O}(\mathcal{O}'(\sigma))$; and define recursively $\mathcal{O}^0(\sigma) := \sigma$, and $\mathcal{O}^n := \mathcal{O}(\mathcal{O}^{n-1})$.

The functions J_A and J_B give, respectively, the contribution of an individual A-factor and an individual B-factor to the interpolants.

$$J_A(\pi) := [\![\mathcal{B}(\pi)]\!] \to [\![\pi]\!] \tag{3}$$

$$J_B(\pi) := [\![\mathcal{A}(\pi)]\!] \to [\![\pi]\!] \tag{4}$$

Let S be a set of paths. The notation $S|_\nu$ represents the subset of S containing the paths σ such that $L(\sigma) = \nu$ for $\nu \in \{s, w\}$. Let (A, B) be an EUF interpolation instance, G the corresponding congruence graph, and $x \neq y \in A \cup B$ that is in conflict with G. Let $P = (A, B, G[\overline{xy}]^C)$. The algorithm $Itp_L(P)$ computes the EUF interpolant over A for a path $\overline{xy}$. It is defined using four sub-procedures I_A, I'_A, I_B, and I'_B that map congruence graphs to partial interpolants, and are invoked depending on which partition the conflict $x \neq y$ belongs to and what label the path $\overline{xy}$ has:

$$Itp_L(P) := \begin{cases} I_A(\overline{xy}) & \text{if } (x \neq y) \in B \text{ and } L(\overline{xy}) = s, \\ I'_A(\overline{xy}) & \text{if } (x \neq y) \in A \text{ and } L(\overline{xy}) = s, \\ \neg I_B(\overline{xy}) & \text{if } (x \neq y) \in A \text{ and } L(\overline{xy}) = w, \text{ and} \\ \neg I'_B(\overline{xy}) & \text{if } (x \neq y) \in B \text{ and } L(\overline{xy}) = w. \end{cases} \tag{5}$$

The sub-procedures for I_A and I_B are defined as

$$I_A(\pi) := \bigwedge_{\sigma \in \mathcal{A}(\pi)} J_A(\sigma) \wedge \bigwedge_{\sigma \in \mathcal{BA}(\pi)|_s} I_A(\sigma) \bigwedge_{\sigma \in \mathcal{BA}(\pi)|_w} \neg I'_B(\sigma) \tag{6}$$

and

$$I_B(\pi) := \bigwedge_{\sigma \in \mathcal{B}(\pi)} J_B(\sigma) \wedge \bigwedge_{\sigma \in \mathcal{AB}(\pi)|_w} I_B(\sigma) \wedge \bigwedge_{\sigma \in \mathcal{AB}(\pi)|_s} \neg I'_A(\sigma). \tag{7}$$

For the cases where either the conflict $x \neq y \in A$ and $L(\overline{xy}) = s$, or the conflict $x \neq y \in B$ and $L(\overline{xy}) = w$, the path $\overline{xy} = \pi$ needs to be decomposed for computing the partial interpolant as $\pi_1\theta_b\pi_2$ or $\pi_1\theta_a\pi_2$, where θ_κ is the longest subpath of π with κ-colorable endpoints. Hence, I'_A and I'_B are

$$\begin{aligned} I'_A(\pi) := I_A(\theta_b) \wedge \bigwedge_{\sigma \in \mathcal{B}(\pi_1) \cup \mathcal{B}(\pi_2)} I_A(\sigma) \\ \wedge ([\![\mathcal{B}(\pi_1) \cup \mathcal{B}(\pi_2)]\!] \to \neg[\![\theta_b]\!]), \end{aligned} \tag{8}$$

and

$$\begin{aligned} I'_B(\pi) := I_B(\theta_a) \wedge (\bigwedge_{\sigma \in \mathcal{A}(\pi_1) \cup \mathcal{A}(\pi_2)} I_B(\sigma)) \\ \wedge([\![\mathcal{A}(\pi_1) \cup \mathcal{A}(\pi_2)]\!] \to \neg[\![\theta_a]\!]). \end{aligned} \tag{9}$$

Theorem 2: Given two sets of equalities and disequalities A and B such that $A \cup B$ is unsatisfiable, a colored congruence graph G^C containing a path $\pi := \overline{xy}$ such that $(x \neq y) \in A \cup B$, and a labeling function L, Eq. (5) computes a valid interpolant for A using L over G^C.

The following example shows how Eq. (5) can be used to compute the interpolants from [17].

Example 2: Let $A := \{(x_1 = f(x_2)), (f(x_3) = x_4), (x_4 = f(x_5)), (f(x_6) = x_7)\}$ and $B := \{(x_2 = x_3), (x_5 = x_6), (x_1 \neq x_7)\}$. Figure 2 shows a possible congruence graph G^C that proves the joint unsatisfiability of A and B (by proving $(x_1 = x_7)$ such that $(x_1 \neq x_7) \in A \cup B$) and its tree representation, with each node annotated by its partial interpolant. In this example we use the constant labeling function $L_s = s$. From Eq. (5) we have that $Itp(\overline{x_1x_7}) = I_A(\overline{x_1x_7})$, because $L_s(\overline{x_1x_7}) = s$ and $(x_1 \neq x_7) \in B$. The call to $I_A(\overline{x_1x_7})$ is represented by the root node in the tree in Figure 2. First we compute $\mathcal{A}(\overline{x_1x_7}) = \{\overline{x_1x_7}\}$ and $\mathcal{BA}(\overline{x_1x_7}) = \{\overline{x_2x_3}, \overline{x_5x_6}\}$. Then from Eq. (6) we have that $I_A(\overline{x_1x_7}) = J_A(\overline{x_1x_7}) \wedge I_A(\overline{x_2x_3}) \wedge I_A(\overline{x_5x_6})$. The calls to $I_A(\overline{x_2x_3})$ and $I_A(\overline{x_5x_6})$ are represented by the edges from the leaf nodes to the root in the tree in Figure 2. We then proceed computing $\mathcal{A}(\overline{x_2x_3}) = \emptyset$ and $\mathcal{BA}(\overline{x_2x_3}) = \emptyset$ which lead to $I_A(\overline{x_2x_3}) = \top$; and $\mathcal{A}(\overline{x_5x_6}) = \emptyset$ and $\mathcal{BA}(\overline{x_5x_6}) = \emptyset$ which lead to $I_A(\overline{x_5x_6}) = \top$ (the partial interpolants of the leaf nodes). Finally we have that $I_A(\overline{x_1x_7}) = ((x_2 = x_3) \wedge (x_5 = x_6)) \to (x_1 = x_7)$ is the partial interpolant of the root node, representing the final interpolant for A.

A. The Interpolant Strength

Let $P = (A, B, G[\pi]^C)$ and L_s and L_w the weak and the strong labeling functions. We will show in Th. 3 that $Itp_{L_s}(P) \to Itp_{L_w}(P)$, and then in Ex. 3 that there are cases where the strength relation is strict in the sense that there are models that satisfy $Itp_{L_w}(P)$ but do not satisfy $Itp_{L_s}(P)$. Theorem 3 needs Lemma 4 which in turn is a generalization of Lemma 2. We then show our main result on EUF in Theorem 4, that is, we provide a way to compare

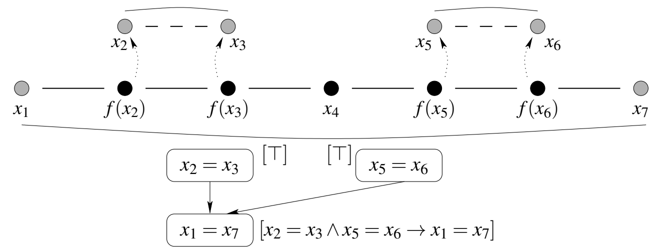

Figure 2. Computing partial interpolants for the EUF-interpolation system.

the strength of interpolants based on the labeling functions used.

Lemma 2: Let G^C be a congruence graph with coloring C, and ω a factor from G. Then $I_A(\omega) \wedge I_B(\omega) \to [\![\omega]\!]$.

Lemma 3: Let π be an arbitrary path in the congruence graph, and $\phi(\pi)$ the set of all factors in π. Then $I_A(\pi) = \bigwedge_{\sigma \in \phi(\pi)} I_A(\sigma)$ and $I_B(\pi) = \bigwedge_{\sigma \in \phi(\pi)} I_B(\sigma)$.

Lemma 4: Lemma 2 holds when ω is a path containing multiple factors.

Theorem 3: For fixed A, B, and $G[\overline{xy}]^C$, for the corresponding interpolants defined in Eq. (5) it holds that $Itp_{L_s}(A, B, G[\overline{xy}]^C) \to Itp_{L_w}(A, B, G[\overline{xy}]^C)$.

To show that the implication is not trivial in general, we show by example that three different labeling functions being applied to the congruence graph from Ex. 1 result in three pairwais inequal interpolants.

Example 3: Consider again the sets A and B and the congruence graph G^C from Ex. 1 and Fig. 1. Let L_c be a custom labeling function mapping the paths to labels as $\{\overline{x_1 x_2} \mapsto s, \overline{x_1 v_1} \mapsto s, \overline{v_1 v_2} \mapsto s, \overline{v_2 x_2} \mapsto s, \overline{y_1 t_1} \mapsto w, \overline{t_1 t_2} \mapsto w, \overline{t_2 y_2} \mapsto w, \overline{z_1 s_1} \mapsto w, \overline{s_1 s_2} \mapsto w, \overline{s_2 z_2} \mapsto w, \overline{r_1 u_1} \mapsto w, \overline{u_1 u_2} \mapsto w, \overline{u_2 r_2} \mapsto w\}$. We recall that the labeling function only needs to be defined on the factors and the path that contradicts the original disequality, in this case $\overline{x_1 x_2}$. The labels are shown over curves representing which path is being labeled. The labeling function L_c represents the intent of generating stronger partial interpolants closer to $(x_1 = x_2)$, and weaker partial interpolants in the inner explanations. Let Itp_s, Itp_w and Itp_c be, respectively, the interpolants generated by Eq. (5) by using the labeling functions L_s, L_w and L_c. The computed interpolants are $Itp_s = ((t_1 = t_2) \to (v_1 = v_2)) \wedge ((u_1 = u_2) \to (s_1 = s_2))$, $Itp_w = \neg((u_1 = u_2) \wedge ((s_1 = s_2) \to (t_1 = t_2)) \wedge \neg(v_1 = v_2))$, and $Itp_c = ((t_1 = t_2) \to (v_1 = v_2)) \wedge \neg(((s_1 = s_2) \to (t_1 = t_2)) \wedge (u_1 = u_2) \wedge \neg(t_1 = t_2))$. The reader is welcome to verify that $Itp_s \to Itp_c \to Itp_w$, and none of them is equivalent to another.

Finally we present our main result providing a way to partially order interpolation algorithms into a lattice based on their strength. From this follows that the constant labeling functions L_s and L_w give, respectively, the strongest and the weakest interpolants within this framework.

Theorem 4: Let $\sqsupseteq$ be a strength relation defined over the labels s and w such that $s \sqsupseteq s$, $w \sqsupseteq w$ and $s \sqsupseteq w$. Let (A, B) be an interpolation instance, G^C a congruence graph proving the unsatisfiability of $A \wedge B$, and L and L' two labeling functions such that $L(\sigma) \sqsupseteq L'(\sigma)$ for all the factors σ of G^C. Then $Itp_L(A, B, G^C) \to Itp_{L'}(A, B, G^C)$.

B. Interpolant Size

The EUF-interpolation system presented above introduces a way of computing interpolants of different strength by labeling the factors of a congruence graph as s or w, depending on the required strength. Each labeling function results potentially in a different interpolant, and creating meaningful labeling functions is a challenging task on its own. For the labeling functions L_s and L_w we give the following results with respect to their size.

Theorem 5: Let $P = (A, B, G^C)$. The interpolant with the smallest number of equalities over all interpolants computable with the EUF interpolation system is $Itp_{L_s}(P)$ if $\pi \in B$ and $Itp_{L_w}(P)$ if $\pi \in A$.

The EUF interpolation system presented in this section provides a rich platform that allows the adjustment of interpolants to particular tasks. In the following we will study how the approach works in practice.

IV. Experiments

We implemented and integrated the EUF interpolation system together with propositional interpolation into the OpenSMT2 solver and HiFrog, an interpolation-based incremental model checker for C [6]. We report experiments in two different settings in the implementation: running the approach (i) integrated in HiFrog; and (ii) over unsatisfiable EUF benchmarks from SMT-LIB (i.e., the QF_UF benchmarks). The benchmarks and the software are available at http://verify.inf.usi.ch/euf-interpolation. Before describing the experiments we give a concise explanation on how EUF and propositional interpolation are integrated.

A. Integration of Propositional and EUF Interpolation.

An SMT solver takes as input a propositional formula where some atoms are interpreted as equalities or inequalities over a theory, that in our experiments is the theory of equalities over uninterpreted functions. If a satisfying truth assignment for the propositional structure is found, a theory solver is queried to determine the consistency of its equalities. In case of inconsistency the theory solver adds a reason-entailing clause to the propositional structure. The process ends when either a theory-consistent truth assignment is found or the propositional structure becomes unsatisfiable. The SMT framework provides a natural integration for the theory and propositional interpolants. The clauses provided by the theory solver are annotated with their theory interpolant and are used as partial interpolants in the propositional interpolation system (see, e.g., [15]). Similar to EUF, the propositional interpolation algorithms control the strength of the resulting interpolant by choosing the partition for the shared variables through labeling [15]. The labeling has to be followed then by the theory interpolation algorithm to preserve interpolant soundness. In the following experiments we use instances

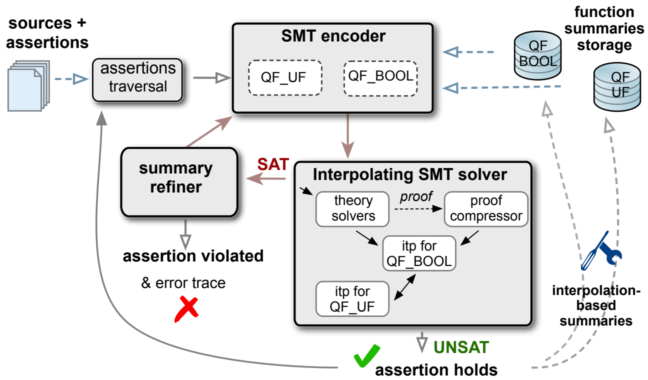

Figure 3. HiFrog overview

of the propositional labeled interpolation system [28], [15] supported by OpenSMT2, and in particular the McMillan's algorithms M_s and M_w [7], the Pudlák's algorithm P [29], and the proof-sensitive algorithms PS, PS_s, and PS_w [15] that use the proof structure to optimize the labeling. Fig. 4 shows the algorithms ordered with respect to the logical strength of the interpolants they compute.

B. *Interpolation-Based Incremental Verification*

We integrated the EUF-interpolation system with the incremental model checker HiFrog as part of OpenSMT2, and used it to verify a set of C benchmarks from SV-COMP (https://sv-comp.sosy-lab.org/) and other sources. In total we checked 973 verification conditions over these problems in the experiments.

We use both purely propositional logic and QF_UF to model the programs. The incremental C model checker HiFrog attempts to prove or refute the validity of a sequence of verification conditions using an SMT solver and an encoding in EUF or in bit-precise propositional logic. Figure 3 shows HiFrog's verification flow. The problem instance is first preprocessed and then encoded into an instance of a decision problem in SMT. An SMT solver computes whether the assertion holds by determining the satisfiability of the instance. If the instance is unsatisfiable, the assertion holds, and interpolation is used to extract function summaries from the proof. These summaries are then stored and used in lieu of the precise encoding of a function to incrementally verify the consequent assertions. If the instance is satisfiable, the witnessing truth assignment corresponds to an execution violating the assertion. However, due to the over-approximative nature of both EUF and the function summaries, the execution might be spurious. In this case the model checker uses the precise encoding instead of the summaries to decide the correct answer.

Table I gives an overview of our results. In parentheses after the names we mark the total number of assertions in the instance. The table shows the verification time for HiFrog with propositional logic in the column *Bool*; and with EUF in the columns marked *EUF Time*. Unlike the bit-precise propositional model, the EUF model provides an over-approximation of the program behavior. If HiFrog reports that a safety property is true under EUF it is also true for the propositional model. However, if a property is reported false,

Table I
SUMMARY OF VERIFICATION RESULTS ON A SET OF C BENCHMARKS.

	EUF Results				EUF Time (s)		
Name (asrts)	**Corr**	**SAT**	**Sp**	**Bool**	**EUF**	**Sp**	**Full**
floppy1 (18)	15	3	3	69.6	8.3	34.7	34.7
floppy2 (21)	18	3	3	192.1	42.8	122.5	122.5
kbfiltr1 (10)	10	0	0	4.1	1.3	1.3	1.3
diskperf1 (14)	11	3	3	193.7	20.5	67.8	67.8
floppy3 (19)	16	4	3	76.2	9.6	36.4	43.7
kbfiltr2 (13)	13	0	0	10.2	3.1	3.1	3.1
floppy4 (22)	19	4	3	207.3	46.7	127.9	144.1
kbfiltr3 (14)	14	1	0	18.7	5.6	5.6	14.6
tcas_asrt (162)	149	145	13	86.0	16.7	21.6	100.0
cafe (115)	100	100	15	19.2	4.2	5.8	14.7
s3 (131)	123	112	8	1.5	1.6	1.8	3.0
mem (149)	146	52	3	44.6	59.9	60.0	78.5
ddv (152)	56	105	96	260.3	11.6	122.0	122.9
token (54)	54	20	0	962.3	150.6	150.6	998.6
disk (79)	62	72	17	8195.0	237.6	638.2	8151.2

Table II
INTERPOLATION ALGORITHM COMPARISON ON A SET OF C BENCHMARKS.

	$\mathbf{M_s + Itp_s}$		$\mathbf{M_s + Itp_w}$		$\mathbf{M_w + Itp_s}$		$\mathbf{M_w + Itp_w}$	
Name	**t**	**refs**	**t**	**refs**	**t**	**refs**	**t**	**refs**
floppy1	10.9	28672	9.8	27648	**8.3**	**24320**	12.7	32256
floppy2	58.9	**37120**	64.8	41216	**46.7**	37632	59.6	40704
kbfiltr1	1.5	**4864**	1.5	**4864**	**1.3**	**4864**	1.5	**4864**
diskperf1	30.1	45568	**20.5**	**44544**	29.7	47104	26.0	48384
floppy3	**9.6**	28928	13.6	34304	**9.6**	**26624**	10.8	29952
kbfiltr2	**3.0**	**4864**	3.1	**4864**	3.1	**4864**	**3.0**	**4864**
floppy4	57.2	41472	**46.7**	43008	48.6	**40704**	58.6	43776
kbfiltr3	**5.7**	**10240**	6.5	10496	5.6	**10240**	6.3	10496
tcas_asrt	17.2	**59648**	16.8	60160	**16.7**	**59648**	17.3	60160
cafe	**4.2**	**6656**	4.3	**6656**	**4.2**	**6656**	4.3	6656
s3	1.7	**0**	1.7	**0**	**1.6**	**0**	**1.6**	**0**
mem	60.1	**23808**	**59.9**	25088	60.7	**23808**	60.1	25088
ddv	**11.2**	**7936**	11.6	**7936**	11.6	**7936**	11.7	**7936**
token	151.4	15616	151.0	**13568**	152.8	15616	**150.6**	**13568**
disk	**237.6**	**9472**	241.3	38912	240.4	**9472**	246.4	38912
Total	660.3	324684	653.1	363264	**640.9**	**319488**	670.5	367616

it may indicate either a real or a spurious counterexample introduced by the EUF abstraction. In case of false properties the model checker can for instance consult the propositional encoding to get the correct result. The three columns under the label *EUF Results* list, from left to right, the number of correctly identified assertions using EUF encoding, the number of reachable assertions, and how many of the reachable assertions were spurious. The table reports run times for three variations of the model checker. Column *EUF* reports the time used only by the EUF check. Column *Sp* reports the time when HiFrog is allowed to query the spuriousness of the counter-example from an oracle and only needs to consult the propositional encoding if the answer is yes. Column *Full* reports the time when HiFrog needs to resort to the propositional encoding always in case of a failure to verify. Notably the use of EUF as an abstraction technique usually speeds up the solving even in the case of the full overhead.

Finally we report the effect of interpolation algorithm strength to the number of required refinements and the run time for the four combinations $M_s + Itp_s$, $M_s + Itp_w$, $M_w + Itp_s$ and $M_w + Itp_w$ in Table II. We note that the number of summary refinements varies sometimes considerably over the combinations, demonstrating the advantage of the flexibility our framework provides for the EUF-interpolation. The number of summary refinement shows the total number of function summaries that were used in whole verification process, did not work, and were replaced by precise encoding of functions, hence the smaller number is, the more efficient is the solving process. The best-performing algorithm in this benchmark set is $M_w + Itp_s$ with both lowest total run time and the lowest total number of refinements. We note that the run time and the number of refinements do not always correlate, and that in particular the combination $M_s + Itp_s$ works very well with respect to refinements while losing nevertheless clearly in total run time.

Our experiments with a model checker solving real life problems shows two main results. The first is that using EUF to represent software instead of only Boolean formulas is beneficial, and leads to an impressive speed up in verification time. The second result, is that it is important to fine tune the interpolation algorithms used for Boolean and EUF interpolation in order to ultimately optimize convergence in the model checker.

C. Interpolation over SMT-LIB Benchmarks

In addition to integrating the EUF interpolation into the model checker, we also report a more controlled set of experiments on generating interpolants of different strength and size. We computed interpolants from over 2000 benchmarks from the QF_UF category of SMT-LIB, and report here the results of 106 benchmarks that resulted in non-trivial interpolation instances having complex EUF proofs with large congruence graphs. In total this set contains over two and a half million individual EUF interpolants. Following [17], [30], we randomly split the assertions in each benchmark to partitions A and B.

a) Logical strength: The theory interpolation algorithms use three labeling functions L_s, L_w (see Sec. III), and L_r, a labeling function that labels all components randomly as either s or w. The algorithms are called, respectively, Itp_s, Itp_w, and Itp_r. We use the proof-sensitive interpolation algorithm [15] in the propositional structure. This results in three final interpolants I_s, I_w and I_r for each benchmark.

We computed the strength relationship for each theory partial interpolant as well as the final SMT interpolants. Even though the EUF interpolants are often simple, in 71% of them it was possible to generate at least two interpolants of different strength, and 5.7% resulted in all three having different strength.

After solving and interpolating, we ran extra experiments to check the strength relations of the final interpolants I_s, I_w and I_r. Since the final interpolants are much more complex, of the 106 benchmarks, 55 ran out of memory while computing the strength relations. For the remaining 51, all the three final interpolants were pairwise inequivalent, confirming that the framework is able to generate interpolants of different strength.

b) Interpolant size: Since the propositional and EUF interpolation algorithms are to a large degree independent, it is natural to ask what combination of the algorithms is most efficient. This experiment studies the question using the interpolant size as a measure of efficiency. The six propositional and three EUF interpolation algorithms result in 18 combinations. We measure the sizes of the final interpolants both in *(i)* the number of Boolean connectives (Fig. 5); and *(ii)* the number of EUF equalities (Fig. 6). Excluding the instances where we encountered memory outs we report the results on 82 of the original 106 benchmarks. For each benchmark, we computed the smallest number of Boolean connectives or equalities in the interpolant among all the configurations (*best*) and the ratio *combination/best* for each possible combination, which shows us how much worse each combination did compared to the best combination for that benchmark. Notice that the ratio of the best combination for a benchmark is one and therefore no ratio can be less than one. The bars present the average and the crosses the median of those ratios among all the benchmarks for each combination.

In Fig. 5 the combination M_w + Itp_w gives the smallest number of Boolean connectives, and M_s + Itp_s appears in the second place. The median of M_w + Itp_w is 1, which means that it was responsible for the smallest number of connectives in at least half of the benchmarks, and its average of 1.2 shows that even when this was not the case, the combination was still close to the optimum. On the losing side, we make two observations. The EUF interpolation algorithm Itp_r leads to a larger number of Boolean connectives, and the propositional interpolation algorithm P leads to larger interpolants.

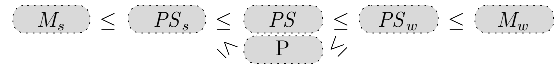

Figure 4. The relative strength of the propositional interpolation algorithms [15].

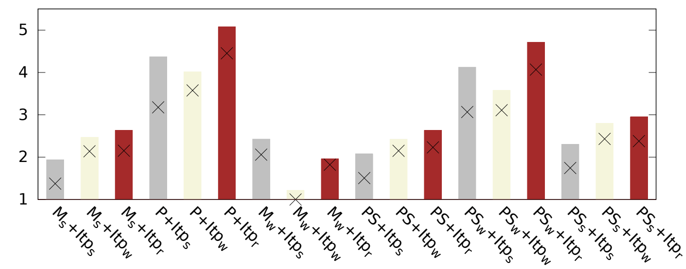

Figure 5. Comparison between interpolation combinations with respect to the number of Boolean connectives in the final interpolant

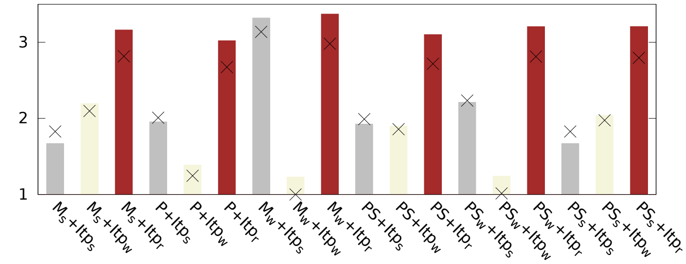

Figure 6. Comparison between interpolation combinations with respect to the number of equatilies in the final interpolant

Interestingly the combinations $PS + Itp_s$ and $PS_s + Itp_s$ have low medians and average, which are good, but not the best. This seemingly contradicts our earlier observation in [15] that PS and PS_s consistently lead to small number of connectives in the interpolant. Based on the experiments the likely reason is the soundness restriction in integration (see Sec. IV-A), since the results get gradually worse when the propositional and the EUF interpolation algorithms disagree more on the labeling, best being $PS_s + Itp_s$ and the worst $PS_w + Itp_s$.

The same trend is seen in Fig. 6 in the number of EUF equalities. A strong propositional interpolation algorithm (M_s, PS_s) combined with Itp_s leads to smaller interpolants compared to their combination with Itp_w; and a weak propositional interpolation algorithm (M_w, PS_w) combined with Itp_w leads to smaller interpolants compared to their combination with Itp_s. Interestingly PS, a propositional interpolation algorithm that tends to balance the distribution of variables [15], leads to very similar results when combined with Itp_s and Itp_w.

Our experiments with interpolation over complex SMT benchmarks show that the interpolants generated by the EUF system presented in this work indeed have strictly different logical strength. Moreover, in the combination of Boolean and EUF interpolants, it is important to match the strength of the used interpolation algorithms in order to reduce the size of the generated interpolants.

V. Conclusions

We present and analyse a new interpolation framework for the theory of Equalities and Uninterpreted Functions, capable of generating interpolants of different strength and small size in a controlled way. The technique bases on the use of dual partial interpolants parameterized by a labeling function. We confirm the analysis with experiments and show the feasibility of generating multiple interpolants of different strengths. In addition, we report on the size of the created interpolants, comparing different combinations of propositional and EUF interpolation algorithms. Our major contribution work is the integration of a complete interpolation-based model checker to the system, and showing the significant impact the interpolant strength has on both run time and convergence.

In the future we intend to generalize the approach to be applicable to other theories, and study the effects of different labeling functions on fix-point computation in other model-checking applications.

References

[1] D. Detlefs, G. Nelson, and J. B. Saxe, "Simplify: A theorem prover for program checking," vol. 52, no. 3, pp. 365–473, 2005.

[2] A. Stump, C. W. Barrett, D. L. Dill, and J. R. Levitt, "A decision procedure for an extensional theory of arrays," in *Proc. LICS 2001*. IEEE Computer Society, 2001, pp. 29–37.

[3] B. Godlin and O. Strichman, "Regression verification: proving the equivalence of similar programs," *Softw. Test., Verif. Reliab.*, vol. 23, no. 3, pp. 241–258, 2013.

[4] R. Bruttomesso, A. Cimatti, A. Franzén, A. Griggio, Z. Hanna, A. Nadel, A. Palti, and R. Sebastiani, "A lazy and layered SMT($\mathcal{BV}$) solver for hard industrial verification problems," in *Proc. CAV 2007*, ser. LNCS, vol. 4590. Springer, 2007, pp. 547 – 560.

[5] L. Hadarean, K. Bansal, D. Jovanović, C. Barret, and C. Tinelli, "A tale of two solvers: Eager and lazy approaches to bit-vectors," in *Proc. CAV 2014*, ser. LNCS. Springer, 2014, pp. 680 – 695.

[6] L. Alt, S. Asadi, H. Chockler, K. E. Mendoza, G. Fedyukovich, A. E. J. Hyvärinen, and N. Sharygina, "HiFrog: SMT-based function summarization for software verification," in *Proc. TACAS 2017*, ser. LNCS, vol. 10206, 2017, pp. 207–213.

[7] K. L. McMillan, "Interpolation and SAT-based model checking," in *Proc. CAV 2003*, ser. LNCS, vol. 2725. SV, 2003, pp. 1–13.

[8] A. R. Bradley, "SAT-based model checking without unrolling," in *Proc. VMCAI 2011*, ser. LNCS, vol. 6538. Springer, 2011, pp. 70–87.

[9] D. Beyer and M. E. Keremoglu, "CPAchecker: A tool for configurable software verification," in *Proc. CAV 2011*, ser. LNCS, vol. 6806. Springer, 2011, pp. 184–190.

[10] A. Gurfinkel, T. Kahsai, A. Komuravelli, and J. A. Navas, "The SeaHorn verification framework," in *Proc. CAV 2015*, ser. LNCS, vol. 9206. Springer, 2015, pp. 343–361.

[11] A. E. J. Hyvärinen, M. Marescotti, L. Alt, and N. Sharygina, "OpenSMT2: An SMT solver for multi-core and cloud computing," in *Proc. SAT 2016*, ser. LNCS, vol. 9710. Springer, 2016, pp. 547–553.

[12] V. D'Silva, "Propositional interpolation and abstract interpretation," in *Proc. ESOP 2010*, ser. LNCS, vol. 6012. Springer, 2010, pp. 185–204.

[13] S. F. Rollini, O. Sery, and N. Sharygina, "Leveraging interpolant strength in model checking," in *Proc. CAV 2012*, ser. LNCS, vol. 7358. Springer, 2012, pp. 193–209.

[14] S. F. Rollini, L. Alt, G. Fedyukovich, A. E. J. Hyvärinen, and N. Sharygina, "PeRIPLO: A framework for producing effective interpolants in sat-based software verification," in *Proc. LPAR 2013*, ser. LNCS, vol. 8312. Springer, 2013, pp. 683–693.

[15] L. Alt, G. Fedyukovich, A. E. J. Hyvärinen, and N. Sharygina, "A proof-sensitive approach for small propositional interpolants," in *Proc. VSTTE 2015*, ser. LNCS, vol. 9593. Springer, 2016, pp. 1–18.

[16] K. L. McMillan, "An interpolating theorem prover," *Theor. Comput. Sci.*, vol. 345, no. 1, pp. 101–121, 2005.

[17] A. Fuchs, A. Goel, J. Grundy, S. Krstic, and C. Tinelli, "Ground interpolation for the theory of equality," *Logical Methods in Computer Science*, vol. 8, no. 1, 2012.

[18] G. Weissenbacher, "Interpolant strength revisited," in *Proc. SAT 2012*, ser. LNCS, vol. 7317. Springer, 2012, pp. 312–326.

[19] L. Kovács, S. F. Rollini, and N. Sharygina, "A parametric interpolation framework for first-order theories," in *Proc. MICAI 2013*, ser. LNCS, vol. 8265, 2013, pp. 24–40.

[20] A. Brillout, D. Kroening, P. Rümmer, and T. Wahl, "An interpolating sequent calculus for quantifier-free Presburger arithmetic," *Journal of Automated Reasoning*, vol. 47, no. 4, pp. 341–367, 2011.

[21] A. Griggio, T. T. H. Le, and R. Sebastiani, "Efficient interpolant generation in satisfiability modulo linear integer arithmetic," in *Proc. TACAS 2011*, ser. LNCS, vol. 6605. Springer, 2011, pp. 143–157.

[22] A. Rybalchenko and V. Sofronie-Stokkermans, "Constraint solving for interpolation," in *Proc. VMCAI 2007*, ser. LNCS, vol. 4349. Springer, 2007, pp. 346–362.

[23] A. Albarghouthi and K. L. McMillan, "Beautiful interpolants," in *Proc. CAV 2013*, ser. LNCS, vol. 8044. Springer, 2013, pp. 313–329.

[24] R. Bruttomesso, S. Ghilardi, and S. Ranise, "Quantifier-free interpolation of a theory of arrays," *Logical Methods in Computer Science*, vol. 8, no. 2, 2012.

[25] S. Gao and D. Zufferey, "Interpolants in nonlinear theories over the reals," in *Proc. TACAS 2016*, ser. LNCS, vol. 9636. Springer, 2016, pp. 625–641.

[26] D. Kroening and O. Strichman, *Decision Procedures - An Algorithmic Point of View, Second Edition*, ser. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2016.

[27] R. Nieuwenhuis and A. Oliveras, "Proof-producing congruence closure," in *Proc. RTA 2005*, ser. LNCS, vol. 3467. Springer, 2005, pp. 453–468.

[28] V. D'Silva, D. Kroening, M. Purandare, and G. Weissenbacher, "Interpolant strength," in *Proc. VMCAI 2010*, ser. LNCS, vol. 5944. Springer, 2010, pp. 129–145.

[29] P. Pudlák, "Lower bounds for resolution and cutting plane proofs and monotone computations," *Journal of Symbolic Logic*, vol. 62, no. 3, pp. 981–998, 1997.

[30] A. Cimatti, A. Griggio, and R. Sebastiani, "Efficient interpolant generation in satisfiability modulo theories," in *Proc. TACAS 2008*, ser. LNCS, vol. 4963. Springer, 2008, pp. 397–412.

APPENDIX

Theorem 2. *Given two sets of equalities and disequalities A and B such that $A \cup B$ is unsatisfiable, G^C a colored congruence graph containing a path $\pi := \overline{xy}$ such that $(x \neq y) \in A \cup B$ and a labeling function L, Eq. (5) computes a valid interpolant for A using L over G^C.*

Proof: In order to analyze Eq. (5) we have to analyze Eq. (8), Eq. (9), Eq. (6) and Eq. (7).

(i) $I'_A(\pi)$

Eq. (8) is the same presented in [17], and computes interpolants for A when the disequality $(x \neq y)$ is in A.

(ii) $\neg I'_B(\pi)$

Eq. (9) is the dual of Eq. (8), and clearly computes interpolants for B when the disequality is in B. We can transform it into an interpolant for A by negating it, as it is done in Eq. (5).

(iii) $I_A(\pi)$

Eq. (6) behaves similarly to the interpolation procedure presented in [17], with the addition of dual interpolants and labeling functions. First J_I computes the individual contribution of the A-factors that prove π in G^C ($\mathcal{A}(\pi)$) to the interpolant, and then conjoins it with the interpolants of their B-premise sets ($\mathcal{BA}(\pi)$).

(iv) $\neg I_B(\pi)$

Eq. (7) is the dual of Eq. (6) and computes an interpolant for B, which can be transformed into an interpolant for A by negating it. ■

Definition 2: Let G be a congruence graph, and σ an arbitrary factor from G. We say that σ is *relevant* to ω if either $J_A(\sigma)$ or $J_B(\sigma)$ is called during the computation of $I_A(\omega)$ or $I_B(\omega)$.

Lemma 2. *Let G^C be a congruence graph with coloring C, and ω a factor from G. Then $I_A(\omega) \wedge I_B(\omega) \rightarrow [\![\omega]\!]$.*

Proof: Let R^ω be the set of factors relevant (Def. 2) to ω in a congruence graph, R^ω_A the subset of R^ω containing only A-factors and R^ω_B the subset of R^ω containing only B-factors.

From Eq. (6) we can clearly see that

$$R^\omega_A := \mathcal{A}(\omega) \cup \mathcal{A}(\mathcal{BA}(\omega)) \cup \ldots \cup \mathcal{A}((\mathcal{BA})^k(\omega)) \cup \ldots, \quad (10)$$

and from Eq. (7) that

$$R^\omega_B := \mathcal{B}(\omega) \cup \mathcal{B}(\mathcal{AB}(\omega)) \cup \ldots \cup \mathcal{B}((\mathcal{AB})^k(\omega)) \cup \ldots. \quad (11)$$

Because congruence graphs are acyclic and finite, for any ω there exists an integer n such that $\mathcal{A}((\mathcal{BA})^n(\omega)) = \emptyset$ and $\mathcal{B}((\mathcal{AB})^n(\omega)) = \emptyset$, which allows us to rewrite the previous equations as

$$R^\omega_A := \mathcal{A}(\omega) \cup \mathcal{A}(\mathcal{BA}(\omega)) \cup \ldots \cup \mathcal{A}((\mathcal{BA})^n(\omega)), \quad (12)$$

$$R^\omega_B := \mathcal{B}(\omega) \cup \mathcal{B}(\mathcal{AB}(\omega)) \cup \ldots \cup \mathcal{B}((\mathcal{AB})^n(\omega)). \quad (13)$$

If ω is an A-factor, we have that $\mathcal{A}(\omega) = \{\omega\}$ and therefore we can write $\mathcal{B}(\omega) \equiv \mathcal{BA}(\omega)$. Using that we can infer that $\mathcal{A}((\mathcal{BA})^{n+1}(\omega)) = \emptyset$ (by the definition of n), and we can change Eq. (13) to

$$R^\omega_B := \mathcal{BA}(\omega) \cup (\mathcal{BA})^2(\omega)) \cup \ldots \cup (\mathcal{BA})^{n+1}(\omega). \quad (14)$$

We follow the proof assuming that ω is an A-factor, using Eq. (12) and Eq. (14) to represent R^ω_A and R^ω_B respectively, and then argue that the proof is symmetrical for the case where ω is a B-factor.

From Eq. (12) and Eq. (14), we can then see that

$$\begin{aligned} R^\omega := \mathcal{A}(\omega) \cup \mathcal{BA}(\omega) \cup \mathcal{A}(\mathcal{BA}(\omega)) \cup \ldots \cup \\ \mathcal{A}((\mathcal{BA})^n(\omega)) \cup (\mathcal{BA})^{n+1}(\omega) \end{aligned} \quad (15)$$

and therefore

$$I_A(\omega) \wedge I_B(\omega) = (\bigwedge_{\sigma \in R^\omega_A} J_A(\sigma)) \wedge (\bigwedge_{\sigma \in R^\omega_B} J_B(\sigma)). \quad (16)$$

When J_A and J_B are computed over a set that has an empty premise set, the result is not a conjunction of implications, but a conjunction of equalities. In this case, $J_B((\mathcal{BA})^{n+1}(\omega)) = [\![(\mathcal{BA})^{n+1}(\omega)]\!]$ because $\mathcal{A}((\mathcal{BA})^{n+1}(\omega)) = \emptyset$. Thus, after applying the functions J_A and J_B we have that

$$\begin{aligned} I_A(\omega) \wedge I_B(\omega) = \ & \left(\bigwedge_{i=0}^n \bigwedge_{\sigma \in \mathcal{A}((\mathcal{BA})^i(\omega))} ([\![\mathcal{B}(\sigma)]\!] \rightarrow [\![\sigma]\!])\right) \\ & \wedge \left(\bigwedge_{i=0}^n \bigwedge_{\sigma \in (\mathcal{BA})^i(\omega)} ([\![\mathcal{A}(\sigma)]\!] \rightarrow [\![\sigma]\!])\right) \\ & \wedge [\![(\mathcal{BA})^{n+1}(\omega)]\!]. \end{aligned} \quad (17)$$

We know that formulas of the form $((a_1 \rightarrow b_1) \wedge \ldots \wedge (a_n \rightarrow b_n)) \rightarrow ((\bigwedge_{i \in 1..n} a_i) \rightarrow (\bigwedge_{i \in 1..n} b_i))$ are tautologies. Using that and Eq. (17), we can then show that

$$\begin{aligned} I_A(\omega) \wedge I_B(\omega) \ & \rightarrow \left(\bigwedge_{i=0}^n ([\![(\mathcal{BA})^{i+1}(\omega)]\!] \rightarrow [\![\mathcal{A}((\mathcal{BA})^i(\omega))]\!])\right) \\ & \wedge \left(\bigwedge_{i=0}^n ([\![\mathcal{A}((\mathcal{BA})^i(\omega))]\!] \rightarrow [\![(\mathcal{BA})^i(\omega)]\!])\right) \\ & \wedge [\![(\mathcal{BA})^{n+1}(\omega)]\!] \end{aligned} \quad (18)$$

Because $[\![(\mathcal{BA})^{n+1}(\omega)]\!]$ has no implicant, it has to be true in the formula, starting a chain that satisfies the antecedents of all the implications in Eq. (18), since $(\mathcal{BA})^{n+1}(\omega)$ is the premise set of $\mathcal{A}((\mathcal{BA})^n(\omega))$, which is the premise set of $(\mathcal{BA})^n(\omega)$ and so on. Because of that, we can simplify the formula to

$$I_A(\omega) \wedge I_B(\omega) \rightarrow \left(\bigwedge_{i=0}^{n} [\![\mathcal{A}((\mathcal{BA})^i(\omega))]\!]\right) \wedge \left(\bigwedge_{i=0}^{n+1} [\![(\mathcal{BA})^i(\omega)]\!]\right). \quad (19)$$

Therefore, we have that

$$\forall r \in R^\omega.(I_A(\omega) \wedge I_B(\omega)) \rightarrow [\![r]\!]. \quad (20)$$

For the case where ω is a B-factor, we have that $\mathcal{B}(\omega) = \{\omega\}$, which implies that $\mathcal{AB}(\omega) \equiv \mathcal{A}(\omega)$. Using that, we can infer that $\mathcal{B}((\mathcal{AB})^{n+1}(\omega)) = \emptyset$ and we can also change Eq. (12) to

$$R^\omega_A := \mathcal{AB}(\omega) \cup (\mathcal{AB})^2(\omega) \cup \ldots \cup (\mathcal{AB})^n(\omega). \quad (21)$$

Using Eq. (21) and Eq. (13) to represent R^ω_A and R^ω_B respectively, the same reasoning is followed and we can show that Eq. (20) holds also if ω is a B-factor.

If ω is an A-factor, then $\mathcal{A}(\omega) = \{\omega\}$ and J_A is called for ω in the first iteration of Eq. (6). On the other hand, if ω is a B-factor, then $\mathcal{B}(\omega) = \{\omega\}$ and J_B is called for ω in the first iteration of Eq. (7). This shows that ω is a relevant factor and by Eq. (20) we conclude the proof. ■

Lemma 3. *Let π be an arbitrary path in the congruence graph, and $\phi(\pi)$ the set of all factors in π. Then $I_A(\pi) = \bigwedge_{\sigma\in\phi(\pi)} I_A(\sigma)$ and $I_B(\pi) = \bigwedge_{\sigma\in\phi(\pi)} I_B(\sigma)$.*

Proof: By the definition of $\mathcal{A}$ in Eq. (2), we know that $\mathcal{A}(\pi) = \bigcup_{\sigma\in\phi(\pi)} \mathcal{A}(\sigma)$. By the definition of I_A, we can see that J_A is computed individually for each element of $\mathcal{A}(\pi)$ in $I_A(\pi)$, and I_A is called recursively for the B-premise sets of each individual element of $\mathcal{A}(\pi)$. Therefore, $\bigwedge_{\sigma\in\phi(\pi)} I_A(\sigma) = \bigwedge_{\sigma\in\mathcal{A}(\pi)} I_A(\sigma)$, which has the same effect of $I_A(\pi)$. The result is analogous for $I_B(\pi)$. ∎

Lemma 4. *Lemma 2 holds when ω is a path containing multiple factors.*

Proof: Let ω be a path built by multiple factors and $\phi(\omega)$ the set containing these factors. By Lemma 3 we know that $I_A(\omega) = \bigwedge_{\sigma\in\phi(\omega)} I_A(\sigma)$ and $I_B(\omega) = \bigwedge_{\sigma\in\phi(\omega)} I_B(\sigma)$; by Lemma 2 we know that $\forall\sigma \in \phi(\omega).(I_A(\sigma) \wedge I_B(\sigma)) \rightarrow [\![\sigma]\!]$. Because the elements of $\phi(\omega)$ are factors linking nodes to prove ω, we know that $(\bigwedge_{\sigma\in\phi(\omega)}[\![\sigma]\!]) \rightarrow [\![\omega]\!]$. Therefore we have that $(I_A(\omega) \wedge I_B(\omega)) \rightarrow [\![\omega]\!]$. ∎

Theorem 3. *For fixed A, B, G^C, and $\overline{xy}$, for the corresponding interpolants defined in Eq. (5) it holds that $Itp(A, B, G^C, \overline{xy}, L_s) \rightarrow Itp(A, B, G^C, \overline{xy}, L_w)$.*

Proof: We only consider the case where $(x \neq y) \in B$ and note that the case where $(x \neq y) \in A$ is completely symmetrical. Let $\pi := \overline{xy}$ and $\psi = I_A(\pi) \wedge I'_B(\pi)$. By Eq. (9) we have that

$$\begin{aligned}\psi = \quad & I_B(\theta_A)\wedge \\ & \left(\textstyle\bigwedge_{\sigma\in\mathcal{A}(\pi_1)\cup\mathcal{A}(\pi_2)} I_B(\sigma)\right) \wedge \\ & ([\![\mathcal{A}(\pi_1) \cup \mathcal{A}(\pi_2)]\!] \rightarrow \neg[\![\theta_A]\!]) \wedge I_A(\pi),\end{aligned} \tag{22}$$

where π is decomposed as $\pi_1\theta_A\pi_2$, and θ_A is the largest subpath of π with A-colorable endpoints. In order to show that $I_A(\pi) \rightarrow \neg I'_B(\pi)$, we prove that $\psi \rightarrow \bot$, which leads to the theorem. In $I'_B(\pi)$, π is split into $\pi_1\theta_A\pi_2$. From the definition of θ_A, we know that π_1 and π_2 are B-factors. Therefore, using Lemma 3, we can say that $I_A(\pi) = I_A(\pi_1) \wedge I_A(\theta_A) \wedge I_A(\pi_2)$. Because π_1 and π_2 are subpaths of π, we know that $\mathcal{A}(\pi_1), \mathcal{A}(\pi_2) \subseteq \mathcal{A}(\pi)$. Using Lemma 3, we have that $(I_A(\pi_1) \wedge I_A(\pi_2)) = \bigwedge_{\sigma\in\mathcal{A}(\pi_1)} I_A(\sigma) \wedge \bigwedge_{\sigma\in\mathcal{A}(\pi_2)} I_A(\sigma)$.

We can now see that both $\bigwedge_{\sigma\in\mathcal{A}(\pi_1)\cup\mathcal{A}(\pi_2)} I_B(\sigma)$ and $\bigwedge_{\sigma\in\mathcal{A}(\pi_1)\cup\mathcal{A}(\pi_2)} I_A(\sigma)$ are contained in ψ. From Lemma 2 we know that *(i)* $\psi \rightarrow [\![\mathcal{A}(\pi_1) \cup \mathcal{A}(\pi_2)]\!]$. $I_A(\theta_A)$ and $I_B(\theta_A)$ are also contained in ψ, therefore from Lemma 2 we have that *(ii)* $\psi \rightarrow [\![\theta_A]\!]$. From *(i)* and *(ii)* we see that $\psi \rightarrow ([\![\theta_A]\!] \wedge \neg[\![\theta_A]\!])$. ∎

Theorem 4. *Let $\sqsupseteq$ be a label strength operator such that $s \sqsupseteq s$, $w \sqsupseteq w$ and $s \sqsupseteq w$. Let $Itp_L = (A, B, G[\overline{xy}]^C)$ and $Itp_{L'} = (A, B, G[\overline{xy}]^C)$ be two interpolation algorithms where L and L' are two labeling functions such that $L(\sigma) \sqsupseteq L'(\sigma)$ for all the factors σ of G^C. Then $Itp_L \rightarrow Itp_{L'}$.*

Proof: If $(x \neq y) \in B$, we can either use I_A or $\neg I_B$ to create an interpolant for A. On the other hand, if $(x \neq y) \in A$, we can use either I'_A or $\neg I_B$. Since only Eq. (6) and Eq. (7) use labeling functions, we analyze only those equations in this proof.

From Eq (6) we can see that when a factor σ has label a a weakening step is applied, using $\neg I'_B(\sigma)$ instead of $I_A(\sigma)$. Let *Itp* be the interpolant generated without weakening, and Itp' the interpolant generated having applied the weakening step. We know that $I_A(\sigma) \subseteq Itp$ and $\neg I'_B(\sigma) \subseteq Itp'$. From Theorem 3 we know that $I_A(\sigma) \rightarrow \neg I'_B(\sigma)$, therefore we have that $Itp \rightarrow Itp'$.

Following the same reasoning, from Eq. (7) we can see that when a factor σ has label b a strengthening step is applied, using $\neg I'_A(\sigma)$ instead of $I_B(\sigma)$. Let *Itp* be the interpolant generated without strengthening, and Itp' the interpolant generated having applied this strengthening step. We know that $\neg I_B(\sigma) \subseteq Itp$ and $\neg\neg I'_A(\sigma) \subseteq Itp'$. From Theorem 3 we have that $I'_A(\sigma) \rightarrow \neg I_B(\sigma)$. Therefore we have that $Itp' \rightarrow Itp$. ∎

Theorem 5. *Let $Itp_L = (A, B, G[\pi]^C)$. If $\pi \in B$, $L = L_s$ leads to the interpolant that contains the smallest number of equalities, and if $\pi \in A$, $L = L_w$ leads to the interpolant that contains the smallest number of equalities.*

Proof: Consider the labeling function L_s and the computation of $I_A(\pi)$. The usage of this labeling function makes the formula be entirely computed by I_A, never using I_B. Now let $I_A(\delta)$ be some arbitraty subcomputation of $I_A(\pi)$. First, $\bigwedge_{\sigma\in\mathcal{A}(\delta)} J_A(\sigma)$ is computed. From Eq. 3 we know that this formula contains the equality $[\![\sigma_A]\!]$ for every $\sigma_A \in \mathcal{A}(\delta)$ and the equality $[\![\sigma_B]\!]$ for every $\sigma_B \in \mathcal{BA}(\delta)$. Suppose then that a factor γ from $\mathcal{BA}(\delta)$ has label w, which results in the computation of $\neg I'_B(\gamma)$. We know that γ is a B-factor because it came from $\mathcal{BA}(\delta)$, therefore *(i)* $\theta_A = \gamma$; and *(ii)* $\mathcal{B}(\gamma) = \{\gamma\}$. From *(i)* we have that $I'_B(\gamma)$ computes $I_B(\gamma)$, and from *(ii)* we have that $\mathcal{B}(\gamma) = \{\gamma\}$ and $J_B(\gamma)$ is then computed. This reintroduces $[\![\gamma]\!]$ in the interpolant (as the implicated part of $J_B(\gamma)$), since $I_A(\delta)$ already introduced it in the implicant part of some implication in $\bigwedge_{\sigma\in\mathcal{A}(\delta)} J_A(\sigma)$. Notice that if γ had label s this equality would not be reintroduced. The reasoning is symmetrical for the case where L_w is used for the computation of I_B. Therefore we have that $I_A(\pi)$ and $I_B(\pi)$ contain exactly the same equalities, with the difference being the side of the implication (implicant or implicated) that an equality appears in (because of Eq. 3 and Eq. 4). We can also see that any other labeling function L will introduce at least one equality in the interpolant more than once (when it changes from I_A to $\neg I'_B$ or from I_B to $\neg I'_A$).

If $\pi \in B$, an interpolant can be computed by either $I_A(\pi)$ or $\neg I'_B(\pi)$. The interpolant $I_A(\pi)$ has less equalities because it does not introduce the conflict in the interpolant, as $\neg I'_B(\pi)$ does with the term $([\![\mathcal{A}(\pi_1) \cup \mathcal{A}(\pi_2)]\!] \rightarrow \neg[\![\theta_A]\!])$. Therefore, L_s leads to the interpolant with the least number of equalities.

If $\pi \in A$, an interpolant can be computed by either $\neg I_B(\pi)$ or $I'_A(\pi)$. Symmetrically, the interpolant $\neg I_B(\pi)$ has less equalities because it does not introduce the conflict in the interpolant, as $I'_A(\pi)$ does with the term $([\![\mathcal{B}(\pi_1) \cup \mathcal{B}(\pi_2)]\!] \rightarrow \neg[\![\theta_B]\!])$. Therefore, L_w leads to the interpolant with the least number of equalities. ∎