Beautiful Interpolants

Aws Albarghouthi[1] and Kenneth L. McMillan[2]

[1]University of Toronto
[2]Microsoft Research

Abstract. We describe a compositional approach to Craig interpolation based on the heuristic that simpler proofs of special cases are more likely to generalize. The method produces simple interpolants because it is able to summarize a large set of cases using one relatively simple fact. In particular, we present a method for finding such simple facts in the theory of linear rational arithmetic. This makes it possible to use interpolation to discover inductive invariants for numerical programs that are challenging for existing techniques. We show that in some cases, the compositional approach can also be more efficient than traditional lazy SMT as a decision procedure.

1 Introduction

beau·ti·ful *adjective* \'byü-ti-fəl\
1: *...exciting aesthetic pleasure*
2: *generally pleasing* [2]

In mathematics and physics, the beauty of a theory is an important quality. A simple or elegant argument is considered more likely to generalize than a complex one. Imagine, for example, proving a conjecture about an object in N dimensions. We might first try to prove the special case of two or three dimensions, and then generalize the argument to the N-dimensional case. We would prefer a proof of the two-dimensional case that is simple, on the grounds that it will be less prone to depend on particular aspects of this case, thus more likely to generalize.

We can apply this heuristic to the proof of programs or hardware systems. We produce a proof of correctness for some bounded collection of program executions. From this proof we can derive a conjectured invariant of the program using Craig interpolation methods, e.g., [21, 23, 20, 10]. The simpler our conjecture, the less it is able to encode particular aspects of our bounded behaviors, so the more likely it is to be inductive. Typically, our bounded proofs will be produced by an SMT (satisfiability modulo theories) solver. Simplicity of our interpolant-derived conjecture depends on simplicity of the SMT solver's proof. Unfortunately, for reasons we will discuss shortly, SMT solvers may produce proofs that are far more complex than necessary.

In this paper, we consider an approach we call *compositional SMT* that is geared to produce simple interpolants. It is compositional in the sense that the interpolant acts as an intermediate assertion between components of the formula, localizing the reasoning. This approach allows us to solve inductive invariant

generation problems that are difficult for other techniques, and in some cases can solve bounded verification problems much more efficiently than standard lazy SMT [7] methods. The approach is simple to implement and uses an unmodified SMT solver as a "black box".

A lazy SMT solver separates the problems of Boolean and theory reasoning. To test satisfiability of a formula A relative to a theory $\mathcal{T}$, it uses a SAT solver to find propositionally satisfying assignments. These can be thought of as disjuncts in the disjunctive normal form (DNF) of A. A theory solver then determines feasibility of these disjuncts in $\mathcal{T}$. In the negative case, it produces a *theory lemma*. This is a validity of the theory that contradicts the disjunct propositionally. In the worst case, each theory lemma rules out only one amongst an exponential collection of disjuncts.

In compositional SMT, we refute satisfiability of a conjunction $A \wedge B$ by finding an *interpolant* formula I, such that $A \Rightarrow I$, $B \Rightarrow \neg I$ and I uses only the symbols common to A and B. We do this by building two collections of feasible disjuncts of A and B that we call *samples*. We then try to construct a simple interpolant I for the two sample sets. If I is an interpolant for A and B, we are done. Otherwise, we use our SMT solver to find a new sample that contradicts either $A \Rightarrow I$ or $B \Rightarrow \neg I$, and restart the process with the new sample.

Unlike the theory solver in lazy SMT, our interpolant generator can "see" many different cases and try to find a simple fact that generalizes them. This more global view allows compositional SMT to find very simple interpolants in cases when lazy SMT produces an exponential number of theory lemmas.

We develop the technique here in the quantifier-free theory of linear rational arithmetic (QFLRA). This allows us to apply some established techniques based on Farkas' lemma to search for simple interpolants. The contributions of this paper are (1) A compositional approach to SMT based on sampling and interpolation (2) An interpolation algorithm for QFLRA based on finding linear separators for sets of convex polytopes. (3) A prototype implementation that demonstrates the utility of the technique for invariant generation, and shows the potential to speed up SMT solving.

Organization. Sec. 2 illustrates our approach on a simple example. Sec. 3 gives a general algorithm for interpolation via sampling. Sec. 4 describes an interpolation technique for sets of convex polytopes. Sec. 5 presents our implementation and experimental results. Related work is discussed in Sec. 6.

2 Motivating Example

Figure 1 shows two QFLRA formulas A and B over the variables x and y. For clarity of presentation, the two formulas are in DNF, where $\mathsf{A}_1, \mathsf{A}_2$, and A_3 are the disjuncts of A, and B_1 and B_2 are the disjuncts of B. Intuitively, since a disjunct is a conjunction of linear inequalities (*half-spaces*), it represents a *convex polyhedron* in $\mathbb{R}^2$. Fig. 2(a) represents A and B geometrically, where each disjunct is highlighted using a different shade of gray.

$$\begin{aligned} \mathsf{A} =&(x \le 1 \wedge y \le 3) && (\mathsf{A}_1) \\ &\vee (1 \le x \le 2 \wedge y \le 2) && (\mathsf{A}_2) \\ &\vee (2 \le x \le 3 \wedge y \le 1) && (\mathsf{A}_3) \end{aligned} \qquad \begin{aligned} \mathsf{B} =&(x \ge 2 \wedge y \ge 3) && (\mathsf{B}_1) \\ &\vee (x \ge 3 \wedge 2 \le y \le 3) && (\mathsf{B}_2) \end{aligned}$$

Fig. 1. Inconsistent formulas A and B.

Since A and B do not intersect, as shown in Fig 2(a), $\mathsf{A} \wedge \mathsf{B}$ is unsatisfiable. Thus, there exists an interpolant I such that $\mathsf{A} \Rightarrow \mathsf{I}$ and $\mathsf{I} \Rightarrow \neg\mathsf{B}$. One such I is the half-space $x + y \le 4$, shown in Fig. 2(b) as the region encompassing all of A, but not intersecting with B. We now discuss how our technique constructs such an interpolant.

We start by *sampling* disjuncts from A and B. In practice, we sample a disjunct from a formula φ by finding a model $m \models \varphi$, using an SMT solver, and evaluating all linear inequalities occurring in φ with respect to m. Suppose we sample the disjuncts A_2 and B_1. We now find an interpolant of $(\mathsf{A}_2, \mathsf{B}_1)$. To do so, we utilize Farkas' lemma and encode a system of constraints that is satisfiable *iff* there exists a *half-space interpolant* of $(\mathsf{A}_2, \mathsf{B}_1)$. That is, we are looking for an interpolant comprised of a single linear inequality, not an arbitrary Boolean combination of linear inequalities. In this case, we might find the half-space interpolant $y \le 2.5$, shown in Fig. 2(c).

We call $y \le 2.5$ a *partial interpolant*, since it is an interpolant of A_2 and B_1, which are parts of (i.e., subsumed by) A and B, respectively. We now check if this partial interpolant is an interpolant of (A, B) using an SMT solver. First, we check if $\mathsf{A} \wedge y > 2.5$ is satisfiable. Since it is satisfiable, $\mathsf{A} \not\Rightarrow y \le 2.5$, indicating that $y \le 2.5$ is not an interpolant of (A, B). A satisfying assignment of $\mathsf{A} \wedge y > 2.5$ is a model of A that lies outside the region $y \le 2.5$, for example, the point $(1, 3)$ shown in Fig. 2(c). Since $(1, 3)$ is a model of the disjunct A_1, we add A_1 to the set of samples in order to take it into account.

At this point, we have two A samples, A_1 and A_2, and one B sample B_1. We now seek an interpolant for $(\mathsf{A}_1 \vee \mathsf{A}_2, \mathsf{B}_1)$. Of course, we can construct such an interpolant by taking the disjunction of two half-space interpolants: one for $(\mathsf{A}_1, \mathsf{B}_1)$ and one for $(\mathsf{A}_2, \mathsf{B}_1)$. Instead, we attempt to find a *single* half-space that is an interpolant of $(\mathsf{A}_1 \vee \mathsf{A}_2, \mathsf{B}_1)$ – we say that the samples A_1 and A_2 are *merged* into a *sampleset* $\{\mathsf{A}_1, \mathsf{A}_2\}$. As before, we construct a system of constraints and solve it for such an interpolant. In this case, we get the half-space $x+y \le 4$, shown in Fig. 2(d). Since $x + y \le 4$ is an interpolant of (A, B), the algorithm terminates successfully. If there is no half-space interpolant for $(\mathsf{A}_1 \vee \mathsf{A}_2, \mathsf{B}_1)$, we *split* the sampleset $\{\mathsf{A}_1, \mathsf{A}_2\}$ into two samples, and find two half-space interpolants for $(\mathsf{A}_1, \mathsf{B}_1)$ and $(\mathsf{A}_2, \mathsf{B}_1)$.

The key intuition underlying our approach is two-fold: (1) Lazily sampling a small number of disjuncts from A and B often suffices for finding an interpolant for all of A and B. (2) By merging samples, e.g., as A_1 and A_2 above, and encoding a system of constraints to find half-space interpolants, we are forcing

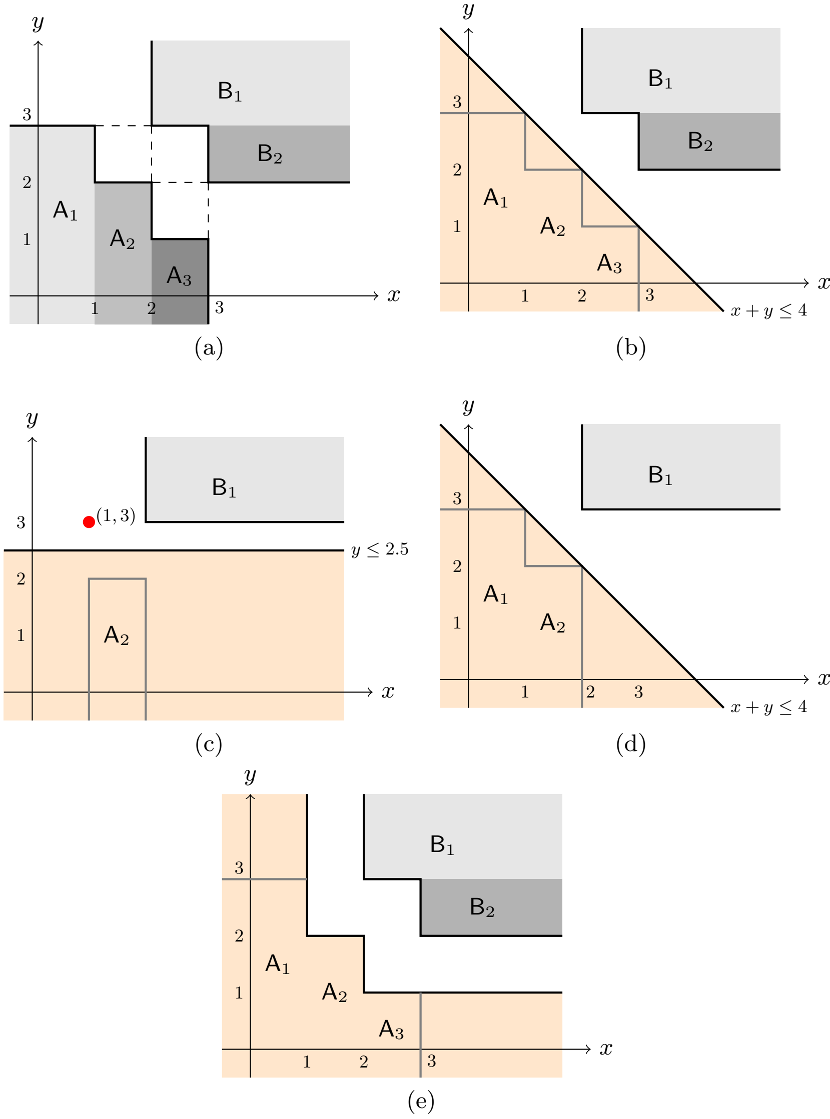

Fig. 2. (a) Illustration of the formulas A and B. (b) An interpolant $x + y \leq 4$ for (A, B). (c) An interpolant $y \leq 2.5$ for $(\mathsf{A}_2, \mathsf{B}_1)$. (d) An interpolant $x + y \leq 4$ for samples $(\mathsf{A}_1 \vee \mathsf{A}_2\, \mathsf{B}_1)$. (e) An interpolant computed by MathSAT5.

the procedure to take a holistic view of the problem and produce simpler and possibly more general interpolants.

Given the formulas A and B, an SMT solver is generally unable to find the simple fact $x + y \leq 4$, due to the specificity of the theory lemmas it produces. For example, we used the MathSAT SMT solver to find an interpolant for A

and B, and it produced the following formula[1]:

$$(x \leq 2 \wedge y \leq 2) \vee ((x \leq 1 \vee y \leq 1) \wedge ((x \leq 2 \wedge y \leq 2) \vee (x \leq 1 \vee y \leq 1))),$$

illustrated in Fig. 2(e). This interpolant is more complex and does not capture the emerging pattern of the samples. As we add more disjuncts to A and B following the pattern, the interpolants produced by MathSAT grow in size, whereas our approach produces the same result. This ability to generalize a series is key to invariant generation. In the SMT approach, the theory solver sees only one pair of disjuncts from A and B at a time, and thus cannot generalize.

3 Constructing Interpolants from Samples

We now present Csmt (Compositional SMT), a generic algorithm for computing an interpolant for a pair of quantifier-free first-order formulas (A, B). Csmt attempts to partition samples from A and B as coarsely as possible into *samplesets*, such that each A sampleset can be separated from each B sampleset by an atomic interpolant formula (for linear arithmetic, this means a single linear constraint). Although Csmt applies to any theory that allows quantifier-free interpolation, for concreteness, we consider here only linear arithmetic.

Preliminaries. A formula of *quantifier-free linear rational arithmetic*, LRA, is a Boolean combination of *atoms*. Each atom is a linear inequality of the form $c_1x_1 + \cdots + c_nx_n \triangleleft k$, where $c_1, \ldots, c_n$ and k are rational constants, $x_1, \ldots, x_n$ are distinct variables, and $\triangleleft$ is either $<$ or $\leq$. The atom is an *open* or *closed half-space* if $\triangleleft$ is $<$ or $\leq$, respectively. We use $\mathsf{LinIq}(\varphi)$ to denote the set of atoms appearing in formula φ, and $\mathsf{Vars}(\varphi)$ to denote the set of variables. We will often write $cx \triangleleft k$ for a half-space $c_1x_1 + \cdots + c_nx_n \triangleleft k$, where c is the row vector of the coefficients $\{c_i\}$, and x is the column vector of the variables $\{x_i\}$.

A *model* of φ is an assignment of rational values to $\mathsf{Vars}(\varphi)$ that makes φ true. Given a model m of φ, we define the *sample* of φ w.r.t. m, written $sample_\varphi(m)$, as the formula

$$\bigwedge\{P \mid P \in \mathsf{LinIq}(\varphi), m \models P\} \wedge \bigwedge\{\neg P \mid P \in \mathsf{LinIq}(\varphi), m \not\models P\}.$$

Note, there are finitely many samples of φ and φ is equivalent to the disjunction of its samples. Geometrically, each sample can be thought of as a convex polytope.

Given two formulas $A, B \in \mathsf{LRA}$ such that $A \wedge B$ is unsatisfiable, an *interpolant* of (A, B) is a formula $I \in \mathsf{LRA}$ such that $\mathsf{Vars}(I) \subseteq \mathsf{Vars}(A) \cap \mathsf{Vars}(B)$, $A \Rightarrow I$, and $B \Rightarrow \neg I$. An interpolant exists for every inconsistent (A, B).

General Algorithm. We formalize Csmt in Fig. 3 as a set of guarded commands. In each, if the condition above the line is satisfied, the assignment below may be executed. The state of Csmt is defined by the following variables:

[1] Produced using MathSAT 5.2.2, and slightly modified for clarity of presentation.

$$\frac{}{\mathsf{S}_A := \emptyset \quad \mathsf{S}_B := \emptyset \quad \mathsf{PItp} := \emptyset}\ \textsc{Init}$$

- Rules for distributing samples into samplesets:

$$\frac{X \in \{A, B\} \quad s_1 \in \mathsf{S}_X \quad s_2 \in \mathsf{S}_X \quad s_1 \neq s_2}{\mathsf{S}_X := (\mathsf{S}_X \setminus \{s_1, s_2\}) \cup \{s_1 \cup s_2\}}\ \textsc{Merge}$$

$$\frac{X \in \{A, B\} \quad s_1 \cup s_2 \in \mathsf{S}_X \quad s_1 \neq \emptyset \quad s_2 \neq \emptyset}{\mathsf{S}_X := (\mathsf{S}_X \setminus \{s_1 \cup s_2\}) \cup \{s_1\} \cup \{s_2\}}\ \textsc{Split}$$

- Rules for constructing and checking interpolants:

$$\frac{s_A \in \mathsf{S}_A \quad s_B \in \mathsf{S}_B \quad \textsc{HalfItp}(s_A, s_B) \neq \diamond \quad \mathsf{PItp}(s_A, s_B) \text{ is undefined}}{\mathsf{PItp}(s_A, s_B) := \textsc{HalfItp}(s_A, s_B)}\ \textsc{PartialItp}$$

$$\frac{C = \textsc{Cand}(\mathsf{S}_A, \mathsf{S}_B, \mathsf{PItp}) \neq \diamond \quad m \models A \wedge \neg C}{\mathsf{S}_A := \mathsf{S}_A \cup \{\{\mathit{sample}_A(m)\}\}}\ \textsc{CheckItpA}$$

$$\frac{C = \textsc{Cand}(\mathsf{S}_A, \mathsf{S}_B, \mathsf{PItp}) \neq \diamond \quad m \models C \wedge B}{\mathsf{S}_B := \mathsf{S}_B \cup \{\{\mathit{sample}_B(m)\}\}}\ \textsc{CheckItpB}$$

- Termination conditions:

$$\frac{s_A \in \mathsf{S}_A \quad s_B \in \mathsf{S}_B \quad |s_A| = |s_B| = 1 \quad \textsc{HalfItp}(s_A, s_B) = \diamond}{A \wedge B \text{ is satisfiable}}\ \textsc{Sat}$$

$$\frac{C = \textsc{Cand}(\mathsf{S}_A, \mathsf{S}_B, \mathsf{PItp}) \quad A \Rightarrow C \quad C \Rightarrow \neg B}{C \text{ is an interpolant of } (A, B)}\ \textsc{Unsat}$$

Fig. 3. CSMT as guarded commands.

– *Sampleset collections* $\mathsf{S}_A, \mathsf{S}_B$ are sets of samplesets of A and B, respectively. Initially, as dictated by the command INIT, $\mathsf{S}_A = \mathsf{S}_B = \emptyset$.
– *Partial interpolant map* PItp is a map from pairs of samplesets to half-spaces. Invariantly, if (s_A, s_B) is in the domain of PItp then $\mathsf{PItp}(s_A, s_B)$ is an interpolant for $(\bigvee s_A, \bigvee s_B)$.

We do not attempt to find a smallest set of half-spaces that separate the samples, as this problem is NP-complete. This can be shown by reduction from *k-polyhedral separability*: given two sets of points on a plane, is there a set of less than k half-spaces separating the two sets of points [25]. Instead, we heuristically cluster the samples into large samplesets such that each pair of samplesets (s_A,s_B) is linearly separable. Even with a minimal clustering, this solution may be sub-optimal, in the sense of using more half-spaces than necessary. Since our objective is a heuristic one, we will seek reasonably simple interpolants with moderate effort, rather than trying to optimize.

In practice, we heuristically search the space of clusterings using MERGE and SPLIT. MERGE is used to combine two samplesets in $\mathsf{S}_{\{A,B\}}$ and make them a single sampleset. SPLIT performs the opposite of MERGE: it picks a sampleset in $\mathsf{S}_{\{A,B\}}$ and splits it into two samplesets. The command PARTIALITP populates the map PItp with interpolants for pairs of samplesets (s_A, s_B). This is done by calling HALFITP (s_A, s_B), which returns a half-space interpolant of $(\bigvee s_A, \bigvee s_B)$ if one exists, and the symbol $\diamond$ otherwise (Sec. 4 presents an implementation of HALFITP).

The commands CHECKITPA and CHECKITPB check if the current *candidate interpolant* is *not* an interpolant of (A, B), in which case, samples produced from counterexamples are added to S_A and S_B. The function CAND constructs a candidate interpolant from PItp. If the domain of PItp contains $\mathsf{S}_A \times \mathsf{S}_B$, then

$$\text{CAND}(\mathsf{S}_A, \mathsf{S}_B, \mathsf{PItp}) \equiv \bigvee_{s_A \in \mathsf{S}_A} \left(\bigwedge_{s_B \in \mathsf{S}_B} \mathsf{PItp}(s_A, s_B) \right)$$

Otherwise the result is $\diamond$. The result of CAND is an interpolant of (A', B'), where A' and B' are the disjunction of all samples in S_A and S_B, respectively. This DNF formula may not be optimal in size. Synthesizing optimal candidate interpolants from partial interpolants is a problem that we hope to explore in the future.

If SAT or UNSAT apply, then the algorithm terminates. SAT checks if two singleton samplesets do not have a half-space separating them. Since both samples define convex polytopes, if no half-space separates them, then they intersect, and therefore $A \wedge B$ is satisfiable. UNSAT checks if a candidate interpolant C is indeed an interpolant, in which case CSMT terminates successfully.

Example 1. Consider the formulas A and B from Sec. 2. Suppose that CSMT is in the state $\mathsf{S}_A = \{\{\mathsf{A}_2\}\}$, $\mathsf{S}_B = \{\{\mathsf{B}_1\}\}$, $\mathsf{PItp} = \emptyset$, where A_2 and B_1 are the samples of A and B defined in Sec. 2. By applying PARTIALITP, we find a half-space separating the only two samplesets. As a result, $\mathsf{PItp}(\{\mathsf{A}_2\}, \{\mathsf{B}_1\}) = y \leq 2.5$. Suppose we now apply CHECKITPA. The candidate interpolant $\text{CAND}(\mathsf{S}_A, \mathsf{S}_B, \mathsf{PItp})$ at this point is $y \leq 2.5$. Since $\mathsf{A} \wedge y > 2.5$ is satisfiable, CHECKITPA adds the sampleset $\{\mathsf{A}_1\}$, which is not subsumed by the candidate interpolant, to S_A. Now, $\mathsf{S}_A = \{\{\mathsf{A}_1\}, \{\mathsf{A}_2\}\}$. Since we have two samplesets in S_A, we apply MERGE and get $\mathsf{S}_A = \{\{\mathsf{A}_1, \mathsf{A}_2\}\}$. PARTIALITP is now used to find a half-space interpolant for the samplesets $\{\mathsf{A}_1, \mathsf{A}_2\}$ and $\{\mathsf{B}_1\}$. Suppose it finds the plane $x + y \leq 4$. Then UNSAT is applicable and the algorithm terminates with $x + y \leq 4$ as an interpolant of (A, B). □

The key rule for producing simpler interpolants is MERGE, since it decreases the number of samplesets, and forces the algorithm to find a smaller number of half-spaces that separate a larger number of samples. In Example 1 above, if we do not apply MERGE, we might end up adding all the samples of A and B to S_A and S_B. Thus, producing an interpolant with a large number of half-spaces like the one illustrated in Fig. 2(e).

Theorem 1 (Soundness of CSMT). *Given two formulas A and B, if* CSMT *terminates using*

1. *Sat, then* $A \wedge B$ *is satisfiable.*
2. *Unsat, then* $\text{Cand}(\mathsf{S}_A, \mathsf{S}_B, \mathsf{PItp})$ *is an interpolant of* (A, B).

Proof (Sketch). In case 1, by definition of the rule Sat, we know that there is a sample a of A and a sample b of B such that there does not exist a half-space I that is an interpolant of (a, b). Since both a and b define convex polytopes, if a and b do not have a half-space separating them, then $a \wedge b$ is satisfiable, and therefore $A \wedge B$ is satisfiable.

In case 2, the candidate interpolant C, checked in rule Unsat, is over the shared variables of A and B (by definition of HalfItp), $A \Rightarrow C$, and $C \Rightarrow \neg B$. Therefore, it is an interpolant of (A, B). □

We now consider the termination (completeness) of Csmt. It is easy to see that one can keep applying the commands Merge and Split without ever terminating. To make sure that does not happen, we impose the restriction that for any sampleset in $P_{\{A,B\}}$, if it is ever split, it never reappears in the sampleset collection. For example, if a sampleset $s_A \in \mathsf{S}_A$ is split into two samplesets s_A^1 and s_A^2 using Split, then s_A cannot reappear in S_A. Given this restriction, Csmt always terminates.

Theorem 2 (Completeness of Csmt). *For any two formulas A and B,* Csmt *terminates.*

Proof (Sketch). Since there are finitely many samples, CheckItpA, CheckItpA and PartialItp must be executed finitely. Due to our restriction, Merge is also bounded. Thus, if we do not terminate, eventually Split reduces all samplesets to singletons, at which point Sat or Unsat must terminate the procedure.

4 Half-space Interpolants

In this section, we present a constraint-based implementation of the parameter HalfItp of Csmt. Given two samplesets s_A and s_B, our goal is to find a half-space interpolant $ix \triangleleft k$ of $(\bigvee s_A, \bigvee s_B)$. Since both s_A and s_B represent a union (set) of convex polytopes, we could compute the *convex hulls* of s_A and s_B and use techniques such as [28, 6] to find a half-space separating the two convex hulls. To avoid the potentially expensive convex hull construction, we set up a system of linear constraints whose solution encodes both the separating half-space and the proof that it is separating. We then solve the constraints using an LP solver. This is an extension of the method in [28] from pairs of convex polytopes to pairs of *sets* of convex polytopes.

The intuition behind this construction is simple. We can represent the desired separator as a linear constraint I of the form $ix \leq k$, where x is a vector of variables, i is a vector of unknown coefficients, and k is an unknown constant. We wish to solve for the unknowns i and k. To express the fact that I is a separator, we apply Farkas' lemma. This tells us that a set of linear constraints

$\{C_j\}$ implies I exactly when I can be expressed as a linear combination of $\{C_j\}$ with non-negative coefficients. That is, when $\Sigma_j c_j C_j + d \equiv I$ for some non-negative $\{c_j\}$ and d. The key insight is that this equivalence is itself a set of linear equality constraints with unknowns $\{c_j\}$, d, i and k. The values of $\{c_j\}$ and d constitute a certificate that in fact $\{C_j\}$ implies I. We can therefore construct constraints requiring that each sample in s_A implies (is contained in) I, and similarly that each sample in s_B implies $\neg I$ (equivalent to $-ix < -k$). Solving these constraints we obtain a separator I and simultaneously a certificate that I is a separator. What is new here is only that we solve for multiple Farkas proofs: one for each sample in s_A or s_B.

We now make this construction precise. Let $N_A = |s_A|$ and $N_B = |s_B|$. Each sample in s_A is represented as a vector inequality $A_j x \leq a_j$, for $j \in [1, N_A]$. Similarly, samples in s_B are of the form $B_j x \leq b_j$, for $j \in [1, N_B]$. Here, x is a column vector of the variables $\mathsf{Vars}(A) \cup \mathsf{Vars}(B)$. For example, the sample $y \leq 1 \wedge z \leq 3$ is represented as follows:

$$\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} y \\ z \end{pmatrix} \leq \begin{pmatrix} 1 \\ 3 \end{pmatrix}$$

In the remainder of this section, we assume that all samples are conjunctions of closed half-spaces, i.e., non-strict inequalities. (In the Appendix, we present a simple extension to our construction for handling open half-spaces.) It follows that if there exists a half-space interpolant for $(\bigvee s_A, \bigvee s_B)$, then there exists a *closed* half-space that is also an interpolant. Thus, our goal is to find a closed half-space $ix \leq k$ that satisfies the following two conditions:

$$\forall j \in [1, N_A] \cdot A_j x \leq a_j \Rightarrow ix \leq k \tag{1}$$

$$\forall j \in [1, N_B] \cdot B_j x \leq b_j \Rightarrow \neg ix \leq k \tag{2}$$

Condition (1) specifies that $ix \leq k$ subsumes all s_A samples. Condition (2) specifies that $ix \leq k$ does not intersect with any of the s_B samples. By forcing coefficients of unshared variables to be 0 in $ix \leq k$, we ensure that $ix \leq k$ is an interpolant of $(\bigvee s_A, \bigvee s_B)$. To construct such half-space interpolant, we utilize Farkas' lemma [29]:

Theorem 3 (Farkas' lemma). *Given a satisfiable system of linear inequalities* $Ax \leq b$ *and a linear inequality* $ix \leq k$*, then:* $Ax \leq b \Rightarrow ix \leq k$ *iff there exists a row vector* $\lambda \geq 0$ *s.t.* $\lambda A = i$ *and* $\lambda b \leq k$.

Using this fact, we construct a constraint Φ_A that, when satisfiable, implies that a half-space $ix \leq k$ satisfies condition (1). Consider a sample $A_j x \leq a_j$, where A_j is an $m_j \times n_j$ matrix. We associate with this sample a row vector λ_{A_j} of size m_j, consisting of fresh variables, called *Farkas coefficients* of the m_j linear inequalities represented by $A_j x \leq a_j$. We now define Φ_A as follows:

$$\Phi_A \equiv \forall j \in [1, N_A] \cdot \ \lambda_{A_j} \geq 0 \ \wedge \ \lambda_{A_j} A_j = i \ \wedge \ \lambda_{A_j} a_j \leq k$$

The row vector i is of the form $(i_{x_1} \ \cdots \ i_{x_n})$, where each i_{x_j} is a variable denoting the coefficient of variable x_j in the interpolant. Similarly, k is a variable

denoting the constant in the interpolant. Suppose we have an assignment to i, k, and λ_{A_j} that satisfies Φ_A. Then, by Farkas' lemma, the half-space $ix \leq k$ satisfies condition (1), where i and k are replaced by their assignment values. This because a satisfying assignment of Φ_A implies that for every $j \in [1, N_A]$, $A_j x \leq a_j \Rightarrow ix \leq k$.

We now encode a constraint Φ_B that enforces condition (2). As before, we associate a row vector λ_{B_j} with each sample $B_j x \leq b_j$. We define Φ_B as follows:

$$\Phi_B \equiv \forall j \in [1, N_B] \cdot \ \lambda_{B_j} \geq 0 \ \wedge \ \lambda_{B_j} B_j = -i \ \wedge \ \lambda_{B_j} b_j < -k$$

Following Farkas' lemma, a satisfying assignment of Φ_B results in a half-space $-ix < -k$ that subsumes all samples in s_B. That is, $B_j x \leq b_j \Rightarrow \neg ix \leq k$, for all $j \in [1, N_B]$, thus satisfying condition (2). Therefore, a satisfying assignment of $\Phi_A \wedge \Phi_B$ produces a half-space interpolant $ix \leq k$ for $(\bigvee s_A, \bigvee s_B)$. Note that our encoding implicitly ensures that coefficients of unshared variables are 0 in $ix \leq k$. See Appendix for an example of solving these constraints to obtain an interpolant. Thm. 4 below states soundness and completeness of our encoding for samples that are conjunctions of closed half-spaces.

Theorem 4 (Soundness and Completeness of HalfItp). *Given two samplesets (where all samples are systems of closed half-spaces), s_A and s_B, and their corresponding encoding $\Phi \equiv \Phi_A \wedge \Phi_B$, then:*

1. *If Φ is satisfiable using a variable assignment m, then $i^m x \leq k^m$ is an interpolant for $(\bigvee s_A, \bigvee s_B)$, where i^m and k^m are the values of i and k in m, respectively.*
2. *Otherwise, no half-space interpolant exists for $(\bigvee s_A, \bigvee s_B)$.*

5 Implementation and Evaluation

We implemented the compositional SMT technique, Csmt, in the C♯ language, using the Z3 SMT solver [26] for constraint solving and sampling. The primary heuristic choice in the implementation is how to split and merge samplesets. The heuristics we use for this are described in the Appendix, along with some optimizations that improve performance.

To experiment with Csmt, we integrated it with Duality [24], a tool that uses interpolants to construct inductive invariants. We will call this CsmtDua. In the configuration we used, Duality can be thought of as implementing *Lazy Abstraction With Interpolants* (LAWI), as in Impact [21]. The primary difference is that we use a large-block encoding [8], so that edges in the abstract reachability tree correspond to complete loop bodies rather than basic blocks.

Sequence interpolants with Csmt. Duality produces sequences of formulas corresponding to (large block) program execution paths in static single assignment (SSA) form, and requires interpolants to be computed for these sequences. An interpolant sequence for formulas $A_1, \ldots, A_n$ is a sequence of formulas $I_1, \ldots, I_{n-1}$ where I_i is an interpolant for $(\bigwedge_{k \leq i} A_k, \bigwedge_{k > i} A_k)$. The interpolant sequence must also be be *inductive*, in the sense that $I_{i-1} \wedge A_i \Rightarrow I_i$. In

Program	CsmtDua	CPA	Ufo	InvGenAI	InvGenCS
f2	✔	✗	✗	$✗_f$	$✗_f$
gulv	✔	✗	✗	$✗_f$	$✗_f$
gulv_simp	✔	✔	✔	✔	✔
substring1	✔	✗	✔	✔	✔
pldi08	✔	✔	✔	$✗_f$	$✗_f$
pldi082unb	✔	✗	✗	✔	✔
xy0	✔	✗	✗	✔	✔
xy10	✔	✔	✔	✔	✔
xy4	✔	✗	✗	✔	✔
xyz	✔	✗	✗	✔	✔
xyz2	✔	✗	✗	✔	✔
dillig/01	✔	✔	✔	✔	✔
dillig/03	✔	✗	✗	✔	✔
dillig/05	✔	✗	✔	✔	✔
dillig/07	✔	✔	✔	✔	✔
dillig/09	✔	✗	✗	✔	✗
dillig/12	✔	✗	✔	✗	✗
dillig/15	✔	✔	✔	✔	✔
dillig/17	✔	✔	✗	✔	✔
dillig/19	✔	✔	✔	$✗_f$	$✗_f$
dillig/20	✔	✔	✔	$✗_f$	$✗_f$
dillig/24	✔	✔	✔	✔	✔
dillig/25	✔	✔	✔	✔	$✗_f$
dillig/28	✔	✗	✗	✔	✔
dillig/31	✔	✔	✗	✔	✗
dillig/32	✔	✔	✔	✔	✔
dillig/33	✔	✔	$✗_f$	✗	✗
dillig/35	✔	✔	✔	$✗_f$	$✗_f$
dillig/37	✔	✔	✔	$✗_f$	$✗_f$
dillig/39	✔	✔	✔	✔	✔
#Solved	**30**	17	17	21	18

Fig. 4. Verification results of Csmt, CPAChecker, Ufo, and two configurations of InvGen on a collection of C benchmarks.

this application, an inductive interpolant sequence can be thought of as a Hoare logic proof of the given execution path.

A key heuristic for invariant generation is to try to find a *common* interpolant for positions that correspond to the same program location. The requirement to find a simple common interpolant forces us to "fit" the emerging pattern of consecutive loop iterations, much as occurs in Figure 2(b). We can easily reduce the problem of finding a common interpolant to finding a single interpolant for a pair (A, B), provided we know the correspondence between variables in successive positions in the sequence.

Say we have substitutions σ_i mapping each program variables to its instance at position i in the SSA sequence. We construct formula A as $\bigvee_{i<n}(\bigwedge_{k\leq i} A_k)(\sigma_i^{-1})$ and B as $\bigvee_{i<n}(\bigwedge_{k>i} A_k)(\sigma_i^{-1})$. That is, A represents all the prefixes of the execution path and B all the suffixes. If I is an interpolant for (A, B), then the sequence $\{I_i\}$ where $I_i = I\sigma_i$ is an interpolant sequence for $A_1, \ldots, A_n$. If it is *inductive*, we are done, otherwise we abandon the goal of a common interpolant.

	iZ3		Csmt	
#Iters	Time(s)	Size	Time(s)	Size
2	0.083	13	0.556	2
4	0.076	15	1.284	2
6	0.087	35	2.111	2
8	0.133	59	3.439	2
10	0.195	124	5.480	2
12	0.473	447	7.304	2
14	0.882	762	19.750	11
16	0.710	158	15.040	4
18	0.753	147	18.730	4
20	0.847	57	44.136	24
22	0.867	45	54.710	17
24	0.857	47	138.197	45
26	0.895	23	56.643	6
28	0.888	11	31.417	2
30	0.882	1	31.318	1

Fig. 5. Size of interpolants from Csmt and iZ3 for BMC unrollings of increasing length.

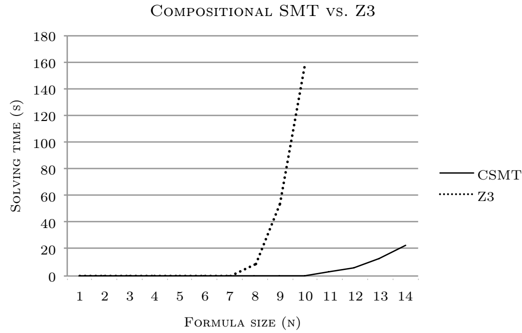

Fig. 6. Solving time (s) of Csmt and Z3 vs. formula size n.

5.1 Evaluation

We now present an evaluation of Csmt. First, we compare CsmtDua against a variety of verification tools on a set of benchmarks requiring non-trivial numerical invariants. Second, we demonstrate how our prototype implementation of Csmt can outperform Z3 at SMT solving. Finally, we compare the size of the interpolants computed by Csmt to those computed by iZ3 [22], an interpolating version of Z3.

Csmt for Verification. We compared CsmtDua against the state-of-the-art verifiers CPAChecker (CPA) [9] and Ufo [5] – the top abstraction-based verifiers from the 2013 software verification competition (SV-COMP) [1]. CPA is a lazy abstraction tool that admits multiple configurations – we used its predicate abstraction and interpolation-based refinement configuration `predicateAnalysis`. Ufo is an interpolation-based verification tool that guides interpolants with abstract domains [4, 3] – we did not use any abstract domains for guidance (using an abstract domain does not change the number of solved benchmarks). We also compared against InvGen [16], an invariant generation tool that combines non-linear constraint solving, numerical abstract domains, and dynamic analysis to compute safe invariants. We use InvGenAI to denote the default configuration of InvGen, and InvGenCS to denote a configuration where abstract interpretation invariants are not used to strengthen invariants computed by constraint solving (`-nostrength` option).

To study the effectiveness of CsmtDua, we chose small C programs from the verification literature that require non-trivial numeric invariants. The benchmarks `f2`, `gulv*`, and `substring1` are from [14]. The benchmarks `pldi08*` are from [15]. The benchmarks `xy*` are variations on a classic two-loop example requiring linear congruences. The benchmarks `dillig/*` were provided Dillig, *et al.* [12]. Some are from the literature and some are original and are

intended to test arithmetic invariant generation. We omitted benchmarks using the mod operator, as we do not support this, and also elided some duplicates and near-duplicates. The benchmarks and tool output are available at `http://www.cs.utoronto.ca/~aws/cav13.zip`.

Fig. 4 shows the result of applying Csmt and the aforementioned tools on 30 such benchmarks. In the table, ✔ means the tool verified the program, ✗ means that the tool timed out (after 60s), and $✗_f$ means the tool terminated unsuccessfully. The run times in successful cases tend to be trivial (less than 2s), so we do not report them. We observe that CsmtDua produced proofs for all benchmarks, whereas CPA and Ufo failed to prove 13, and InvGenAI failed to prove 9.

This shows the effectiveness of the heuristic that simple proofs tend to generalize. For example, in the case of `pldi082unb`, CsmtDua produces the intricate invariant $x+y-2N \leq 2 \wedge y \leq x$. On the other hand, interpolant-based refinement in CPA and Ufo diverges, producing an unbounded sequence of predicates only containing x and N.

Compositional SMT Solving. One way prove unsatisfiability of a formula $\Phi \equiv A \wedge B$ is to exhibit an interpolant I of (A, B). Using Csmt, this might be more efficient than direct SMT solving because (1) Csmt only makes SMT queries on the components A and B, and (2) by merging samples, Csmt can find proofs not available to the SMT solver's theory solver. This is especially true if Φ has many disjuncts.

Suppose, for example, we have

$$A \equiv x_0 = y_0 = 0 \wedge \bigwedge_{i=1}^{n} (\mathrm{inc}_i \vee \mathrm{eq}_i)$$

$$B \equiv \bigwedge_{i=n+1}^{2n} (\mathrm{dec}_i \vee \mathrm{eq}_i) \wedge x_{2n} = 0 \wedge y_{2n} \neq 0,$$

where inc_i is $x_i = x_{i-1} + 1 \wedge y_i = y_{i-1} + 1$, dec_i is $x_i = x_{i-1} - 1 \wedge y_i = y_{i-1} - 1$ and eq_i is $x_i = x_{i-1} \wedge y_i = y_{i-1}$. The formula $\Phi_n = A \wedge B$ is essentially the BMC formula for our benchmark `xy0`, where the two loops are unrolled n times. The pair (A, B) has a very simple interpolant, that is, $x_n = y_n$, in spite of the fact that each of A and B have 2^n disjuncts. Moreover, the conjunction $A \wedge B$ has 2^{2n} disjuncts. In a lazy SMT approach, each of these yields a separate theory lemma. Fig. 6 shows the time (in seconds) taken by Z3 and Csmt to prove unsatisfiability of Φ_n for $n \in [1, 14]$. For $n = 10$, Z3 takes 160 seconds to prove unsatisfiability of Φ_n, whereas Csmt requires 1.4 seconds. For $n > 10$, Z3 terminates without producing an answer. This shows that a compositional approach can be substantially more efficient in cases where a large number of cases can be summarized by a simple interpolant.

Interpolant Size. We now examine the relative complexity of the interpolants produced by Csmt and SMT-based interpolation methods, represented by iZ3 [22]. For our formulas, we used BMC unrollings of `s3_clnt_1`, an SSH benchmark from the software verification competition (SV-COMP) [1]. Fig. 5 shows the sizes of

the interpolants computed by iZ3 and CSMT (and the time taken to compute them) for unrollings (#ITER) of N iterations of the loop, with the interpolant taken after $N/2$ iterations. The size of the interpolant is measured as the number of operators and variables appearing in it. Since the loop has a reachability depth of 14, all the interpolants from $N = 30$ are *false*. For $N = 12$, the interpolant size for iZ3 is 447, whereas for CSMT it is 2, a 200X reduction. A large reduction in interpolant size is observed for most unrolling lengths. Notice that, since this example has few execution paths, SMT is quite fast. This illustrates the opposite case from the previous example, in which the compositional approach is at a significant performance disadvantage.

6 Related Work

We compare CSMT against interpolation and invariant generation techniques.

Constraint-based Techniques. In [28], Rybalchenko and Stokkermans describe a method of for computing half-spaces separating two convex polytopes using Farkas' lemma. Here, we generalize this method to separators for sets of polytopes. This helps us search for a simple interpolant separating all the samples, rather than constructing one separating plane for each sample pair. This in turn gives us an interpolation procedure for arbitrary formulas in QFLRA, rather than just conjunctions of literals.

Interpolants from Classifiers. Our work is similar in flavor to, and inspired by, an interpolant generation approach of Sharma et al. [30]. This approach uses point samples of A and B (numerical satisfying assignments) rather than propositional disjuncts. A machine learning technique – *Support Vector Machines* (SVM's) – is used to create linear separators between these sets. The motivation for using disjuncts (polytopes) rather than points is that they give a broader view of the space and allow us to exploit the logical structure of the problem. In particular, it avoids the difficult problem of clustering random point samples in a meaningful way. In practice, we found that bad clusterings led to complex interpolants that did not generalize. Moreover, we found the SVM's to be highly sensitive to the sample set, in practice often causing the interpolation procedure to diverge, *e.g.*, by finding samples that approach the boundary of a polytope asymptotically.

Interpolants from Refutation Proofs. A number of papers have studied extracting "better" interpolants from refutation proofs. For example, [13, 31, 27] focused on the process of extracting interpolants of varying strengths from refutation proofs. Hoder et al. [18] proposed an algorithm that produces syntactically smaller interpolants by applying transformations to refutation proofs. Jhala and McMillan [19] described a modified theory solver that yields interpolants in a bounded language. In contrast, we have taken the approach of structuring the proof search process expressly to yield simple interpolants. Our method can compute simpler and more general interpolants, by discovering facts that are not found in refutation proofs produced by lazy SMT solvers. On the other hand, constructing interpolants from refutation proofs can be much faster

in cases where the number of theory lemmas required is small. Also, while the compositional approach may be applicable to the various theories handled by proof-based interpolation, we have as yet only developed a method for LRA.

Template methods. A more direct approach to synthesize linear inductive invariants is based on Farkas' Lemma and non-linear constraint solving [11]. The invariant is expressed as a fixed conjunction of linear constraints with unknown coefficients, and one solves simultaneously for the invariant and the Farkas proof of its correctness. This has the advantage, relative to the interpolant approach, that it does not require unfolding the program and the disadvantage that it requires a non-linear arithmetic solver. Currently, such solvers do not scale to more than a few variables. Thus, the difficulty of finding a solution grows rapidly with the number of constraints in the invariant [11]. An example of a tool using this approach is InvGen, run without abstract interpretation (called InvGenCS in Table 4). An examination of the 12 cases in which InvGenCS fails shows that in most the invariant we found has at least three conjuncts, and in some it is disjunctive, a case that the authors of InvGen have found impractical using the method [16]. Thus, it appears that by searching for simple interpolants, we can synthesize invariants with greater propositional complexity than can be obtained using the constraint-based approach.

7 Conclusion

We have developed a compositional approach to interpolation based on the heuristic that simpler proofs of special cases are more likely to generalize. The method produces simple (perhaps even beautiful) interpolants because it is able to summarize a large set of cases using one relatively simple fact. In particular, we presented a method for finding such simple facts in the theory of linear rational arithmetic. This made it possible to use interpolation to discover inductive invariants for numerical programs that are challenging for existing techniques. We also observed that for formulas with many disjuncts, the compositional approach can be more efficient than non-compositional SMT.

Our work leaves many avenues open for future research. For example, can the method be effectively applied to integer arithmetic, or the theory of arrays? From a scalability standpoint, we would like to improve the performance of Csmt on formulas requiring more complex interpolants. One possible direction is parallelism, e.g., instead of decomposing a formula into $A \wedge B$, we could decompose it into multiple sets of conjuncts and use techniques such as [17] to parallelize SMT solving.

References

1. Competition On Software Verification, `http://sv-comp.sosy-lab.org/`
2. Merriam-Webster Dictionary (Dec 2012), `www.merriam-webster.com`
3. Albarghouthi, A., Gurfinkel, A., Chechik, M.: Craig Interpretation. In: SAS'12. pp. 300–316 (2012)
4. Albarghouthi, A., Gurfinkel, A., Chechik, M.: From Under-approximations to Over-approximations and Back. In: TACAS'12. pp. 157–172 (2012)

5. Albarghouthi, A., Li, Y., Gurfinkel, A., Chechik, M.: Ufo: A framework for abstraction- and interpolation-based software verification. In: CAV'12 (2012)
6. Alur, R., Dang, T., Ivancic, F.: Counter-example guided predicate abstraction of hybrid systems. In: TACAS. pp. 208–223 (2003)
7. Barrett, C.W., Sebastiani, R., Seshia, S.A., Tinelli, C.: Satisfiability modulo theories. In: Handbook of Satisfiability (2009)
8. Beyer, D., Cimatti, A., Griggio, A., Keremoglu, M.E., Sebastiani, R.: "Software Model Checking via Large-Block Encoding". In: FMCAD'09. pp. 25–32 (2009)
9. Beyer, D., Keremoglu, M.E.: "CPAchecker: A Tool for Configurable Software Verification". In: CAV'11 (2011)
10. Cimatti, A., Griggio, A., Sebastiani, R.: "Efficient Generation of Craig Interpolants in Satisfiability Modulo Theories". ACM Trans. Comput. Log. 12(1), 7 (2010)
11. Colón, M., Sankaranarayanan, S., Sipma, H.: Linear invariant generation using non-linear constraint solving. In: CAV. pp. 420–432 (2003)
12. Dillig, I., Dillig, T., Li, B.: personal communication (2012)
13. D'Silva, V., Kroening, D., Purandare, M., Weissenbacher, G.: Interpolant strength. In: VMCAI. pp. 129–145 (2010)
14. Gulavani, B.S., Chakraborty, S., Nori, A.V., Rajamani, S.K.: Automatically Refining Abstract Interpretations. In: TACAS'08. vol. 4963, pp. 443–458
15. Gulwani, S., Srivastava, S., Venkatesan, R.: Program analysis as constraint solving. In: PLDI'08. pp. 281–292
16. Gupta, A., Rybalchenko, A.: Invgen: An efficient invariant generator. In: CAV'09. pp. 634–640
17. Hamadi, Y., Marques-Silva, J., Wintersteiger, C.M.: Lazy decomposition for distributed decision procedures. In: PDMC'11. pp. 43–54
18. Hoder, K., Kovács, L., Voronkov, A.: Playing in the grey area of proofs. In: POPL'12. pp. 259–272
19. Jhala, R., McMillan, K.L.: A practical and complete approach to predicate refinement. In: TACAS'06. pp. 459–473
20. Kroening, D., Weissenbacher, G.: Lifting propositional interpolants to the word-level. In: FMCAD'07. pp. 85–89 (2007)
21. McMillan, K.L.: Interpolation and SAT-Based Model Checking. In: CAV'03. pp. 1–13 (2003)
22. McMillan, K.L.: Interpolants from Z3 proofs. In: FMCAD'11. pp. 19–27
23. McMillan, K.L.: An interpolating theorem prover. In: TACAS. pp. 16–30 (2004)
24. McMillan, K.L., Rybalchenko, A.: Computing relational fixed points using interpolation. Tech. Rep. MSR-TR-2013-6, Microsoft Research (2013)
25. Megiddo, N.: On the complexity of polyhedral separability. Discrete & Computational Geometry 3, 325–337 (1988)
26. de Moura, L., Bjørner, N.: "Z3: An Efficient SMT Solver". In: TACAS'08. LNCS, vol. 4963, pp. 337–340 (2008)
27. Rollini, S.F., Sery, O., Sharygina, N.: Leveraging interpolant strength in model checking. In: CAV'12. pp. 193–209
28. Rybalchenko, A., Sofronie-Stokkermans, V.: Constraint solving for interpolation. J. Symb. Comput. 45(11), 1212–1233 (2010)
29. Schrijver, A.: Theory of linear and integer programming. John Wiley & Sons, Inc., New York, NY, USA (1986)
30. Sharma, R., Nori, A.V., Aiken, A.: Interpolants as classifiers. In: CAV'12. pp. 71–87
31. Weissenbacher, G.: Interpolant strength revisited. In: SAT'12. pp. 312–326

A Handling Open Half-spaces

In Sec. 4, we assumed that all samples are conjunctions of closed half-spaces. We now extend the above encoding to handle open half-spaces.

For every vector λ_{A_j}, we use $\lambda^{<}_{A_j}$ to denote the sub-vector of λ_{A_j} that represents Farkas coefficients of open half-spaces. The same holds for B samples. For example, consider the formula $y \leq 1 \wedge z < 0$, represented as $Ax \triangleleft a$:

$$\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} y \\ z \end{pmatrix} \begin{matrix} \leq \\ < \end{matrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

Let $\lambda_A = (\lambda^1_A \ \lambda^2_A)$ be the vector of Farkas coefficients associated with $Ax \triangleleft a$. Then, the sub-vector $\lambda^{<}_A = (\lambda^2_A)$, since λ^2_A is Farkas coefficient of the open half-space $z < 0$.

We now rewrite Φ_B as follows (the only difference is that $\lambda_{B_j} b_j < -k$ is relaxed to $\lambda_{B_j} b_j \leq -k$):

$$\Phi_B \equiv \forall j \in [1, N_B] \cdot \ \lambda_{B_j} \geq 0 \ \wedge \ \lambda_{B_j} B_j = -i \ \wedge \ \lambda_{B_j} b_j \leq -k$$

Intuitively, if $\lambda_{B_j} b_j = -k$, then the linear combination of sample j in s_B coincides with the half-space $ix \leq k$. Similarly, if $\lambda_{A_j} a_j = k$, then the linear combination of sample j in s_A coincides with the half-space $ix \leq k$. We now use this fact to detect when an interpolant $ix \triangleleft k$ should be open or closed. The following two constraints use the Boolean variable open to indicate whether $\triangleleft$ is $<$ or $\leq$.

$$\Phi^{<}_A \equiv \forall j \in [1, N_A] \cdot \lambda_{A_j} a_j = k \wedge \lambda^{<}_{A_j} = 0 \Rightarrow \neg \mathsf{open}$$
$$\Phi^{<}_B \equiv \forall j \in [1, N_B] \cdot \lambda_{B_j} b_j = -k \wedge \lambda^{<}_{B_j} = 0 \Rightarrow \mathsf{open}$$

Finally, an assignment of $i, k, \lambda_{A_j}, \lambda_{B_j}$, and open that satisfies $\Phi_A \wedge \Phi_B \wedge \Phi^{<}_A \wedge \Phi^{<}_B$ produces an interpolant $ix \triangleleft k$, where $\triangleleft$ is $<$ if open is assigned to *true*, and $\leq$ otherwise. The following theorem states soundness and completeness of our encoding.

Theorem 5 (Soundness and Completeness of HalfItp). *Given two samplesets, s_A and s_B, and their corresponding encoding $\Phi \equiv \Phi_A \wedge \Phi_B \wedge \Phi^{<}_A \wedge \Phi^{<}_B$, then:*

1. *If Φ is satisfiable using a variable assignment m, then $i^m x \triangleleft k^m$ is an interpolant for $(\bigvee s_A, \bigvee s_B)$, where i^m and k^m are the values of i and k in m, respectively, and $\triangleleft$ is $<$ if open^m is true, and $\leq$ otherwise.*
2. *Otherwise, no half-space interpolant exists for $(\bigvee s_A, \bigvee s_B)$.*

B Heuristics and Optimizations for CSMT

Our implementation of CSMT is a determinization of the commands in Fig. 3. The primary heuristic decision we must make in CSMT is how and when to

apply Split and Merge. First, when a new sample s is added by CheckItpA or CheckItpB, we Merge it into one of the existing samplesets. We choose the sampleset that contains a sample most syntactically similar to s. When we fail to find a linear separator for samplesets s_A and s_B, we must split one of the two. We do this by removing one sample. In this choice, we are aided by the unsatisfiable core of the Farkas constraints produced by Z3. We only consider removing a sample if its Farkas constraints appear in the core, which indicates that in some way this sample is responsible for the non-existence of a separator. Once the sample is removed to its own sampleset, we consider merging it into an existing set using the above-described heuristic.

B.1 Multiple separators

We note that it is also possible to use an SMT solver to solve for multiple separators at once and to assign each pair of samples to one of the separators. This technique is effective when the number of required separating planes is small. It can be combined with the merging/splitting search approach. However, it is not needed to solve the benchmark problems we tried.

B.2 Sample simplification

Each sample is a conjunction of literals that may contain many variables not in the shared vocaburaly. Any of these variables may be existentially quantified without affecting the interpolant. In practice we have found it is useful to use quantifier elimination to remove some variables. This results in fewer inequality constraints in the sample and thus fewer Farkas coefficients to solve for. Presently, we look for implied inequalities of the form $v = e$, where v is a non-shared variable and e is an expression not containing v. We then eliminate the variable v by substitution. In principle, it is also possible to eliminate some variables by Fourier-Motzkin steps, though we have not attempted this.

B.3 Additional constraints on the Farkas proofs

By adding constraints, we can control the Farkas proof we obtain, and thus the interpolant. One heuristic we have found useful is to first try to solve for a separator that uses only unit coefficients and only a bounded set of variables. If the bound is 2, we obtain octagon constraints. If there is no such separator, we can relax the constraints. Another useful approach is to try to set as many Farkas coefficients to zero as possible. This means that fewer inequalities are actually used in the Farkas proofs. Heuristically, simpler Farkas proofs tend to result in better interpolants (though there is only one benchmark in our set that requires this heuristic to terminate with an inductive invariant).

C Example of solving Farkas constraints

The following example illustrates solving the Farkas constraints, from Section 4, to obtain an interpolant.

Example 2. Recall the example from Sec. 2. We would like to find a half-space interpolant for the two samplesets $s_{\mathsf{A}} = \{\mathsf{A}_1, \mathsf{A}_2\}$ and $s_{\mathsf{B}} = \{\mathsf{B}_1\}$. The following represents the formulas $\mathsf{A}_1, \mathsf{A}_2$, and B_1 as $A_1\boldsymbol{x} \leq a_1$, $A_2\boldsymbol{x} \leq a_2$, and $B_1\boldsymbol{x} \leq b_1$, respectively, where $\boldsymbol{x}$ is the column vector of variables:

$$\overbrace{\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}}^{A_1} \overbrace{\begin{pmatrix} x \\ y \end{pmatrix}}^{\boldsymbol{x}} \leq \overbrace{\begin{pmatrix} 1 \\ 3 \end{pmatrix}}^{a_1} \qquad \overbrace{\begin{pmatrix} 1 & 0 \\ -1 & 0 \\ 0 & 1 \end{pmatrix}}^{A_2} \begin{pmatrix} x \\ y \end{pmatrix} \leq \overbrace{\begin{pmatrix} 2 \\ -1 \\ 2 \end{pmatrix}}^{a_2}$$

$$\overbrace{\begin{pmatrix} -1 & 0 \\ 0 & -1 \end{pmatrix}}^{B_1} \begin{pmatrix} x \\ y \end{pmatrix} \leq \overbrace{\begin{pmatrix} -2 \\ -3 \end{pmatrix}}^{b_1}$$

We now show how a satisfying assignment for $\Phi_{\mathsf{A}} \wedge \Phi_{\mathsf{B}}$ represents a half-space interpolant. First, let us set $i = (1\ 1)$ and $k = 4$. Now, the only variables that require assignments are Farkas coefficients λ_{A_1}, λ_{A_2}, and λ_{B_1}.

- Let $\lambda_{A_1} = (1\ 1)$. As a result, $\lambda_{A_1} A_1 = (1\ 1) = i$ and $\lambda_{A_1} a_1 = 4 = k$. By Farkas' lemma, this indicates that $i\boldsymbol{x} \leq k$ subsumes A_1.
- Let $\lambda_{A_2} = (1\ 0\ 1)$. As a result, $\lambda_{A_2} A_2 = (1\ 1) = i$ and $\lambda_{A_2} a_2 = 4 = k$. Similar to A_1, this means that $i\boldsymbol{x} \leq k$ subsumes A_2. At this point, our variable assignment satisfies Φ_{A}.
- Finally, to satisfy Φ_{B}, let $\lambda_{B_1} = (1\ 1)$. As a result, $\lambda_{B_1} B_1 = (-1\ -1) = -i$ and $\lambda_{B_1} b_1 = -5 < -k$. This means that $-x - y \leq -5$ subsumes B_1, and contradicts $x + y \leq 4$.

The above assignment satisfies $\Phi_{\mathsf{A}} \wedge \Phi_{\mathsf{B}}$, and indicates that $x + y \leq 4$ is a half-space interpolant for $(\mathsf{A}_1 \vee \mathsf{A}_2, \mathsf{B}_1)$. □