LRA Interpolants from No Man's Land

Leonardo Alt, Antti E. J. Hyvärinen and Natasha Sharygina

Università della Svizzera italiana, Lugano, Switzerland
leonardoaltt@gmail.com, antti.hyvaerinen@usi.ch, natasha.sharygina@usi.ch

Abstract. Interpolation is becoming a standard technique for over-approximating state spaces in software model checking with Satisfiability Modulo Theories (SMT). In particular when modelling programs with linear arithmetics, the standard state-of-the-art technique might provide either interpolants that are too specific or too generic to be useful for a given application. In this work we introduce the SI-LRA interpolation system for linear real arithmetics that allows the tuning of interpolants based on shifting between the primal and dual interpolants. We prove a strength relation between the interpolants constructed by SI-LRA, and integrate SI-LRA into a propositional interpolator in an SMT solver. Our evaluation, performed using a state-of-the-art software model checker, reveals that correct tuning with SI-LRA can reduce the number of needed refinements by up to one third and provide lower runtimes.

1 Introduction

Many modern software verification techniques rely on satisfiability modulo theories (SMT) solvers which accept as input instances of the propositional satisfiability problem (SAT) where some of the Boolean variables are interpreted as equalities in some first-order theory. Satisfiability in linear real arithmetics (LRA) consists of determining whether a set of linear equalities defined over real variables has a solution. Most LRA solvers use highly efficient variants of the Simplex algorithm [9], making LRA ubiquitous in model checking and as an intermediary solving step for non-linear real arithmetics (NRA), linear integer arithmetics (LIA), and other techniques.

One of the core tasks in software verification is to find invariants that prove the absence of bad behavior and are inductive with respect to a program loop. In many powerful verification approaches the invariants are synthesized as generalizations of simple examples produced by SMT solvers on short loop-free fragments of program executions. In this work we consider the problem of proving that a set of linear equalities over real variables does not have a solution, and how these inequalities can be relaxed while still preserving the property of not having solutions. In particular in model checking this problem is known as *interpolation* over LRA. For clarity we present the *strength-controlled interpolation system for LRA* (SI-LRA), a novel interpolation algorithm based on computing a primal and a dual interpolant and constructing the final interpolant from the area between these two (hence the *No Man's Land* in the title). We present the

system as an extension of [17] which allows the use of a *strength factor* control the strength. Our experimental results on the interpolating model checker HiFrog [2] suggest that mid-range strengths result in high-quality interpolants as indicated by fewer required refinements, and surprisingly that this does not correlate with the algorithm runtime.

Given the central role of interpolation in program verification, it is natural to search for LRA interpolants that meet the needs of different model checkers. Such needs include efficiency in generating interpolants, simplicity of the formulas representing the interpolants, and how they integrate with SMT solvers. Nevertheless the flexibility of LRA interpolation approaches is still an issue, and many existing techniques do not allow the interpolation-based application to interfere with the interpolant generation. To the best of our knowledge, this paper presents the first flexible interpolation system for LRA capable of generating an infinite amount of interpolants for a given interpolation problem.

Related work. The Propositional Labeled Interpolation System (LIS-PROP), introduced in [7] and further developed in [21,20,3,24], provides a framework for adjusting Craig interpolants with respect to their strength and, to some extent, size for a particular application. LIS-PROP is extended to interpolation of equalities over uninterpreted functions in [4], while this work presents a way of controlling the strength of interpolants in linear real arithmetics.

The pioneering work in LRA interpolation was presented in [19] and then used in [17], where inference rules are given and extended to accommodate the interpolation algorithm of [19]. A more practical approach is taken in [9], where LRA interpolation is done inside an SMT solver by extracting the proof of unsatisfiability from the Simplex algorithm. Both [9] and [17] are only capable of computing a single interpolant, while our SI-LRA is able to generate an infinite amount of interpolants.

In [1] the authors construct convex interpolants for $A \wedge B$, where A and B are disjunctions of constraints, leading potentially to general and simple interpolants. A similar smoothing approach is followed in [5]. SI-LRA is orthogonal to both these approaches, since we aim at creating interpolants of different strength. The techniques can be used together, with the algorithms from [1,5] using SI-LRA to generate interpolants for specific subsets of A and B.

Interpolants can be generated by first solving a constraint problem in linear arithmetics, and then applying a linear programming solver as a black box [23]. However, since most SMT solvers already use the Simplex algorithm for LRA solving, the use of a black-box solver introduces an extra overhead. The algorithm from [23] is also not able to generate multiple interpolants for a given problem, and does not provide strength control as SI-LRA does. A similar approach is presented in [25] for computing shared interpolants for a maximal number of conflicts by minimizing the number of linear equations in the interpolant with respect to computing an interpolant from a single theory conflict. The approach suffers from the same overhead as [23], and does not provide control over the strength of the generated interpolants.

An approach related to the interpolation system discussed in this work is presented for Nonlinear Real Arithmetics in [10]. This work is also able to provide interpolants of different strength, controlled by a labeling function. The interpolation system from [10] uses the proof of unsatisfiability generated by a decision procedure based on Interval Constraint Propagation. In a high level, their system is similar to [19].

Finally, [22] constructs interpolants based on program semantics. Applying semantics to the real equations would provide a powerful heuristic for LRA interpolation.

This work is organized as follows. We present our notation in Sec. 2 and the SI-LRA interpolation framework in Sec. 3. Section 4 presents an experimental evaluation using a model checker and conclusions are drawn in Sec. 5.

2 Preliminaries

This section introduces the quantifier-free theory of Linear Real Arithmetic (LRA) and interpolation. We refer the reader to [16] and [9] for an in-depth discussion on decision procedures for linear real arithmetics as used by modern SMT solvers.

An LRA instance is a conjunction of *linear constraints* of the form $c \bowtie a_1x_1 + \ldots + a_nx_n$ where $\bowtie \in \{<, \leq\}$, constants a_i are rational numbers, and the *problem variables* x_i assume rational values from $\mathbb{Q}$. The *General Simplex* is an algorithm that takes as input an LRA instance and determines whether there are values for the problem variables such that the set of constraints is satisfiable. In case there are values for the problem variables that satisfy the constraints the algorithm provides one such concrete valuation. If the set of constraints is unsatisfiable the algorithm returns an unsatisfiable subset Γ of the constraints as an explanation for the unsatisfiability. In standard SMT solving the explanations are used for guiding the SAT solver underlying the SMT solver to avoid similar conflicts. In practice this is done by adding the *explanation clause* $C_{\text{LRA}} = \neg(\bigwedge_{c \in \Gamma})$ to the SAT solver. In the context of interpolation they have a further use for constructing safe overapproximations of search spaces.

Given an unsatisfiable first-order formula $A \wedge B$, a *Craig interpolant* [6] for A is a first-order formula I_A such that $A \rightarrow I_A$, $I_A \wedge B$ is unsatisfiable, and I_A is defined over the non-logical symbols shared between A and B. Given an interpolation algorithm $Itp(A, B)$, the *dual interpolant* for A is the negation of the interpolant computed by $Itp(B, A)$.

2.1 LRA Interpolation

An explanation Γ is a minimal unsatisfiable subset of the constraints of a query. The explanation can be used to create a proof of unsatisfiability which in turn can be used to compute an interpolant. A straightforward but inflexible approach to create LRA interpolants is presented in [17]. The idea is to create a tree that represents the proof of unsatisfiability and annotate the nodes with partial

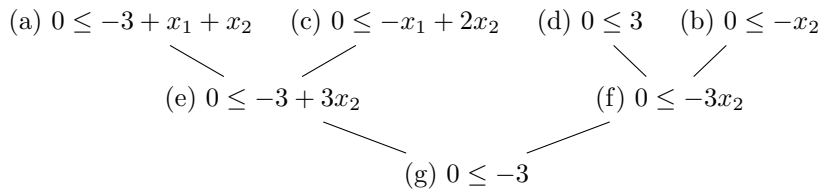
Table 1. LRA proof system for BI-LRA [17].

Hyp $\frac{}{\phi}\ \phi \in \Gamma$	Comb $\frac{0 \leq t \quad 0 \leq u}{0 \leq t + u}$
Taut $\frac{}{0 \leq c}\ c \in \mathbb{Q}, c \geq 0$	Mult $\frac{0 \leq c \quad 0 \leq t}{0 \leq ct}\ c \in \mathbb{Q}, c > 0$

interpolants. We present this system here and will extend it using the duality of interpolants to create the SI-LRA in Sec. 3. To clarify the difference, we call the system from [17] the *basic interpolation system for LRA* (BI-LRA).

A *term* in BI-LRA is a linear combination $c_0 + c_1x_1 + \ldots + c_nx_n$, where $x_1 \ldots x_n$ are distinct variables and $c_0 \ldots c_n$ are constants. This proof system accepts constraints of the form $0 \leq term$. The rules for deriving a proof in BI-LRA are given in Table 1, where t and u are terms, and Γ is the explanation for unsatisfiability.

Example 1. Consider the unsatisfiable formula $(3 \leq x_1 + x_2) \wedge (0 \leq -x_2) \wedge (0 \leq 2x_2 - x_1)$. Once the constraints have been transformed into the form accepted by the proof system, we can compute the following proof using the rules in Table 1:



Applying the *Hyp* rule to the constraints (a)-(c) is straightforward. We then apply rule *Comb* to (a) and (c) to infer (e). We apply rule *Taut* to obtain the tautology (d) $0 \leq 3$ and then *Mult* to multiply (b) by (d) and infer (f) $0 \leq -3x_2$. Now we can apply *Comb* to combine (e) and (f) to finally infer (g) $0 \leq -3$, a contradiction.

An LRA interpolation problem is a 3-tuple $P = (A, B, R)$, where A and B are two sets of LRA constraints such that they are unsatisfiable when conjoined, and R is the proof of unsatisfiability for $A \wedge B$. The intuition behind BI-LRA is to use the contribution from the constraints from A to the sum that leads to the contradiction. The contribution from A is then effectively the interpolant, since (i) it is implied by A; (ii) summing the contribution with B leads to a contradiction; and (iii) summing all the contributions from A removes the symbols local to A, therefore leaving only the common symbols from A and B to the interpolant.

The interpolation rules of BI-LRA are given in Table 2, where x, y, x' and y' are terms, and $[\phi]$ is the annotated term such that $0 \leq \phi$ is the partial interpolant for that node.

Table 2. Interpolation system from [17].

Hyp-A	Hyp-B
$\dfrac{}{0 \le x\ [x]}\ (0 \le x) \in A$	$\dfrac{}{0 \le x\ [0]}\ (0 \le x) \in B$
Comb	**Mult**
$\dfrac{0 \le x\ [x'] \quad 0 \le y\ [y']}{0 \le x + y\ [x' + y']}$	$\dfrac{0 \le c \quad 0 \le x\ [x']}{0 \le cx\ [cx']}$

Example 2. Continuing Example 1, suppose $A = \{0 \le -3 + x_1 + x_2, 0 \le -x_1 + 2x_2\}$ and $B = \{0 \le -x_2\}$. The interpolant for A is computed by BI-LRA as follows. Using the *Hyp-A* and *Hyp-B* rules we can infer the partial interpolants

$$\frac{}{(a)\ 0 \le -3 + x_1 + x_2\ [-3 + x_1 + x_2]}\ (0 \le -3 + x_1 + x_2) \in A,$$

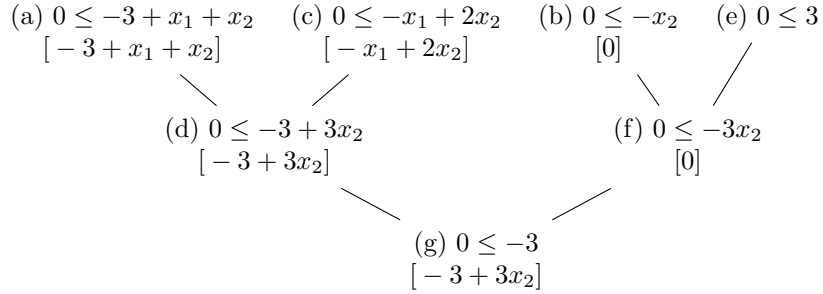
$$\frac{}{(c)\ 0 \le -x_1 + 2x_2\ [-x_1 + 2x_2]}\ (0 \le -x_1 + 2x_2) \in A, \text{ and}$$

$$\frac{}{(b)\ 0 \le -x_2\ [0]}\ (0 \le -x_2) \in B.$$

Following the proof from Example 1, we apply the *Comb* rule on (c) and (a) to infer (d)

$$\frac{0 \le -3 + x_1 + x_2 \quad 0 \le -x_1 + 2x_2}{(d)\ 0 \le -3 + 3x_2\ [-3 + 3x_2]}.$$

We now apply the *Taut* rule to use the tautology (e) $0 \le 3$ and then *Mult* rule on (e) and (b) to infer (e) $0 \le -3x_2\ [0]$. By applying *Comb* on (d) and (f) we infer (g) $0 \le -3\ [-3 + 3x_2]$. The annotated proof of unsatisfiability is



and the final interpolant for A is $0 \le -3 + 3x_2$.

2.2 Propositional Interpolation

In our experiments in Sec. 4 we combine SI-LRA with the propositional interpolation interpolation system (LIS-PROP) from [8,3]. We briefly introduce the

system here; for lack of space we assume familiarity with propositional resolution (see, e.g., [3]).

A resolution refutation is a directed, acyclic tree where the leaves are *source clauses*, the inner nodes are *resolvent clauses*, and the root is the empty clause. LIS-PROP takes as input two propositional formulas A, B in conjunctive normal form, a resolution refutation R of $A \wedge B$, and a *labeling function* $L(p, C) \mapsto \{a, b, ab\}$, where (p, C) is an occurrence of a variable p in a clause C, of the refutation R, and a, b, ab are *labels*. The system returns an interpolant I for A such that the shape of I depends on L. For all variable occurrences (p, C) in R, $L(p, C) = a$ if p is local to A; and $L(p, C) = b$ if p is local to B. However, the label can be freely chosen for the shared variables, allowing in practice a significant amount of flexibility in constructing interpolants. The label of a variable occurrence in a resolvent C is determined by the labels of the variables in its antecedents: If a variable occurs in both its antecedents with different labels, the label of the new occurrence is ab; and in all other cases the label is equivalent to the label in its antecedent or both antecedents.

An interpolation algorithm based on LIS-PROP [3] computes an interpolant with a dynamic algorithm by annotating each clause of R with a *partial interpolant* starting from the source clauses. The partial interpolant of a source clause C is

$$I(C) = \begin{cases} \bigvee\{l \mid l \in C \text{ and } L(var(l), C) = b\} & \text{if } C \in A, \text{ and} \\ \bigwedge\{\neg l \mid l \in C \text{ and } L(var(l), C) = a\} & \text{if } C \in B, \end{cases} \tag{1}$$

The partial interpolant of a resolvent clause C with pivot p and antecedents C^+ and C^-, where $p \in C^+$ and $\neg p \in C^-$, is

$$I(C) = \begin{cases} I(C^+) \vee I(C^-) & \text{if } L(p, C^+) = L(p, C^-) = a, \\ I(C^+) \wedge I(C^-) & \text{if } L(p, C^+) = L(p, C^-) = b, \text{ and} \\ (I(C^+) \vee p) \wedge (I(C^-) \vee \neg p) & \text{otherwise.} \end{cases} \tag{2}$$

The final interpolant can be obtained as $I(\bot)$ where $\bot$ is the empty clause, i.e., the root of the refutation.

LIS-PROP allows ordering interpolants partially with respect to their logical strength based on the labeling function. In Sec. 4 we use six propositional interpolation algorithms based on LIS-PROP: The algorithms M_s and M_w that are known as the strong and the weak McMillan interpolants [18]; the Huang [12] Krajícek [15] Pudlák [19] interpolation algorithm (P), and the proof-sensitive interpolation algorithms PS, PS_s, and PS_w from [3]. Given an ordering $\leq$ for logical strengths of formulas, where the least element is the strongest, one can show that $\mathrm{M}_s \leq I \leq \mathrm{M}_w$ for any LIS-PROP-based interpolation algorithm I, and that $\mathrm{PS}_s \leq \mathrm{PS} \leq \mathrm{PS}_w$.

The interpolation algorithms from LIS-PROP can be combined with the algorithm BI-LRA, and later with SI-LRA, by first computing the interpolants I_{LRA} for the explanation clauses C_{LRA} of LRA, and then using these interpolants as the partial interpolants $I(C_{\mathrm{LRA}})$.

Table 3. The dual interpolation system

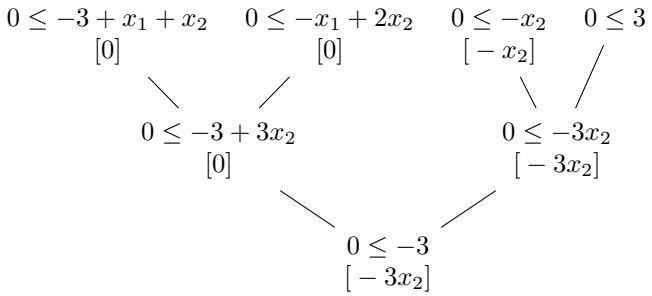
Hyp-A $\dfrac{}{0 \le x\ [0]}\ (0 \le x) \in A$	Hyp-B $\dfrac{}{0 \le x\ [x]}\ (0 \le x) \in B$
Comb $\dfrac{0 \le x\ [x'] \quad 0 \le y\ [y']}{0 \le x + y\ [x' + y']}$	Mult $\dfrac{0 \le c \quad 0 \le x\ [x']}{0 \le cx\ [cx']}$

3 The LRA Interpolation System SI-LRA

The intuition of the interpolation system SI-LRA is to apply the duality of interpolants to the interpolation algorithm BI-LRA to obtain two interpolants and construct new interpolants by a shift transformation that lie between the interpolant and its dual. Let Itp_M be the interpolation algorithm of BI-LRA. We define its dual Itp_D to be $Itp_D(A, B, R) = \neg Itp_M(B, A, R)$. To formalize the presentation we first prove an ordering between Itp_M and Itp_D in Lemma 2 and then define what we mean by interpolant shifting in Theorem 1.

We start the presentation with an example illustrating the dual interpolant.

Example 3. Recall Example 2, where we had $A = \{0 \le -3 + x_1 + x_2, 0 \le -x_1 + 2x_2\}$ and $B = \{0 \le -x_2\}$. The interpolant Itp_D is given by the annotated proof



where the dual interpolant is $\neg(0 \le -3x_2)$.

Formally, the dual interpolation algorithm works as shown in Table 3, where x, y, x' and y' are terms, and $[\phi]$ is the annotated term such that $0 \le \phi$ is the partial interpolant for that node.

Let $[\gamma]$ be the annotation in the root of the proof tree and the inequality $0 \le \gamma$ an interpolant for B. By duality of interpolants, we can assert that $\neg(0 \le \gamma)$ is an interpolant for A. Furthermore, we can state the following relationship between the primal and dual interpolants Itp_M and Itp_D:

Lemma 1. *Let $P = (A, B, R)$ be an interpolation problem. Let the inequality $c_1 \le x$ be the interpolant generated by Itp_M, where c_1 is a constant and x is an LRA term. Then Itp_D generates an interpolant of the form $\neg(c_2 \le -x)$, where c_2 is a constant.*

Proof. Let R be a proof tree that proves the unsatisfiability of $A \wedge B$, annotated with partial interpolants. By Table 1 we know that R is constructed by summing the inequalities from A and B, and by multiplying by a constant. The proof can be seen as a way to parenthesize these operations. Using associativity of sum, we can rearrange any arbitrary proof such that the contradiction is inferred via an application of the *Comb* rule on two inequalities

$$0 \leq -c_1 + x \tag{3}$$

$$0 \leq -c_2 - x \tag{4}$$

such that (i) $0 \leq -c_1 + x$ is inferred only using inequalities from A; and (ii) $0 \leq -c_2 - x$ is inferred only using inequalities from B, where x is an LRA term and $-c_1 - c_2 < 0$. From (i), the partial interpolants for Eq. (3) and Eq. (4), when computing Itp_M are, respectively, $[-c_1 + x]$ and 0; from (ii) follows that the partial interpolants for Eq. (3) and Eq. (4), when computing Itp_D are, respectively, 0 and $[-c_2 - x]$. Therefore, we can see that the term annotated with the contradiction node is $[-c_1 + x]$ when computing Itp_M and $[-c_2 - x]$ when computing Itp_D. Because of that, we know that the interpolant Itp_M is $c_1 \leq x$ and the interpolant Itp_D is $\neg(c_2 \leq -x)$. □

Lemma 2. *Let $c_1 \leq x$ and $\neg(c_2 \leq -x)$ be the interpolants generated for a fixed interpolation problem by Itp_M and Itp_D, respectively. The interpolants generated by Itp_M and Itp_D represent lower bounds for x, where $-c_2 < c_1$.*

Proof. In $Itp_M(P)$, c_1 is clearly a lower bound for x. Transforming $Itp_D(P)$ shows that $\neg(c_2 \leq -x) \equiv \neg(-c_2 \geq x) \equiv -c_2 < x$. By Eq. (3) and Eq. (4), $-c_1 - c_2 < 0$, and therefore it follows that $-c_2 < c_1$. □

3.1 The Strength Factor

By Lemma 2 we have established the strength relation between Itp_M and Itp_D, and are ready to introduce SI-LRA. Our idea is based on the observation that Itp_M and Itp_D represent lower bounds for the same term (Lemma 2), which means that any constant c such that $-c_2 < c \leq c_1$ can substitute c_1 in $Itp_M = c_1 \leq x$, to create a new interpolant $Itp_c = c \leq x$.

Lemma 3. *Let $c_1 \leq x$ and $\neg(c_2 \leq -x)$ be the interpolants generated for the same interpolation problem P by Itp_M and Itp_D, respectively. Let c be a constant such that $-c_2 < c \leq c_1$. Then $Itp_c = c \leq x$ is an interpolant for P.*

Proof. Because $c \leq c_1$, $Itp_M(P) \rightarrow Itp_c(P)$. Because $-c_2 < c$, $Itp_c(P) \rightarrow Itp_D(P)$. Therefore Itp_c is an interpolant for P. □

Algorithm 1 SI-LRA

1: **procedure** ITP($P = (A, B, R, \alpha)$)
2: **Requires** $0 \leq \alpha \leq 1$.
3: **if** $\alpha = 1$ **then**
4: **return** $Itp_D(P)$
5: **if** $\alpha = 0$ **then**
6: **return** $Itp_M(P)$
7: $c_1 \leq x \leftarrow Itp_M(P)$
8: $\neg(c_2 \leq -x) \leftarrow Itp_D(P)$
9: **return** $c_1 - \alpha(c_1 + c_2) \leq x$

Since the bounds c_1 and $-c_2$ change from problem to problem, it is easier to normalize this interval and apply a *strength factor*. Let P be an interpolation problem such that $Itp_M(P) = c_1 \leq x$ and $Itp_D(P) = -c_2 \leq -x$. Given a factor α such that $0 \leq \alpha \leq 1$, we can create a new interpolant $Itp_\alpha \equiv c_\alpha \leq x$, where $c_\alpha = c_1 - \alpha(c_1 + c_2)$.

We extend the LRA interpolation problem to include the strength factor, $P = (A, B, R, \alpha)$ and for clarity present SI-LRA in the form of an algorithm in Alg. 1. In case $-c_2 < c_1$ it is possible to generate infinitely many interpolants of different strength for a given interpolation problem, giving a very fine-grained control over the strength of the generated interpolants.

Theorem 1. *Let α and α' be two strength factors such that $0 \leq \alpha < \alpha' \leq 1$. Then $Itp(A, B, R, \alpha) \rightarrow Itp(A, B, R, \alpha')$.*

Proof. Analogous to the proof of Lemma 3. □

Theorem 1 shows that the strength of the LRA interpolants can be controlled by the strength factor. The interpolation algorithm Itp_M from [17] is represented by the strength factor 0, and generates the strongest interpolants among SI-LRA. The dual interpolation algorithm Itp_D generates the weakest interpolants.

The main advantage of SI-LRA can be visualized when it is combined with propositional interpolation in an SMT solver.

Example 4. Let $A = x \leq 1 \wedge y \leq 1$ and $B = (y \geq 4 \wedge x \geq 0 \wedge x \leq 3) \vee (x \geq 3 \wedge y \geq 0)$ be two Boolean formulas such that the atoms are LRA terms. Notice that we cannot decide satisfiability of $A \wedge B$ using Simplex only. To achieve this task we can, for instance, use an SMT solver. The formula $A \wedge B$ is unsatisfiable, proven by two unsatisfiable queries to the LRA solver, where each query consists of a conjunction of LRA constraints and is solved using the Simplex algorithm. The LRA interpolants that are generated by these queries are then used in propositional interpolation. Using a fixed propositional interpolation algorithm we get the following interpolants for A when changing the LRA interpolation algorithm:

$$\begin{aligned} Itp_M: &\quad I_M = x \leq 1 \wedge y \leq 1 \\ Itp_D: &\quad I_D = \neg x \geq 3 \wedge \neg y \geq 4 \\ Itp_{0.5}: &\quad I_{0.5} = x \leq 2 \wedge y \leq 2.5 \end{aligned}$$

Notice that $I_M \rightarrow I_{0.5} \rightarrow I_D$. Fig. 1 shows the graphical representation of the problem. The blue region is A and the red region is B. We can see graphically that they are unsatisfiable when conjoined. The interpolant I_M happens to be

the same as A. Interpolants $I_{0.5}$ and I_D are represented, respectively, by the light and dark green areas of the graph.

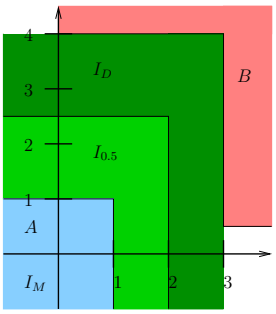


Fig. 1. LRA problem and different interpolants.

Since the strength of the LRA interpolants is given in the level of the theory interpolants, if a propositional interpolation algorithm from the Labeled Interpolation Systems is used, the strength of the final interpolant is maintained.

Theorem 2. *Let $P = (A, B, R)$ be an interpolation problem. Given two LRA interpolation algorithms $Itp_\alpha, Itp_{\alpha'}$ from SI-LRA such that $0 \leq \alpha < \alpha' \leq 1$, and a fixed propositional interpolation algorithm Itp_{prop} from LIS-PROP, the interpolants I, I' computed by the combination algorithm Itp_{prop}, Itp_α and $Itp_{prop}, Itp_{\alpha'}$, respectively, are ordered such that $I \leq I'$.*

Proof. Eq. (1) and Eq. (2) show how an interpolation algorithm from LIS-PROP creates interpolants from the leaves to the root. Eq. (1) is only applied to Boolean clauses and not to theory clauses, so can be disregarded in this context. Let n be a non-leaf node of R that has as children two theory leaves, t_1 and t_2. Let $x_1 = Itp_\alpha(t_1)$, $x_2 = Itp_\alpha(t_2)$, $y_1 = Itp_{\alpha'}(t_1)$ and $y_2 = Itp_{\alpha'}(t_2)$. By Theorem 1, $x_1 \to y_1$ and $x_2 \to y_2$.

We now analyze the three possibilities to build the partial interpolant for n in Eq. (2):

- The first is a disjunction of the partial interpolants of t_1 and t_2. We know that $((x_1 \to y_1) \wedge (x_2 \to y_2)) \to ((x_1 \vee x_2) \to (y_1 \vee y_2))$, so the strength is maintained.
- The second is a conjunction of the partial interpolants of t_1 and t_2. We know that $((x_1 \to y_1) \wedge (x_2 \to y_2)) \to ((x_1 \wedge x_2) \to (y_1 \wedge y_2))$, so the strength is maintained.
- The third is a conjunction of two disjunctions, formed by the partial interpolants of t_1 and t_2 and the pivot of the resolution rule which is an arbitrary variable. It is also true that $((x_1 \to y_1) \wedge (x_2 \to y_2)) \to (((x_1 \vee p) \wedge (x_2 \vee \neg p)) \to ((y_1 \vee p) \wedge (y_2 \vee \neg p)))$, so the strength is maintained.

The case where n has as children a theory leaf and a Boolean leaf clearly holds. Since the annotation of the propositional part of R is not affected, the Theorem holds. □

Note that the proof of Theorem 2 allows choosing different strength factors for different theory leaves in the refutation proof, giving the application even more possibilities to generate suitable interpolants.

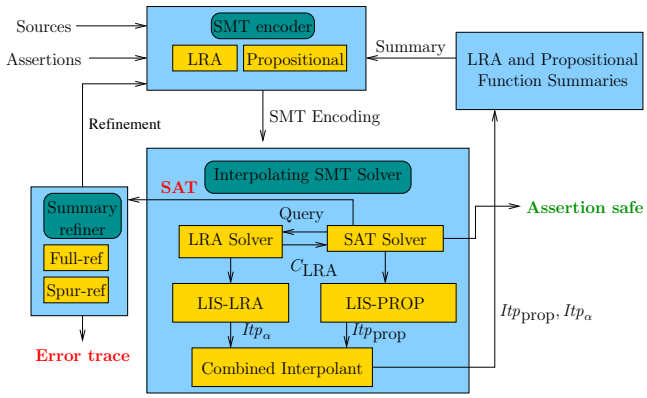


Fig. 2. Components of HiFrog relevant to our experiments

4 Experimental Evaluation

We implemented and integrated SI-LRA into the existing propositional interpolation in OpenSMT2 [14]. Often software verification can be done in a level which ignores arithmetic overflows. In this case LRA can be used in software model checking to abstract the heavy-weight bit-precise propositional encoding to gain speed-up due to the higher-level theory. If a model checker reports that a certain property is true when using LRA, it is also true for the propositional model. However, if a property is determined unsafe in LRA, the generated counterexample might be spurious and introduced by the LRA abstraction. There are several *refinement strategies* a model-checker can recover from such cases (see, e.g., [13]). For simplicity we study two strategies in the experiments: one where a bit-precise solver needs to be invoked for every counterexample (*Full-ref*), and another where the bit-precise solver is invoked only for the spurious counterexamples (*Spur-ref*).

HiFrog applies interpolation-based techniques to create function summaries that are stored and reused while checking incrementally different assertions in a program. Our set of benchmarks consists of C code originating both from the industry and from SV-COMP (`https://sv-comp.sosy-lab.org/`) where this approach is relevant.

Figure 2 describes the main components of the HiFrog model-checking environment relevant to our experiments. The Source files and assertions are given as input to the system which either (i) reports that an assertion is safe and produces a summary for a function in the form of an interpolant, or (ii) outputs an error trace witnessing an execution that breaks an assertion. The SMT encoding can be either in combination of propositional logic and LRA, or purely in propositional logic, possibly using previously computed function summaries (*top* of Fig. 2). The encoding is then inserted to the Interpolating SMT Solver which determines the satisfiability of the encoding. In case of unsatisfiability the

Table 4. Comparison between propositional and LRA encoding in HiFrog.

	LRA Results			LRA Time (s)			
Name (# asserts)	**Correct**	**SAT**	**Spurious**	**LRA**	**Spur-ref**	**Full-ref**	**Bool**
floppy1 (21)	16	5	5	27.5	193	193	192
tcas_asrt (162)	162	132	0	65.5	65.5	144	86.0
kbfiltr1 (10)	10	0	0	5.30	5.30	5.30	4.12
diskperf1 (14)	10	4	4	609	667	667	194
cafe (115)	105	95	10	4.75	5.90	14.8	19.2
s3 (131)	126	109	5	1.77	1.82	3.00	1.50
mem (149)	146	52	3	106	106	125	44.6
ddv (152)	56	105	96	12.5	123	124	260
disk (79)	62	72	17	800	1200	8710	8190
total (833)	693	574	140	1630	2370	9990	8900

solver reports that an assertion is safe, produces a Combined Interpolant using both SI-LRA and LIS-PROP, and stores the summary for future use. In case of satisfiability, the trace is provided to the Summary Refiner, which uses a refinement strategy to classify the trace either as spurious, triggering a re-check, or concrete, resulting in an error trace.

We first compare LRA-based incremental software verification using function summaries to the corresponding approach using propositional logic. To then develop a more in-depth understanding of the performance of SI-LRA we study different interpolants that HiFrog can construct during its search.

Modelling in LRA and propositional logic. Table 4 reports experimental results on how LRA can help the propositional solver in speeding up function-summary-based model-checking. The column *Name* reports the name and the number of assertions for each benchmark, while the columns *LRA Results* and *LRA time* report the results and the runtimes for the LRA-based model-checker. The column *Bool* reports the runtime on the purely propositional model.

The column *Correct* reports how many of the assertions could be immediately solved in the LRA model without refinement; the column *SAT* reports how many potential error traces could be found in the LRA model; and the column *Spurious* reports how many of these error traces were spurious in the end. Surprisingly many of the assertions can be checked only using LRA logic, as indicated by the big numbers in *correct*. However, in the current implementation, especially when resorting to the *Full-ref* mode, we see that the spuriousness checks result in a significant overhead. In the *Spur-ref* mode where a more intelligent strategy for refinement is used, the use of LRA in encoding provides almost a three-fold speed-up for the solving.

In general the experiments implicate that modelling with LRA can provide big speed-ups with respect to propositional models and that the LRA interpolation algorithms are not forming a bottleneck for the solver performance.

Table 5. Number of function refinements for HiFrog using different combinations of propositional and LRA interpolation algorithms.

Alg	**floppy1**	**tcas_asrt**	**kbfiltr1**	**diskperf1**	**cafe**	**s3**	**mem**	**ddv**	**disk**	Σ
M_s,Itp_M	27100	**53800**	**5120**	39900	**6400**	**0**	25600	**7940**	47600	214000
M_s,$Itp_{0.5}$	25100	**53800**	**5120**	39200	**6400**	**0**	**25100**	**7940**	**41500**	204000
M_s,Itp_D	24800	**53800**	5380	39200	**6400**	**0**	25600	**7940**	64000	227000
P,Itp_M	27100	**53800**	**5120**	39700	**6400**	**0**	25600	**7940**	47600	213000
P,$Itp_{0.5}$	25100	**53800**	**5120**	39200	**6400**	**0**	**25088**	**7940**	**41500**	204000
P,Itp_D	24800	**53800**	5380	39200	**6400**	**0**	25600	**7940**	64000	227000
M_w,Itp_M	27100	**53800**	**5120**	41500	**6400**	**0**	25600	**7940**	47600	215000
M_w,$Itp_{0.5}$	25100	**53800**	**5120**	**37400**	**6400**	**0**	**25100**	**7940**	**41500**	**20200**
M_w,Itp_D	**24600**	**53800**	5380	39200	**6400**	**0**	25600	**7940**	64000	227000
PS,Itp_M	27100	**53800**	**5120**	40200	**6400**	**0**	25600	**7940**	47600	214000
PS,$Itp_{0.5}$	25100	**53800**	**5120**	39200	**6400**	**0**	**25088**	**7940**	**41500**	204000
PS,Itp_D	**24600**	**53800**	5380	38100	**6400**	**0**	25600	**7940**	64000	226000
PS_w,Itp_M	27100	**53800**	**5120**	39700	**6400**	**0**	25600	**7940**	47600	213000
PS_w,$Itp_{0.5}$	25100	**53800**	**5120**	39200	**6400**	**0**	**25088**	**7940**	**41500**	204000
PS_w,Itp_D	24800	**53800**	5380	39900	**6400**	**0**	25600	**7940**	64000	228000
PS_s,Itp_M	27100	**53800**	**5120**	39200	**6400**	**0**	25600	**7940**	47600	213000
PS_s,$Itp_{0.5}$	25100	**53800**	**5120**	39200	**6400**	**0**	**25088**	**7940**	**41500**	204000
PS_s,Itp_D	**24600**	**53800**	5380	39200	**6400**	**0**	25600	**7940**	64000	227000

The SI-LRA evaluation. We compare the different combined interpolation algorithms using in total 18 combinations, where the propositional interpolation algorithms range over M_s, M_w, P, PS, PS_s, PS_w of LIS-PROP and the LRA interpolation algorithms range over Itp_M, Itp_D, and $Itp_{0.5}$, i.e., the strong and weak LRA interpolation algorithms and the interpolation algorithm with strength factor $\alpha = 0.5$, from SI-LRA.

One way to compare the behavior of different interpolation algorithms is to observe how many summary refinements are needed in model-checking a set of assertions. The lower the number of refinements is, the more relevant summaries the interpolation algorithm created. Table 5 reports the number of function refinements needed when running different interpolation algorithms for the instances in Table 4. We note that on most instances the interpolation algorithm does affect the required refinements, providing clear evidence that the choice of the interpolation algorithm can significantly affect the work flow of the model checker, resulting in the most extreme case (**disk**) in 35% difference in number of refinements between the extremes. Interestingly the SI-LRA interpolation algorithm $Itp_{0.5}$ provides most of the low refinement numbers, showing that neither one of the straightforward interpolation algorithms provides the most relevant interpolants for our benchmarks. It is also interesting to note that the strength

Table 6. Verification time for HiFrog using different combinations of propositional and LRA interpolation algorithms.

Alg	**floppy1**	**kbfiltr1**	**diskperf1**	**mem**	**disk**	Σ
M_s,Itp_M	28.2	5.23	604	106	809	1550
M_s,$Itp_{0.5}$	33.8	5.42	561	136	988	1720
M_s,Itp_D	28.2	5.28	476	107	1203	1820
P,Itp_M	28.0	5.38	587	104	**799**	**1520**
P,$Itp_{0.5}$	34.1	5.15	548	135	998	1720
P,Itp_D	28.1	5.38	**440**	106	1290	1870
M_w,Itp_M	**27.5**	5.30	609	106	800	1550
M_w,$Itp_{0.5}$	34.1	5.16	607	137	977	1760
M_w,Itp_D	27.9	5.45	475	106	1250	1860
PS,Itp_M	28.1	5.44	666	105	804	1610
PS,$Itp_{0.5}$	34.1	**5.10**	558	135	1000	1730
PS,Itp_D	28.0	5.57	473	107	1240	1850
PS_w,Itp_M	27.8	5.51	616	**104**	826	1580
PS_w,$Itp_{0.5}$	34.1	5.28	535	136	998	1710
PS_w,Itp_D	28.1	5.57	453	106	1250	1840
PS_s,Itp_M	28.1	5.49	604	105	815	1560
PS_s,$Itp_{0.5}$	34.2	5.12	549	136	996	1720
PS_s,Itp_D	27.8	5.62	446	107	1290	1880

of the LRA interpolants that led to the least number of refinements was not the strongest nor the weakest, showing that a fine strength tuning may lead to fast convergence in interpolation-based model checkers.

We report in Table 6 in addition the verification times for HiFrog using different combinations of propositional and LRA interpolation algorithms for the cases where the number of refinements was not constant over different interpolation algorithms. The average runtimes for the remaining instance **tcas_asrt**, **cafe**, **s3**, and **ddv** were 66.0, 4.70, 1.77, and 12.8 seconds, respectively, with small variance. Interestingly the per-instance winning algorithms are almost evenly distributed, making it hard to predict which algorithm provides the lowest run time on our benchmarks, the exception being the strong propositional algorithms M_s and PS_s, which score no wins. Inside each propositional algorithm Itp_M scores in total 18 wins compared to five wins of $Itp_{0.5}$ and 11 wins of Itp_D. However, for certain instances a given LRA algorithm is consistently better: in particular for **kbfiltr1** $Itp_{0.5}$ almost always wins, and for **diskperf1** Itp_D always wins. Finally we note that there is little correlation between the number of refinements and the run times, suggesting that the run time invested in the solving phase may pay off in higher quality interpolants in applications where convergence is the dominating performance criterion as opposed to run time.

In general our preliminary results suggest that there are interesting and non-trivial choices to be made when designing an efficient LRA interpolation algorithm that call for further analyses.

5 Conclusions

This work presents an interpolation system for linear real arithmetics, proves its correctness and orders the produced interpolants with respect to their logical strength, integrates the interpolation system to a propositional interpolation system, and provides experimental evaluation when used in a model-checking application. The system is based on computing an interpolant and its dual, and obtaining by shifting arbitrary "intermediary" interpolants that lie between the two extremes.

Experimental results in model checking suggest that the choice of the interpolant affects both run time and number of refinements, and mid-strength interpolants, i.e., $\alpha = 0.5$, result in small number of refinements. In the future we plan to apply SI-LRA in domains such as software model checking based on the PDR algorithm [11], as well as approaches for adapting the strength, starting with $\alpha = 0.5$, and tuning the factor depending on whether the generate interpolants are too precise or too abstract.

Acknowledgements. This work was supported by the SNF grants 200020_163001 and 200020_166288.

References

1. Albarghouthi, A., McMillan, K.L.: Beautiful interpolants. In: Proc. CAV 2013. pp. 313–329. No. 8044 in LNCS, Springer (2013)
2. Alt, L., Asadi, S., Chockler, H., Even-Mendoza, K., Fedyukovich, G., Sharygina, N.: HiFrog: SMT-based function summarization for software verification. In: Proc. TACAS 2017. pp. 207–2013. No. 10206 in LNCS, Springer (2017)
3. Alt, L., Fedyukovich, G., Hyvärinen, A.E.J., Sharygina, N.: A proof-sensitive approach for small propositional interpolants. In: Proc. VSTTE 2015. pp. 1–18. No. 9593, Springer (2016)
4. Alt, L., Hyvärinen, A.E.J., Asadi, S., Sharygina, N.: Duality-based interpolation for quantifier-free equalities and uninterpreted functions. In: Proc. FMCAD (2017), to appear.
5. Bogomolov, S., Frehse, G., Giacobbe, M., Henzinger, T.A.: Counterexample-guided refinement of template polyhedra. In: Proc. TACAS 2017. LNCS, vol. 10205, pp. 589–606 (2017)
6. Craig, W.: Three uses of the herbrand-gentzen theorem in relating model theory and proof theory. The Journal of Symbolic Logic 22(3), 269–285 (1957)
7. D'Silva, V.: Propositional interpolation and abstract interpretation. In: Proc. ESOP 2010. pp. 185–204. No. 6012 in LNCS, Springer (2010)
8. D'Silva, V., Kroening, D., Purandare, M., Weissenbacher, G.: Interpolant strength. In: Proc. VMCAI 2010. pp. 129–145. No. 5944 in LNCS, Springer (2010)

9. Dutertre, B., de Moura, L.: A fast linear-arithmetic solver for DPLL(T). In: Proc. CAV 2006. pp. 81–94. No. 4144 in LNCS, Springer (2006)
10. Gao, S., Zufferey, D.: Interpolants in nonlinear theories over the reals. In: Proc. TACAS 2016. pp. 625–641. No. 9636 in LNCS, Springer (2016)
11. Gurfinkel, A., Kahsai, T., Komuravelli, A., Navas, J.A.: The SeaHorn verification framework. In: Proc. CAV 2015. pp. 343–361. No. 9206 in LNCS, Springer (2015)
12. Huang, G.: Constructing craig interpolation formulas. In: Proc. COCOON 1995. LNCS, vol. 959, pp. 181–190. Springer (1995)
13. Hyvärinen, A.E.J., Asadi, S., Even-Mendoza, K., Fedyukovich, G., Chockler, H., Sharygina, N.: Theory refinement for program verification. In: Proc. SAT 2017. LNCS, vol. 10491, pp. 347–363. Springer (2017)
14. Hyvärinen, A.E.J., Marescotti, M., Alt, L., Sharygina, N.: OpenSMT2: An SMT solver for multi-core and cloud computing. pp. 547–553. No. 9710 in LNCS, Springer (2016)
15. Krajícek, J.: Interpolation theorems, lower bounds for proof systems, and independence results for bounded arithmetic. Journal of Symbolic Logic 62(2), 457–486 (1997)
16. Kroening, D., Strichman, O.: Decision Procedures: An Algorithmic Point of View, Second Edition. Texts in Theoretical Computer Science. An EATCS Series, Springer (2016)
17. McMillan, K.L.: An interpolating theorem prover. Theor. Comput. Sci. 345(1), 101–121 (2005)
18. McMillan, K.L.: Interpolation and SAT-based model checking. In: Proc. CAV 2003. pp. 1–13. No. 2725 in LNCS, Springer (2003)
19. Pudlák, P.: Lower bounds for resolution and cutting plane proofs and monotone computations. Journal of Symbolic Logic 62(3), 981–998 (1997)
20. Rollini, S.F., Alt, L., Fedyukovich, G., Hyvärinen, A.E.J., Sharygina, N.: PeRIPLO: A framework for producing effective interpolants in SAT-based software verification. In: Proc. LPRA-19. pp. 683–693. No. 8312, Springer (2013)
21. Rollini, S.F., Sery, O., Sharygina, N.: Leveraging interpolant strength in model checking. In: Proc. CAV 2012. pp. 193–209. No. 7358 in LNCS, Springer (2012)
22. Rümmer, P., Subotic, P.: Exploring interpolants. In: Proc. FMCAD 2013. pp. 69 – 76. IEEE (2013)
23. Rybalchenko, A., Sofronie-Stokkermans, V.: Constraint solving for interpolation. In: Proc. VMCAI 2007. pp. 346–362. No. 4349 in LNCS, Springer (2007)
24. Schlaipfer, M., Weissenbacher, G.: Labelled interpolation systems for hyper-resolution, clausal, and local proofs. Journal of Automated Reasoning 57(1), 3–36 (2016)
25. Scholl, C., Pigorsch, F., Disch, S., Althaus, E.: Simple interpolants for linear arithmetic. In: Proc. DATE 2014. pp. 1–6. European Design and Automation Association (2014)