Selfless Interpolation for Infinite-State Model Checking*

Tanja Schindler[1] and Dejan Jovanović[2]

[1] University of Freiburg
[2] SRI International

Abstract. We present a new method for interpolation in satisfiability modulo theories (SMT) that is aimed at applications in model-checking and invariant inference. The new method allows us to control the finite-convergence of interpolant sequences and, at the same time, provides expressive invariant-driven interpolants. It is based on a novel integration of model-driven quantifier elimination and abstract interpretation into existing SMT frameworks for interpolation. We have integrated the new approach into the SALLY model checker and we include experimental evaluation showing its effectiveness.

1 Introduction

Many modern model-checking techniques rely on Craig interpolation [14,30] as a learning oracle to support abstraction refinement and invariant inference. Interpolants themselves are artifacts usually computed from proofs of correctness for a finite unrolling of the system under analysis. While it is possible for a model checker to compute interpolants on its own, in most cases interpolation is provided by the underlying reasoning engine (such as an SMT solver) that is *unaware of the application-specific needs*. The importance of good interpolants is widely acknowledged – a single "magical" interpolant can make a difference between verifying the model instantaneously and verification failure – and it is no surprise that interpolants and their properties have been studied extensively. Some examples of interpolant properties are interpolant (logical) strength [16,38,34], size [18], and beauty [1].

Interpolant properties mentioned above are conceptually appealing but focus on single interpolants in isolation. In this paper we investigate interpolants as used in the IC3/PDR class of model-checking algorithms (e.g. [4,7,23,22]), in the context of analysis of infinite state systems. The IC3/PDR class of algorithms reasons locally, without unrolling the systems, and constructs abstractions and invariant candidates incrementally. The overall algorithm performs not one, but a sequence of reasoning queries that are interleaved and interact with results of interpolation. Therefore, we are ultimately more interested in *properties of the interpolation sequence*[3] rather than a single interpolant. Ideally, the interpolation

* The research presented in this paper has been supported by NSF grant 1528153.

[3] Not to be confused with sequence interpolants [29].

procedure would offer some convergence guarantees of this reasoning sequence. For example, in the PDKIND method [22] (a variant of PDR), interpolation is used to incrementally refine the current candidate invariant by examining induction failures. If this refinement sequence is allowed to continue indefinitely, the PDKIND method will never get to reason past its current invariant candidate and will fail to make progress. Similarly, in IC3/PDR algorithms, a non-converging interpolation sequence will result in verification failure where all reachability frames are refined indefinitely.

Example 1 (Model Checking Divergence). Consider a simple transition system defined with the initial states and a transition relation:

$$I = (x = 0) \wedge (c = 1) \ , \qquad T \equiv (x' = x + c) \wedge (c' = 2c) \ .$$

The system above satisfies the invariant ($x \geq 0$). Nevertheless, well-known interpolation-based model checkers such as NUXMV [5][4] and SPACER [23], and our own tool SALLY [22] diverge and fail to prove the property. □

We propose a new interpolation method that is based on the following two guiding principles:

1. The interpolation method should interact well with the underlying model-checking algorithm and offer guarantees of convergence for interpolation sequences.
2. The interpolation method should be aware of the model checking context and be able to accept suggestions from the model checker in order to produce invariant-driven interpolants.

Our new method is developed within the interpolation framework available in all major SMT solvers (e.g. [10]). For the theory of arithmetic, most SMT solvers produce interpolants through application of the Farkas lemma [35]. We first show how this approach can result in divergence, as in the example above, and then propose a solution based on model-driven quantifier elimination that guarantees convergence. The new method is very flexible and allows the model checker to also provide interpolant suggestions. We use this feature to make the resulting interpolants invariant-directed, by integrating abstract interpretation [12] into the interpolation process. To the best of our knowledge, our interpolation method is the first to combine interpolation, quantifier elimination, and abstract interpretation, in a framework readily reusable in any interpolating SMT solver.

We have implemented our new approach into the SALLY model-checker by relying on the MATHSAT5 SMT solver [9] and the APRON domain library [20]. We present experimental data that shows the effectiveness of the approach and illuminates the impact of quantifier elimination and abstract interpretation.

[4] In order to rely only on interpolation, we disable predicate abstraction in NUXMV.

2 Background

2.1 Satisfiability Modulo Theories

We work in the setting of satisfiability modulo theories (SMT) and assume the usual notation of first order logic (see, e.g. [2]). In the following, we use the letters x, y, z to denote variables, and c to denote constants. We consider the quantifier-free theory of linear arithmetic over the rationals ($\mathcal{T}_{\text{LA}}$). We denote with p and q linear terms over $\mathbb{Q}$, i.e., terms of the form $c_n x_n + \cdots + c_1 x_1 + c_0$ over variables $\boldsymbol{x} = \langle x_1, \ldots, x_n \rangle$ with coefficients $c_0, \ldots, c_n \in \mathbb{Q}$. A theory atom in arithmetic is a linear constraint, i.e., an inequality of the form $(p \diamond 0)$ with $\diamond \in \{\leq, <\}$,[5] and a literal is an atom or its negation. A clause is a disjunction of literals and we denote with $\bot$ the empty clause. As usual, formulas are constructed inductively from atoms and the usual Boolean connectives. We denote with $\text{ATOMS}(F)$ the set of all atoms appearing in a formula F, and with $\text{LITERALS}(F)$ the set of all literals of F, i.e. $\text{LITERALS}(F) = \{a, \neg a \mid a \in \text{ATOMS}(F)\}$. If formula A has all its free variables in $\boldsymbol{x}$, we denote this with $A(\boldsymbol{x})$. A formula $A(\boldsymbol{x})$ is satisfiable if there is an assignment mapping its variables to real values such that A evaluates to **true** in the usual interpretation. A conjunction of literals C is a $\mathcal{T}_{\text{LA}}$-conflict, if C is inconsistent with $\mathcal{T}_{\text{LA}}$. A $\mathcal{T}_{\text{LA}}$-lemma is a clause C such that $\neg C$ is a $\mathcal{T}_{\text{LA}}$-conflict or, in other words, C is a valid statement in arithmetic. To ease notation, we also treat conjunctions and clauses as sets of literals.

Given a formula F that is unsatisfiable in $\mathcal{T}_{\text{LA}}$, a resolution proof of F is a tree $\mathcal{P}$ such that a) the root of $\mathcal{P}$ is the empty clause, b) leafs of $\mathcal{P}$ are either clauses from F, or $\mathcal{T}_{\text{LA}}$-lemmas, and c) each non-leaf node C is an application of Boolean resolution, i.e. $C \equiv (C_1 \vee C_2)$ and has two parents $(C_1 \vee l)$ and $(\neg l \vee C_2)$ as below.

$$\frac{(C_1 \vee l) \qquad (\neg l \vee C_2)}{(C_1 \vee C_2)}$$

Most modern SMT solvers rely on some variant of the DPLL($\mathcal{T}$) framework [32] to check satisfiability of formulas. In this framework, to solve a formula F, a CDCL SAT solver is used to enumerate the truth values of the propositional skeleton of the formula F. As the formula atoms are assigned to **true** or **false** by the SAT solver, a dedicated decision procedure for the theory $\mathcal{T}$ checks the consistency of the literals $A \subseteq \text{LITERALS}(F)$ corresponding to the Boolean assignment. If A is unsatisfiable, the decision procedure returns a $\mathcal{T}$-conflict $C \subseteq A$ (or equivalently a $\mathcal{T}$-lemma $\neg C$) that explains the inconsistency in Boolean terms to the SAT solver. The basic DPLL($\mathcal{T}$) framework can be extended to also provide proofs of unsatisfiability by stitching together the Boolean reasoning and the $\mathcal{T}$-lemmas resulting from the conflicts. In proofs generated by a DPLL($\mathcal{T}$) solver, by construction, the clauses that make up the proof *only contain atoms from the original formula.* [6]

[5] For simplicity we do not consider equality, since they can be eliminated by rewriting $(p = 0)$ with $(p \leq 0) \wedge \neg(p < 0)$.

[6] In general, SMT solvers may introduce new literals to support reasoning in more expressive theories, but for reasoning in $\mathcal{T}_{\text{LA}}$ this is unnecessary.

In the case of linear real arithmetic, we are talking about DPLL($\mathcal{T}_{\text{LA}}$), and the decision procedure most commonly used is based on a variant of the Simplex algorithm [17] engineered specifically for DPLL($\mathcal{T}_{\text{LA}}$). Besides its efficiency, this Simplex algorithm also has a remarkable property that the $\mathcal{T}_{\text{LA}}$-conflicts that it produces are *minimal*.

2.2 Craig Interpolation

Definition 1 (Craig interpolant). *Given two formulas $A(\boldsymbol{x}, \boldsymbol{y})$ and $B(\boldsymbol{y}, \boldsymbol{z})$ such that $A \wedge B$ is unsatisfiable, a* Craig interpolant *is a formula $J(\boldsymbol{y})$ such that $A \Rightarrow J$ and $J \Rightarrow \neg B$. We call the pair (A, B) an* interpolation problem.

The formulation above is the general version of the interpolation problem. In model checking applications, interpolation problems are often specialized so that the formulas in question are of the form $A(\boldsymbol{x}, \boldsymbol{x'})$ and $B(\boldsymbol{x'})$. In such cases, it is easy to see that $J \equiv \neg B$ is a solution to the interpolation problem, and we call J the *trivial interpolant*.

Interpolants for the interpolation problem (A, B) can be computed from a proof of unsatisfiability of the formula $A \wedge B$. The underlying technique of this proof-based interpolation is generally attributed to Pudlák [19,25,33], and was revisited in recent years in the context of model checking and SMT solving (see, e.g. [27,28,10]). Given a proof of unsatisfiability for $A \wedge B$, the interpolant can be computed inductively over the structure of the proof tree. As the proof tree is traversed from the leaves to the root, each clause C in the proof tree is associated a partial interpolant. A *partial interpolant* of a clause C is an interpolant of the formulas

$$(A \wedge (\neg C \downarrow A)) \ , \qquad\qquad (B \wedge (\neg C \downarrow B)) \ .$$

The projection functions $(\cdot \downarrow A)$ and $(\cdot \downarrow B)$ operate on literals and have the following properties. For a literal l, one of $(l \downarrow A)$ and $(l \downarrow B)$ must be l. The projection $(l \downarrow A)$ may be l, if $l \in \text{LITERALS}(A)$, or it must be $\top$ otherwise. Dually, $(l \downarrow B)$ may be l, if $l \in \text{LITERALS}(B)$, or it must be $\top$ otherwise. The projection function is extended to conjunctions of literals, as expected. Intuitively, the projection functions are used to extract literals that come from A or B, with some flexibility in the ownership of shared literals.[7] For a clause C, we write $\neg C \setminus A$ as a shortcut for the literals in $\neg C$ without $(\neg C \downarrow A)$, and analogously for B. Note that the partial interpolant of the empty clause $\bot$ is an interpolant of the original problem that can be read of as the partial interpolant associated to the root of the proof. The rules for computing partial interpolants depend on the type of the proof node, as follows.

1. For an input clause C from A the partial interpolant is $\neg(\neg C \setminus A)$.
2. For an input clause C from B the partial interpolant is $\neg C \setminus B$.

[7] This flexibility allows interpolants of different logical strength [16].

3. For a $\mathcal{T}$-lemma C, the conjunction $\neg C$ is unsatisfiable, and the partial interpolant is computed by a $\mathcal{T}$-lemma interpolator as the interpolant of $(\neg C \downarrow A)$ and $(\neg C \downarrow B)$.
4. For a resolvent clause C the partial interpolant J is computed as follows

$$\frac{(C_1 \vee l) : J_1 \qquad (\neg l \vee C_2) : J_2}{(C_1 \vee C_2) : J} \text{ where } J = \begin{cases} J_1 \vee J_2, & \text{if } l \downarrow B = \top, \\ J_2 \wedge J_2, & \text{if } l \downarrow A = \top, \\ ite(l, J_2, J_1), & \text{otherwise.} \end{cases}$$

For a general and more detailed exposition on the overall framework we refer the reader to [6]. Given a $\mathcal{T}$-lemma interpolator P, we denote with $\text{ITP}[\![P]\!]$ the proof-based interpolation procedure that uses P to interpolate the $\mathcal{T}$-lemma nodes of the proof.

In general, the structure of proof-based interpolants is hard to control: an interpolant will be a Boolean combination of parts of A clauses, parts of B clauses, and the interpolants from the $\mathcal{T}$-lemmas. Nevertheless, we can guarantee that no literals that are exclusively in B can sneak into the interpolant, unless introduced by the lemmas.

Lemma 1. *Given an interpolation problem (A, B), the interpolant $J = \text{ITP}[\![P]\!]$ is a Boolean combination over the atoms of A and atoms from the $\mathcal{T}$-lemma interpolants.*

Proof. We only have to show that no atom from B, that neither appears in A nor is produced by the $\mathcal{T}$-lemma interpolator, can ever sneak into the final interpolant. This is trivial for the partial interpolants for input clauses from A and theory lemmas. The only atoms that can be added to the interpolant in resolution nodes, are atoms for which $l \downarrow A = l$ (and $l \downarrow B = l$) holds, and hence $l \in \text{LITERALS}(A)$. We are left with the case of a proof node that is an input clause from B. In this case the partial interpolant is $J \equiv (\neg C \setminus B)$. By definition, for each $l \in J$ we know that $l \downarrow B = \top$. Therefore we must have $l \downarrow A = l$ and $l \in \text{LITERALS}(A)$. □

2.3 Arithmetic Interpolation

In the proof-based interpolation framework, for an SMT solver to provide interpolation in the theory of arithmetic, it needs to be able to provide interpolation for each $\mathcal{T}_{\text{LA}}$-theory lemma that it contributes to the proof. For an interpolation problem (A, B), the lemmas of the proof correspond to $\mathcal{T}_{\text{LA}}$-conflicts that were found by the solver during the solving process. In case of arithmetic, and SMT solvers based on Simplex, each $\mathcal{T}_{\text{LA}}$-conflict will be a set of literals C that is inconsistent (and minimal). Each conflict C can be separated into the A part and B part, and the goal is to find an interpolant for the interpolation problem $(C \downarrow A, C \downarrow B)$.

For linear arithmetic, the most common way to obtain the interpolant of a conflict is to rely on the Farkas lemma. A $\mathcal{T}_{\text{LA}}$-theory conflict C is an unsatisfiable

conjunction of inequalities

$$I_i \equiv (\sum_j c_{ij} x_j + c_{i0} \ \diamond_i \ 0) \ ,$$

for $\diamond_i \in \{<, \leq\}$. By Farkas lemma, there exist coefficients $k_i > 0$ that can certify the inconsistency, i.e. such that

$$\sum_i k_i \times I_i = (1 < 0) \ .$$

The lemma interpolant can then be given by summing up the A contributions to the conflict, i.e., the interpolant is

$$J \equiv \sum_{I_i \in C \downarrow A} k_i \times I_i \ .$$

It is not hard to see that J is a valid interpolant. The advantage of the Farkas approach is that the coefficients k_i can easily be read off the state of the Simplex solver when it detects a conflict. We will denote the $\mathcal{T}_{\text{LA}}$-lemma interpolator based on the Farkas lemma as P_{FK}.

Note that an interpolant obtained with P_{FK} is always *a single inequality*. The ability to produce a single inequality can be advantageous, as it allows the interpolant to relate variables that might not be syntactically related in A, by using the B part. On the other hand, as we will see in the next section, the disadvantage of P_{FK} is that it can lead to diverging interpolant sequences.

3 Sequences of Interpolants

Behavior of interpolant sequences was first explored in [22], where the notion of finite-covering interpolation was proposed as an assumption that supports termination and deductive power of the PDKIND method.

Definition 2 (Finite Covering Interpolation). *An interpolation procedure P (or a $\mathcal{T}$-lemma interpolator) is* finite covering *if for a fixed $A(\boldsymbol{x}, \boldsymbol{y})$, it can only produce a finite number of distinct interpolants.*

Example 2. If the interpolation problems are of the form $A(\boldsymbol{x}, \boldsymbol{x'})$ and $B(\boldsymbol{x'})$, the trivial interpolation method can always return $\neg B$ as the interpolant. This kind of interpolation is not useful in general and is not finite covering.

Finite covering is a strong property. Most interpolation procedures are proof-based and, since the space of proofs and lemmas is infinite, they do not ensure finite covering. Nevertheless, for theories that admit quantifier elimination, for any given A, one can construct a single interpolant $J(\boldsymbol{y})$ by eliminating $\boldsymbol{x}$ from $(\exists \boldsymbol{x} \ . \ A(\boldsymbol{x}, \boldsymbol{y}))$ that refutes any B that needs to be interpolated. In principle, for arithmetic theories, a finite-covering interpolation procedure could be devised

by relying on procedures such as MCSat [15] that are based on quantifier elimination. But, since none of the available interpolating SMT solvers are MCSat-based, it would be desirable to have some control over the number of potential invariants in the existing proof-based interpolation framework.

Definition 3 (Interpolation Sequence). *Given a formula $A(\boldsymbol{x}, \boldsymbol{y})$ and two sequences of formulas $(J_k(\boldsymbol{y}))$ and $(B_k(\boldsymbol{y}, \boldsymbol{z}))$, we call (J_k) an* interpolation sequence *for A and (B_k) if for all k it holds that*

1. *B_k is consistent with $\bigwedge_{i<k} J_i$;*
2. *B_k is inconsistent with A;*
3. *J_k is the interpolant between A and B_k.*

Definition 4 (Finite Convergence). *We say that an interpolation procedure has a* finite convergence property *if it does not allow infinite interpolation sequences.*

To put the definitions above in perspective, in a typical model checking application, the formula A will correspond to some abstraction of reachable states (including the transition relation), formulas B_k will correspond to potentially bad states, and the interpolants J_k will be learned facts that refute the potentially bad states. The finite convergence property then guarantees that no matter how we choose the potentially bad states, the interpolation procedure will eventually refute all of them. Finite convergence differs from finite covering in that it is semantic and more directly addresses the undesirable interpolant behavior.

Example 3 (Finite Convergence). Consider the interpolation procedure $\text{ITP}[\![P_{\text{FK}}]\!]$, i.e. the standard SMT interpolation for $\mathcal{T}_{\text{LA}}$ based on Farkas derivation. Let $A(x, y_1, y_2)$ be the constraints

$$I_1 \equiv (y_1 - x < 0) \ , \qquad I_2 \equiv (x < 0) \ , \qquad I_3 \equiv (y_2 - x < 0) \ .$$

Now, consider the sequence of formulas (B_k), where $B_k(y_1, y_2) \equiv (y_1 + k y_2 > 0)$. Interpolating from A against an individual B_k using the Farkas approach will always result in an interpolant J_k that is a single inequality constructed as a combination of formulas from A, i.e., we will obtain

$$J_k \equiv 1 \times I_1 + (k+1) \times I_2 + k \times I_3 \equiv (y_1 + k y_2 < 0) \ .$$

Since interpolating from A over the sequence (B_k) results in an infinite sequence of distinct interpolants, the Farkas approach to interpolation does not have the finite-covering property, i.e., neither P_{FK} nor $\text{ITP}[\![P_{\text{FK}}]\!]$ guarantee finite convergence.

On the other hand, we can rely on Fourier-Motzkin quantifier elimination to simply eliminate x from A and obtain the conjunction $J \equiv (y_1 < 0) \wedge (y_2 < 0)$ that is a suitable interpolant for all B_k simultaneously (it is derivable from A and singlehandedly refutes all B_k). Note that, as mentioned before, the Farkas-based interpolants relate variables y_1 and y_2 in the interpolants. On the other hand, the interpolants based on Fourier-Motzkin do not. □

An interesting property of finite convergence is that we can interleave an interpolation procedure P_1 with finite convergence with an arbitrary interpolation procedure P_2 and still obtain finite convergence, as long as the interleaving is fair to the procedure P_1.[8]

The following lemma shows that we do not need to devise an entirely new interpolation procedure to ensure finite convergence. Instead, we only need to devise a finite-covering $\mathcal{T}$-lemma interpolator that can then be used in the standard proof-based interpolation framework.

Lemma 2. *If a $\mathcal{T}$-lemma interpolator P is finite covering, then the proof-based interpolation procedure* $\text{ITP}[\![P]\!]$ *has the finite convergence property.*

Proof. Assume that $\text{ITP}[\![P]\!]$ does not have the finite convergence property. This means that there is an infinite interpolation sequence, i.e. there is a formula A, and two sequences of formulas J_k and B_k as in Definition 3. In this sequence, the interpolants J_k must be distinct functions because for each k

- J_k is inconsistent with B_k; but
- J_i is consistent with B_k, for $i < k$.

On the other hand, P can only produce a finite number of lemma interpolants and, by Lemma 1, $\text{ITP}[\![P]\!]$ (as a proof-based procedure) can only produce Boolean combinations of clauses from A and lemma interpolants. Therefore, the overall procedure $\text{ITP}[\![P]\!]$ will only be able to produce a finite number of distinct interpolants (seen as functions), proving the case by contradiction. □

4 Interpolation with Conflict Resolution

In this section we present a $\mathcal{T}_{\text{LA}}$-lemma interpolator P_{CR} that replaces the traditional interpolator based on the Farkas lemma P_{FK}. Throughout this section we therefore assume a global interpolation problem, i.e. formulas $A(\boldsymbol{x}, \boldsymbol{y})$ and $B(\boldsymbol{y}, \boldsymbol{z})$, with $A \wedge B$ unsatisfiable. In addition, we assume a global ordering on variables so that $\boldsymbol{z} \prec \boldsymbol{y} \prec \boldsymbol{x}$. Our goal is to devise the $\mathcal{T}_{\text{LA}}$-lemma interpolator P_{CR} that is finite covering. In order to achieve this we will rely on a model-driven variant of Fourier-Motzkin (FM) quantifier elimination.

4.1 Fourier-Motzkin Elimination

Given an inconsistent set of inequalities F, a FM proof of F has the same structure as a Boolean resolution proof would, but with clauses replaced with inequalities, and the resolution rule replaced with the FM elimination rule. Given two inequalities sharing a variable x of opposite signs, the FM rule deduces a new inequality with this variable eliminated.[9]

[8] In a way, an interpolation procedure that has the finite convergence property is analogous to the widening operator in abstract interpretation.

[9] The presented rule is over strict inequalities only, other cases are as expected.

$$\text{FM } x \; \frac{I_l \equiv (p - x < 0): \qquad I_u \equiv (x - q < 0)}{R \equiv (p - q < 0)}$$

We first explain the general idea behind the new lemma interpolation procedure. Each $\mathcal{T}_{\text{LA}}$-lemma interpolation problem consists of two sets of inequalities $C_A(\boldsymbol{x}, \boldsymbol{y})$ and $C_B(\boldsymbol{y}, \boldsymbol{z})$, with $C_A \wedge C_B$ unsatisfiable. Therefore, there exists a FM elimination proof of inconsistency that is ordered according to $\prec$. In other words, in the FM proof the $\boldsymbol{x}$ variables are eliminated first, followed by the $\boldsymbol{y}$ variables, and finally the $\boldsymbol{z}$ variables. The order of elimination ensures, for example, that if any inequality I in the proof contains an $\boldsymbol{x}$ variable, I must have been derived from C_A. Let J be the set of inequalities in the proof that do not contain any $\boldsymbol{x}$ variables but were either a) derived from two inequalities that contain $\boldsymbol{x}$ variables; or b) appear in C_A directly. By construction, then $C_A \Rightarrow J$ and J is in $\boldsymbol{y}$ variables only. In addition, the inequalities in J constitute a cut of the proof tree that is enough to refute C_B. In other words, J is an interpolant between C_A and C_B. This selection of inequalities from the proof can be done locally at each resolution node, and we denote the procedure that returns the relevant inequalities as $\text{SELECT}(R, I_l, I_u)$.

Example 4. Let's revisit the interpolation problem of Example 3, i.e., let

$$C_A \equiv (y_1 - x < 0) \wedge (x < 0) \wedge (y_2 - x < 0) \;\; , \qquad C_{B_k} \equiv (y_1 + k y_2 > 0) \;\; .$$

Below is a Fourier-Motzkin proof of unsatisfiability of $C_A \wedge C_{B_k}$, with the variables ordered as $y_2 \prec y_1 \prec x$. We mark inequalities derived only from C_A with red bold font, and all other inequalities with blue.

$$\dfrac{\dfrac{\dfrac{\mathbf{y_1 - x < 0} \qquad \mathbf{x < 0}}{\mathbf{x}} \qquad -y_1 - k y_2 < 0}{y_1} \qquad \dfrac{\mathbf{y_2 - x < 0} \qquad \mathbf{x < 0}}{\mathbf{x}}}{y_2} \qquad \begin{array}{c} \downarrow \\ \mathbf{y_2 < 0} \end{array}$$
$$\dfrac{-k y_2 < 0}{0 < 0}$$

As discussed above, we can examine the proof and get that

$$\begin{aligned} &\text{SELECT}((y_1 < 0), (y_1 - x < 0), (x < 0)) = \{(y_1 < 0)\} \;\; , \\ &\text{SELECT}((y_2 < 0), (y_2 - x < 0), (x < 0)) = \{(y_2 < 0)\} \;\; . \end{aligned}$$

Therefore the set of inequalities $J = \{\, (y_1 < 0), (y_2 < 0) \,\}$ is an interpolant for C_A and C_{B_k} for *any* k. □

4.2 Conflict Resolution

Although we could use FM elimination to derive the $\mathcal{T}_{\text{LA}}$-lemma interpolants, as above, this would likely not be efficient. FM elimination is a quantifier elimination procedure and can become very inefficient even with small numbers of variables. Instead, we will adopt a model-driven variant of FM elimination called *conflict resolution* (CR). The conflict resolution algorithm was originally introduced in [24] for solving systems of linear inequalities. CR is an instance of a

recent class of model-based decision procedures, such as Generalized DPLL [26] and MCSat [15], but is simpler as it targets conjunctions of constraints only. The algorithm is related to FM elimination in the same way the CDCL algorithm is related to Boolean resolution: instead of trying to prove the problem unsatisfiable by saturating the FM rule, conflict resolution attempts to build a model and only applies the FM rule when the model-building fails. This principled way of deriving new inequalities makes it possible to produce a proof while only deducing inequalities that are relevant for unsatisfiability.

We use a variation of the original algorithm [24] adapted to the context of $\mathcal{T}_{\text{LA}}$-lemma interpolation. In this context, we are given two sets of inequalities $C_A(\boldsymbol{x}, \boldsymbol{y})$ and $C_B(\boldsymbol{y}, \boldsymbol{z})$ that together are known to be unsatisfiable. The algorithm will construct a proof that $C_A \wedge C_B$ is unsatisfiable and, as a side-effect, collect the set of inequalities J that will form the interpolant of C_A and C_B. Before we describe the algorithm itself, we go trough some of its ingredients.

We order all the variables so that $\langle v_1, \ldots, v_n \rangle = \langle \boldsymbol{z}, \boldsymbol{y}, \boldsymbol{x} \rangle$ and call i the *level* of variable v_i. A variable v_i is the *top variable* in I, if v_i is the largest variable in I with respect to $\prec$, and we denote with $\textsc{level}(I)$ the function that returns the level i. Given a set of inequalities $\mathcal{I}$, we can partition it by level and we denote with $\mathcal{I}_v$ the set of all inequalities from $\mathcal{I}$ with v as the top variable.

The algorithm maintains an assignment σ of variables to values in $\mathbb{Q}$. Any inequality I with v_i as the top variable implies a bound on the possible values that v_i can take with respect to the current assignment of $v_1, \ldots, v_{i-1}$. For example, if $I \equiv (v_i + p \leq 0)$, then the implied bound is $v_i \leq -\sigma(p)$. For each variable v_i, the algorithm maintains an interval $\textsc{feasible}[v_i] = (l, u)$ that represents the strongest lower and upper bounds inferred on v_i. Additionally, the bounds of this interval are associated with the inequalities $I_l[v_i], I_u[v_i]$ that imply them. We say that the variable v_i is in conflict, denoted with $\textsc{in-conflict}(v_i)$, if the current lower and upper bounds on v_i are in conflict, i.e., when $\textsc{feasible}[v_i]$ is either a (half-)open interval with $l \geq u$, or a closed interval with $l > u$.

The main inference mechanism in the algorithm is *bound propagation* on inequalities, which is an arithmetic analogue to unit propagation that SAT solvers perform on clauses. Given an inequality I with v_i as its top variable, we denote with $\textsc{propagate-bounds}(I, v_i)$ the procedure that computes the bound that I implies on v_i and updates the bound information if the new bound is stronger than the existing one. We overload bound propagation to operate over a set of inequalities $\mathcal{I}_{v_i}$ with v_i as its top variable, and denote with $\textsc{propagate-bounds}(\mathcal{I}_{v_i}, v_i)$ the procedure that resets the current bound information on v_i and then updates it by propagating bounds over all inequalities in $\mathcal{I}_{v_i}$. If, after performing exhaustive propagation over $\mathcal{I}_{v_i}$, the variable v_i is not in conflict, then we can safely pick a value $\alpha \in \textsc{feasible}[v_i]$, which we denote with $\textsc{pick-value}(v_i)$.[10] In this case, by construction, it is guaranteed that the value can be used to satisfy $\mathcal{I}_{v_i}$, i.e., $\sigma\{v_i \mapsto \alpha\} \vDash \mathcal{I}_{v_i}$.

The algorithm starts at level 1 and tries to gradually build a satisfying assignment σ for the variables $\boldsymbol{v}$, by assigning them values one by one. We know

[10] For example we can pick $\frac{l+u}{2}$, which is what we do in our implementation.

Algorithm 1 Interpolation with Conflict Resolution.

Require: Sets of inequalities $C_A(\boldsymbol{x}, \boldsymbol{y})$ and $C_B(\boldsymbol{y}, \boldsymbol{z})$, known to be inconsistent.
Ensure: Set of inequalities J is an interpolant for C_A and C_B.

```
1  function P_CR(C_A, C_B)
2      v ← ⟨z, y, x⟩                                  ▷ order the variables z ≺ y < x.
3      i ← 1; I ← C_A ∪ C_B; J ← ∅                    ▷ initialize and start from bottom
4      loop
5          PROPAGATE-BOUNDS(I_{v_i}, v_i)             ▷ compute bounds for v_i
6          while IN-CONFLICT(v_i) do                  ▷ resolve any conflicts
7              R ← FM-RESOLVE(I_l[v_i], I_u[v_i], v_i)   ▷ compute the resolvent
8              J ← J ∪ SELECT(R, I_l[v_i], I_u[v_i])     ▷ add relevant inequalities to J
9              if (R ≠ ⊥) then                        ▷ backtrack with resolvent
10                 i = LEVEL(R)                       ▷ level to backtrack to
11                 I = I ∪ {R}                        ▷ remember the new inequality
12                 PROPAGATE-BOUNDS(R, v_i)           ▷ update bounds with new inequality
13             else return J                          ▷ J is the interpolant.
14         σ[v_i] ← PICK-VALUE(v_i)                   ▷ pick a value for v_i in FEASIBLE(v_i)
15         i ← i + 1                                  ▷ continue with next variable
```

that a complete model does not exist, and the failed model-building attempts will guide the process of FM resolution. At each level i the algorithm performs bound propagation to compute the interval of potential values that the variable v_i can take with respect to the assignment of variables $v_1, \ldots, v_{i-1}$. If bound propagation produces a feasible interval, then the algorithm assigns to v_i a value in this interval and moves on to the next variable. Otherwise, IN-CONFLICT(v_i) is true, and there are two inequalities[11]

$$I_l[v_i] \equiv (p - v_i < 0) \ , \qquad I_u[v_i] \equiv (v_i - q < 0) \ ,$$

such that the bounds they imply on v_i are inconsistent, i.e., we know that $\sigma(p) \geq \sigma(q)$. Mimicking a SAT solver, we can resolve this conflict by applying Fourier-Motzkin resolution to derive the resolvent $R = (p - q < 0)$. We denote the resolution inference over inequalities I_1 and I_2 that eliminates variable v_i with $R =$ FM-RESOLVE(I_1, I_2, v_i). The inequality R is a potential new node in the FM proof, and we examine the proof inference and add any relevant inequalities to the interpolant J. In addition we use the resolvent R to backtrack as follows. Since the resolvent R does not include v_i, it must be of level less than i. We also know by $\sigma(p) - \sigma(q) \geq 0$ that R is inconsistent with the current model, i.e. that $\sigma \not\models R$. If $R \equiv \bot$, we have found the proof of unsatisfiability and the set of inequalities J is the final interpolant. Otherwise, we use R to backtrack to the level of R and update the bounds of its top variable with new information.

Properties of P_{CR}. The termination and correctness of the algorithm follows from the termination and correctness of the original conflict resolution algorithm [24], and the fact that we can obtain the interpolant from the computed FM proof.

[11] For simplicity we only consider the case of strict inequalities, other cases are similar.

Another way of looking at the P_{CR} algorithm is as a semantic interpolation game where each model that can be constructed for C_B inequalities is refuted by an inequality derived from C_A.

FM elimination allows us to put a bound on the number of literals that can appear in the interpolant J. First, for a fixed global A, and any lemma interpolation problem (C_A, C_B), we know that $\text{LITERALS}(C_A) \subseteq \text{LITERALS}(A)$. Therefore, the inequalities that can appear in J are limited to the inequalities that one can obtain by FM elimination on $\text{LITERALS}(A)$. From this bound and Lemma 2 we can then show the following two properties of P_{CR}.

Lemma 3. *P_{CR} is a finite-covering $\mathcal{T}_{\text{LA}}$-lemma interpolator.*

Lemma 4. *$\text{ITP}[\![P_{\text{CR}}]\!]$ interpolation procedure has the finite convergence property.*

5 Improving Interpolation with Abstract Interpretation

The P_{CR} lemma interpolator described in the previous section ensures finite convergence of interpolation sequences. This is achieved by restricting the language of the potential interpolants by relying on quantifier elimination. Since our interest in interpolation comes from its use in construction of invariants, this also restricts the potential invariants that we can construct and can be seen as a disadvantage of the method. In this section we consider the interpolation problem specifically in the context of model checking and invariant inference and try to remedy this.

Model Checking. We assume a finite set of variables $\boldsymbol{x}$ called state variables. To each variable $x \in \boldsymbol{x}$, we associate its primed version x'. We call any formula $F(\boldsymbol{x})$ over the state variables a state formula, and any formula $T(\boldsymbol{x}, \boldsymbol{x}')$ a state-transition formula. A state-transition system is a pair $\mathfrak{S} = \langle I, T \rangle$, where $I(\boldsymbol{x})$ is a state formula describing the initial states and $T(\boldsymbol{x}, \boldsymbol{x}')$ is a state-transition formula describing the system's evolution.

Example 5. Let $\mathfrak{S} = \langle I, T \rangle$ be a transition system defined as

$$I \equiv (x = 0) \wedge (y = 0) \ , \qquad T \equiv (x' = x + 1) \wedge (y' = y + 1) \ .$$

It is easy to see that $(x = y)$ is an invariant of $\mathfrak{S}$. Nevertheless, consider a typical query that a model checker would use to check if a potential bad state $(x < y)$ is reachable.

$$\overbrace{I(x,y) \wedge T(x,y,x',y')}^{C_A} \wedge \overbrace{(x' < y')}^{C_B} \ .$$

The query above is unsatisfiable, and we can use it to derive an interpolant to use in invariant inference. Unfortunately, by using P_{CR}, since we are only inferring inequalities from C_A, we can never deduce an inequality that relates the variables x and y, and the resulting interpolant is $(x' = 1) \wedge (y' = 1)$.

On the other hand, because it must produce a single constraint, the Farkas-based P_{FK} will relate the variables x and y, and produce the desired interpolant as follows

$$(x' = x + 1) + (y' = y + 1) + (x = 0) + (y = 0) \equiv (x' = y') \ .$$

This remarkable capacity of P_{FK} to relate relevant variables is probably one of the main reasons for its successful adoption. □

Abstract Interpretation. As the example above shows, restricting the language of interpolants with quantifier elimination can put P_{CR} at a loss when inferring invariants. In order to improve the invariant-inference capacity of P_{CR}, we will rely on the tools provided by one of the most successful frameworks for invariant inference – abstract interpretation [12]. Abstract interpretation is a theory for sound approximation of the semantics of transition systems. The appeal of abstract interpretation is that it can efficiently compute a superset of all possible behaviors of a system by means of abstractions. These abstractions are computed by abstracting the semantics of the system with semantic techniques that are orthogonal to the syntactic, proof-based approach of quantifier elimination. Abstract interpretation provides a range of abstract domains $\mathcal{D}^{\#}$ that can be used for approximating $\mathcal{T}_{\text{LA}}$ transition systems, such as the interval [11], octagon [31], and the polyhedra [13] domains.

Since we are working in the context of model checking, we assume a global interpolation problem of the form

$$A(\boldsymbol{x}, \boldsymbol{x}') \equiv (I(\boldsymbol{x}) \wedge T(\boldsymbol{x}, \boldsymbol{x}')) \ , \qquad\qquad B(\boldsymbol{x}') \ ,$$

with $A \wedge B$ unsatisfiable. Note that the usual SMT interpolation procedures *do not have this information* (the relationship between $\boldsymbol{x}$ and $\boldsymbol{x}'$ variables has to be provided by the model checker).

As part of the proof of unsatisfiability of $A \wedge B$, we also assume a $\mathcal{T}_{\text{LA}}$-conflict separated into $C_A(\boldsymbol{x}, \boldsymbol{x}')$ and $C_B(\boldsymbol{x}')$ that we need to interpolate. The formula $C_A \wedge C_B$ is unsatisfiable and we can view C_A as containing one piece of the transition relation T. The transition piece itself is in convenient conjunctive form expressed with linear inequalities and we therefore pick the polyhedra domain as our precise concrete domain $\mathcal{C}^{\#}$. For the abstract domain we can choose any other arithmetic domain $\mathcal{D}^{\#}$ mentioned above. We will be using the following operations provided by the domains:

- Abstraction function $\alpha : \mathcal{C}^{\#} \mapsto \mathcal{D}^{\#}$, mapping concrete domain elements to their abstract representation.
- Concretization function $\gamma : \mathcal{D}^{\#} \mapsto \mathcal{C}^{\#}$, mapping abstract domain elements to their concrete representation.
- Join operator $\sqcup^{\#} : \mathcal{D}^{\#} \times \mathcal{D}^{\#} \mapsto \mathcal{D}^{\#}$ that, given two abstract domain elements, computes the abstract element capturing both of them.
- Projection operator $\exists^{\#}$ that can eliminate variables from elements of $\mathcal{D}^{\#}$.

Example 6. We illustrate the approach on Example 5 where P_{CR} was not satisfactory. Let $\mathcal{D}^{\#}$ be the polyhedra domain (so no abstraction is necessary). First, we project the C_A formula to its state representation and next state projections, obtaining

$$D_A = (\exists^{\#} x', y' \,.\, A) = (x = 0) \wedge (y = 0) \ ,$$
$$D_B = (\exists^{\#} x, y \,.\, A) = (x' = 1) \wedge (y' = 1) \ .$$

As usual in abstract interpretation, we can combine the two domain elements using a join operation to obtain

$$D_J = (D_A\{x, y/x', y'\}) \sqcup^{\#} D_B = (x' = y') \wedge (0 \leq x) \wedge (x \leq 1) \ ,$$

which is the invariant we were looking for.[12] □

As the example above shows, we can use the tools from abstract interpretation to infer new facts that take into account the transition system semantics (at least partially). In general, the facts inferred by abstract interpretation will not constitute an interpolant (they might not be inconsistent with C_B). But, the inferred facts are valid consequences of C_A, so we can freely conjoin them to C_A and resort to P_{CR} to complete the interpolant. The $\mathcal{T}_{\text{LA}}$-lemma interpolator P_{AI}, based on this approach, is fully described in Algorithm 2. We emphasize again that this approach relies on the model checker to provide the information about the transition system – the substitution $D_A\{\boldsymbol{x}/\boldsymbol{x}'\}$ at line 5 can not be done without knowing the correspondence between the $\boldsymbol{x}$ and $\boldsymbol{x}'$ variables.

Algorithm 2 Interpolation with Abstract Interpreation.

Require: Sets of inequalities $C_A(\boldsymbol{x}, \boldsymbol{x}')$ and $C_B(\boldsymbol{x}')$, known to be inconsistent.

1 **function** $P_{\text{AI}}(C_A, C_B)$
2 $\quad D \leftarrow \alpha(C_A)$ ▷ abstract the partial transition
3 $\quad D_A \leftarrow \exists^{\#}\boldsymbol{x}'.\ D$ ▷ project on state variables $\boldsymbol{x}$
4 $\quad D_B \leftarrow \exists^{\#}\boldsymbol{x}\ .\ D$ ▷ project on next-state variables $\boldsymbol{x}'$
5 $\quad D_J \leftarrow (D_A\{\boldsymbol{x}/\boldsymbol{x}'\}) \sqcup^{\#} D_B$ ▷ join the two abstractions
6 $\quad$ **return** $P_{\text{CR}}(C_A \cup \gamma(D_J), C_B)$ ▷ compute the interpolant

Properties of P_{AI}. The argument to show that P_{AI} is finite covering is similar to the argument we used for P_{CR}. For a fixed A, and any $\mathcal{T}_{\text{LA}}$-lemma interpolation problem (C_A, C_B), we know that $\text{LITERALS}(C_A) \subseteq \text{LITERALS}(A)$. Therefore, overall, abstract interpretation will always operate on subsets of $\text{LITERALS}(A)$ and can only ever infer a bounded number of new facts. This finite set of potential new literals does not interfere with finite covering by adding them to P_{CR}, it only increases the basis which quantifier elimination can derive inequalities from.

[12] For readers unfamiliar with the polyhedra domain, the join D_J is computed as the convex closure of the points $\{(0, 0), (1, 1)\}$.

Lemma 5. *P_{AI} is a finite-covering $\mathcal{T}_{\text{LA}}$-lemma interpolator.*

Lemma 6. *ITP$[\![P_{\text{AI}}]\!]$ interpolation procedure has the finite convergence property.*

6 Experiments

We have implemented the new interpolation method in the SALLY model-checker by relying on the MATHSAT5 SMT solver [9] for interpolation and APRON [20] for abstract interpretation over arithmetic domains.[13] We use the default PDKIND implementation in SALLY and denote with PDKIND+CR the method that uses the new interpolation method with conflict resolution (but no abstract interpretation), and with PDKIND+CR+POLKA the method that uses the new interpolation method with both conflict resolution and abstract interpretation based on the polyhedra domain.

We have evaluated the new procedure on a range of benchmarks. Several of our benchmarks are related to fault-tolerant algorithms (`om`, `ttesynchro` and `ttastartup`, `unifapprox`, `azadmanesh`, `approxagree`, `hacms`, and `misc` problem sets). We also used benchmarks from software model checking (`cav12`, `ctigar`). The `lustre` benchmarks are from the benchmark suite of the KIND model-checker, `cons` are simple concurrent programs, and `lfht` problems model a lock-free hash table. Some of the benchmarks were obtained from an existing repository.[14]

Our main goal is to illustrate the impact of the new interpolation method but, to put the results in context, we also compare NUXMV [5,8] (NUXMV was the most robust model checker in our previous work [22]) The results are presented in Figure 1. Each problem instance was run with a timeout of 10 minutes. Each column of the table corresponds to PDKIND with a different $\mathcal{T}_{\text{LA}}$-lemma interpolator, and each row corresponds to a different problem set. For each problem set and interpolator we report the number of problems that the tool has solved, how many of the solved problems were valid and invalid properties, and the total time (in seconds) that the tool took to solve those problems.

First we evaluate the impact of using conflict resolution (PDKIND+CR) and abstract interpretation (PDKIND+CR+POLKA) as the interpolator, compared to the default PDKIND with the Farkas-based interpolator. The results are shown in Figure 1. Overall, by adding conflict resolution as the lemma interpolator (PDKIND+CR), the method can find more counter-examples (but can prove fewer properties) than the default PDKIND. Then, by extending it with abstract interpretation (PDKIND+CR+POLKA), the tool retains the advantage at finding counter examples, but can, in addition, prove more valid properties. These results are aligned with our expectations. With the interpolation providing convergence guarantees, the PDKIND method does not get stuck in individual invariant

[13] SALLY is open source on GitHub. The majority of the interpolator code can be seen in `https://github.com/SRI-CSL/sally/blob/interpolation/src/smt/mathsat5/conflict_resolution.cpp`.

[14] `https://es-static.fbk.eu/people/griggio/vtsa2015/`

Fig. 1. Comparison of different $\mathcal{T}_{\text{LA}}$-lemma interpolators. Rows correspond to different problem sets and columns correspond to PDKIND with different interpolators (separate column for NUXMV for context). Each table entry shows the number of problems that the tool has solved, how many of those were valid and invalid, and the total time it took for the solved instances.

	PDKIND			PDKIND+CR			PDKIND+CR+POLKA			NUXMV		
problem set	solved	valid/invalid	time (s)	solved	valid/invalid	time (s)	solved	valid/invalid	time (s)	solved	valid/invalid	time (s)
approxagree (9)	**9**	**8/1**	**185**	9	8/1	240	9	8/1	238	6	5/1	477
azadmanesh (20)	20	17/3	278	**20**	**17/3**	**173**	20	17/3	174	20	17/3	1269
cav12 (99)	68	48/20	2541	71	49/22	3722	**71**	**49/22**	**2966**	74	51/23	2910
conc (6)	4	4/0	117	3	3/0	6	**5**	**5/0**	**30**	4	4/0	220
ctigar (110)	**74**	**54/20**	**1252**	73	53/20	1532	74	54/20	1686	81	61/20	1829
hacms (5)	4	2/2	955	3	2/1	463	**5**	**3/2**	**923**	4	2/2	459
lfht (27)	17	17/0	106	17	17/0	681	**23**	**23/0**	**2194**	24	24/0	562
lustre (790)	772	438/334	2730	755	419/336	5209	**773**	**438/335**	**1964**	769	434/335	3542
misc (10)	8	7/1	117	8	6/2	200	**9**	**7/2**	**241**	8	8/0	320
om (9)	9	7/2	3	**9**	**7/2**	**1**	**9**	**7/2**	**1**	9	7/2	469
ttastartup (3)	1	1/0	7	1	1/0	7	**1**	**1/0**	**6**	1	1/0	1
ttesynchro (6)	6	3/3	16	**6**	**3/3**	**9**	**6**	**3/3**	**9**	5	2/3	1428
unifapprox (11)	11	8/3	225	11	8/3	134	**11**	**8/3**	**132**	11	8/3	271
	1003	614/389	8532	986	593/393	12377	**1016**	**623/393**	**10564**	1016	624/392	13757

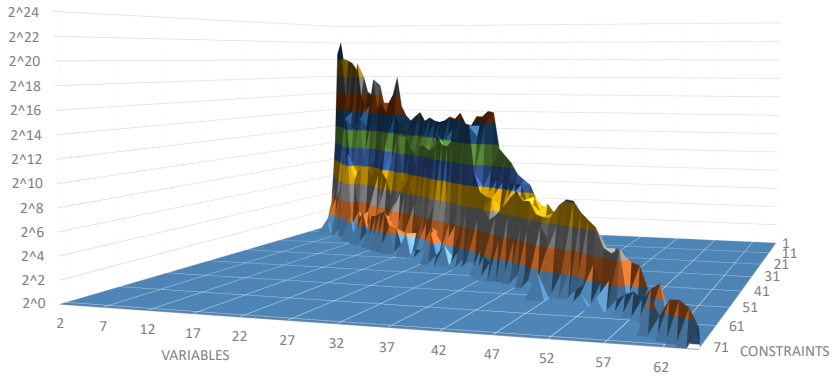
inference frames and can make progress towards the counter-examples. But, due to its restricted interpolation language it is bound to also be restricted in invariant inference, which is why PDKIND+CR can show fewer valid properties correct. The addition of abstract interpretation inferences improves this situation by extending the expressiveness of interpolants and making the interpolants more invariant-directed.

Next, we evaluate the effect of using different abstract domains. The APRON library provides the standard interval [11], octagon [31], and polyhedra [13] domains, and we denote variants of PDKIND that use these domains as PDKIND+CR+BOX, PDKIND+CR+OCT, and PDKIND+CR+POLKA. Results of the comparison are presented in Figure 2. The main takeaway from comparing different abstract domains is that, the more expressive the domain, the better the results. In general, expressive abstract domains are desirable in verification application, but suffer from scalability problems. The polyhedra domain, for example, is worst case exponential complexity in both space and time, and there is ongoing work to try and make it more efficient in practice [36]. In our context, we can easily apply the polyhedra domain even by relying on an off-the-shelf library such as APRON, as we use the domains solely on the cores of the theory lemmas produced by the SMT solver, where the number of variables and lemmas tends to be small. Figure 3 shows the distribution of the interpolation problems with respect to the number of constrains and the number of variables.

Fig. 2. Comparison of different abstract domains. Each row corresponds to a different problem set. Each column corresponds to PDKIND+CR with a different abstract domain. Each table entry shows the number of problems that the engine solved, how many of those were valid and invalid, and the total time it took for the solved instances.

	PDKIND+CR			PDKIND+CR+BOX			PDKIND+CR+OCT			PDKIND+CR+POLKA		
problem set	solved	valid/invalid	time (s)	solved	valid/invalid	time (s)	solved	valid/invalid	time (s)	solved	valid/invalid	time (s)
approxagree (9)	9	8/1	240	9	8/1	240	**9**	**8/1**	**237**	9	8/1	238
azadmanesh (20)	20	17/3	173	**20**	**17/3**	**170**	**20**	**17/3**	**170**	20	17/3	174
cav12 (99)	71	49/22	3722	71	49/22	3291	67	48/19	1474	**71**	**49/22**	**2966**
conc (6)	3	3/0	6	5	5/0	35	5	5/0	47	**5**	**5/0**	**30**
ctigar (110)	73	53/20	1532	**74**	**54/20**	**1395**	73	53/20	884	74	54/20	1686
hacms (5)	3	2/1	463	4	2/2	799	4	3/1	1036	**5**	**3/2**	**923**
lfht (27)	17	17/0	681	20	20/0	1020	20	20/0	772	**23**	**23/0**	**2194**
lustre (790)	755	419/336	5209	757	421/336	3075	762	428/334	3063	**773**	**438/335**	**1964**
misc (10)	8	6/2	200	9	7/2	303	**9**	**7/2**	**220**	9	7/2	241
om (9)	**9**	**7/2**	**1**	**9**	**7/2**	**1**	**9**	**7/2**	**1**	**9**	**7/2**	**1**
ttastartup (3)	1	1/0	7	1	1/0	6	**2**	**1/1**	**459**	1	1/0	6
ttesynchro (6)	**6**	**3/3**	**9**	6	3/3	10	**6**	**3/3**	**9**	**6**	**3/3**	**9**
unifapprox (11)	11	8/3	134	11	8/3	131	**11**	**8/3**	**130**	11	8/3	132
	986	593/393	12377	996	602/394	10476	997	608/389	8502	**1016**	**623/393**	**10564**

Fig. 3. The distribution of the number of constraints and variables in $\mathcal{T}_{\text{LA}}$-lemma interpolation problems over our whole dataset (vertical axis is logarithmic). Of all interpolation problems, 66.55% have 5 variables or less, 87.18% have 10 variables or less, and 96.22% have 20 variables or less.



7 Conclusion

We presented a new approach for proof-based interpolation in SMT that can guarantee convergence of interpolation sequences and is invariant-driven. Both of these properties are valuable in the context of model checking techniques such as IC3/PDR. For example, with the new interpolation method, we can finally guarantee that the theoretical results for our own PDKIND method [22] also hold in practice. The approach combines two orthogonal approaches to invariant inference – symbolic reasoning through quantifier elimination and semantic reasoning with abstract interpretation – to provide interpolants that are both expressive invariant-driven facts and can be controlled to provide convergence guarantees. The new interpolation method takes advantage of the strengths of the individual parts of the usual model-checking reasoning stack. For a system under analysis, the model checker provides information about the system, the SMT solver discharges the control-flow of the system, the interpolator provides the symbolic forward reasoning, and the abstract interpretation improves the interpolants by interpreting the pieces of the system with no control flow. For example, both quantifier elimination and abstract interpretation can be expensive or ineffective on expressive domains when the problems involve many variables and disjunctions. Our new method sidesteps these issues since we only need to reason on the unsatisfiable cores provided by the SMT solver, which are minimal and conjunctive. The overall architecture of the new approach is shown in Figure 4.

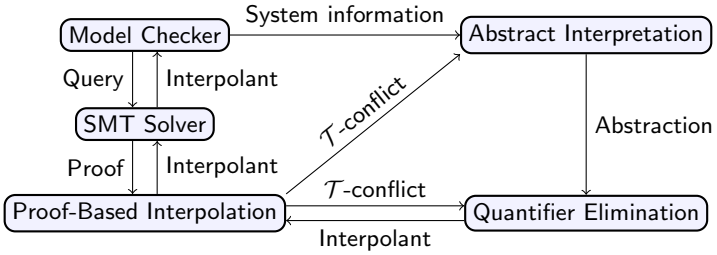


Fig. 4. All participants in the interpolation framework.

The method is implemented in the SALLY model checker, by relying on the proof-based interpolation framework of the MATHSAT5 SMT solver and the APRON abstract domain library. Our experimental evaluation shows that the new interpolation method is effective and improves the performance of SALLY in both invariant inference and bug-finding.

Future Work. The new interpolation method is modular and enables cross-pollination of the different techniques across the whole reasoning stack. This gives rise to many interesting directions for future work. As the first steps, we

plan to explore the use of tools from abstract interpretation to improve the interpolation in the theories of integer arithmetic, bit-vectors and arrays. In the other direction, we also see a possibility to contribute symbolic techniques to abstract interpretation. For example, if we lift the restriction that the lemma interpolants must refute the B part of the interpolation problem, the result of the proof-based interpolant computation is not an interpolant but rather an abstraction of the A formula. This could be a potential direction toward a property-driven logical interpretation (e.g., [37,3]). It is important to note that, although our new method guarantees convergence of the interplant sequences, it does not guarantee the convergence of the overall model-checking procedure. The overall convergence can be achieved by adding even more control over the interpolation language (e.g., [21]), and we plan to explore this direction.

Acknowledgements. We would like to thank Alberto Griggio for providing an interface to MATHSAT5 that allowed us to replace the default $\mathcal{T}$-interpolator with our custom external interpolator.

References

1. A. Albarghouthi and K. L. McMillan. Beautiful interpolants. In *International Conference on Computer Aided Verification*, pages 313–329. Springer, 2013.
2. C. W. Barrett, R. Sebastiani, S. A. Seshia, and C. Tinelli. Satisfiability modulo theories. *Handbook of satisfiability*, 185:825–885, 2009.
3. N. Bjørner and A. Gurfinkel. Property directed polyhedral abstraction. In *International Workshop on Verification, Model Checking, and Abstract Interpretation*, pages 263–281. Springer, 2015.
4. N. Bjørner, K. McMillan, and A. Rybalchenko. On solving universally quantified horn clauses. In *International Static Analysis Symposium*, pages 105–125. Springer, 2013.
5. R. Cavada, A. Cimatti, M. Dorigatti, A. Griggio, A. Mariotti, A. Micheli, S. Mover, M. Roveri, and S. Tonetta. The nuXmv symbolic model checker. In *International Conference on Computer Aided Verification*, pages 334–342. Springer, 2014.
6. J. Christ, J. Hoenicke, and A. Nutz. Proof tree preserving interpolation. In *TACAS*, volume 13, pages 124–138. Springer, 2013.
7. A. Cimatti and A. Griggio. Software model checking via IC3. In *CAV*, volume 7358, pages 277–293. Springer, 2012.
8. A. Cimatti, A. Griggio, S. Mover, and S. Tonetta. IC3 modulo theories via implicit predicate abstraction. In *Tacas*, volume 8413, pages 46–61, 2014.
9. A. Cimatti, A. Griggio, B. J. Schaafsma, and R. Sebastiani. The MathSAT5 SMT solver. In *TACAS*, volume 7795, pages 93–107. Springer, 2013.
10. A. Cimatti, A. Griggio, and R. Sebastiani. Efficient generation of Craig interpolants in satisfiability modulo theories. *ACM Transactions on Computational Logic (TOCL)*, 12(1):7, 2010.
11. P. Cousot and R. Cousot. Static determination of dynamic properties of programs. In *Proceedings of the 2nd International Symposium on Programming, Paris, France*. Dunod, 1976.

12. P. Cousot and R. Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Proceedings of the 4th ACM SIGACT-SIGPLAN symposium on Principles of programming languages*, pages 238–252. ACM, 1977.
13. P. Cousot and N. Halbwachs. Automatic discovery of linear restraints among variables of a program. In *Proceedings of the 5th ACM SIGACT-SIGPLAN symposium on Principles of programming languages*, pages 84–96. ACM, 1978.
14. W. Craig. Three uses of the herbrand-gentzen theorem in relating model theory and proof theory. *The Journal of Symbolic Logic*, 22(3):269–285, 1957.
15. L. De Moura and D. Jovanović. A model-constructing satisfiability calculus. In *International Workshop on Verification, Model Checking, and Abstract Interpretation*, pages 1–12. Springer, 2013.
16. V. D'Silva, D. Kroening, M. Purandare, and G. Weissenbacher. Interpolant strength. In *VMCAI*, volume 10, pages 129–145. Springer, 2010.
17. B. Dutertre and L. De Moura. A fast linear-arithmetic solver for dpll (t). In *Computer Aided Verification*, pages 81–94. Springer, 2006.
18. K. Hoder, L. Kovacs, and A. Voronkov. Playing in the grey area of proofs. In *ACM SIGPLAN Notices*, volume 47, pages 259–272. ACM, 2012.
19. G. Huang. Constructing Craig interpolation formulas. *Computing and Combinatorics*, pages 181–190, 1995.
20. B. Jeannet and A. Miné. Apron: A library of numerical abstract domains for static analysis. In *Computer Aided Verification*, pages 661–667. Springer, 2009.
21. R. Jhala and K. L. McMillan. A practical and complete approach to predicate refinement. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 459–473. Springer, 2006.
22. D. Jovanović and B. Dutertre. Property-directed k-induction. In *Proceedings of the 16th Conference on Formal Methods in Computer-Aided Design*, pages 85–92. FMCAD Inc, 2016.
23. A. Komuravelli, A. Gurfinkel, and S. Chaki. SMT-based model checking for recursive programs. *Formal Methods in System Design*, 48(3):175–205, 2016.
24. K. Korovin, N. Tsiskaridze, and A. Voronkov. Conflict resolution. *Principles and Practice of Constraint Programming-CP 2009*, pages 509–523, 2009.
25. J. Krajíček. Interpolation theorems, lower bounds for proof systems, and independence results for bounded arithmetic. *The Journal of Symbolic Logic*, 62(2):457–486, 1997.
26. K. McMillan, A. Kuehlmann, and M. Sagiv. Generalizing dpll to richer logics. In *Computer Aided Verification*, pages 462–476. Springer, 2009.
27. K. L. McMillan. Interpolation and SAT-based model checking. In *International Conference on Computer Aided Verification*, pages 1–13. Springer, 2003.
28. K. L. McMillan. An interpolating theorem prover. *Theoretical Computer Science*, 345(1):101–121, 2005.
29. K. L. McMillan. Lazy abstraction with interpolants. In *International Conference on Computer Aided Verification*, pages 123–136. Springer, 2006.
30. K. L. McMillan. Interpolation: Proofs in the service of model checking. In *Hanbook of Model-Checking*. Springer, 2014.
31. A. Miné. The octagon abstract domain. *Higher-order and symbolic computation*, 19(1):31–100, 2006.
32. R. Nieuwenhuis, A. Oliveras, and C. Tinelli. Solving sat and sat modulo theories: From an abstract davis–putnam–logemann–loveland procedure to dpll (t). *Journal of the ACM (JACM)*, 53(6):937–977, 2006.

33. P. Pudlák. Lower bounds for resolution and cutting plane proofs and monotone computations. *The Journal of Symbolic Logic*, 62(3):981–998, 1997.
34. S. Rollini, O. Sery, and N. Sharygina. Leveraging interpolant strength in model checking. In *Computer Aided Verification*, pages 193–209. Springer, 2012.
35. A. Schrijver. *Theory of linear and integer programming*. John Wiley & Sons, 1998.
36. G. Singh, M. Püschel, and M. T. Vechev. Fast polyhedra abstract domain. In *POPL*, pages 46–59, 2017.
37. A. Tiwari and S. Gulwani. Logical interpretation: Static program analysis using theorem proving. In *CADE*, volume 4603, pages 147–166. Springer, 2007.
38. G. Weissenbacher. Interpolant strength revisited. In *International Conference on Theory and Applications of Satisfiability Testing*, pages 312–326. Springer, 2012.