Efficient Interpolant Generation in Satisfiability Modulo Theories *

Alessandro Cimatti[1], Alberto Griggio[2], and Roberto Sebastiani[2]

[1] FBK-IRST, Povo, Trento, Italy. cimatti@fbk.eu
[2] DISI, Università di Trento, Italy. {griggio,rseba}@disi.unitn.it

Abstract. The problem of computing Craig Interpolants for propositional (SAT) formulas has recently received a lot of interest, mainly for its applications in formal verification. However, propositional logic is often not expressive enough for representing many interesting verification problems, which can be more naturally addressed in the framework of Satisfiability Modulo Theories, SMT.
Although some works have addressed the topic of generating interpolants in SMT, the techniques and tools that are currently available have some limitations, and their performace still does not exploit the full power of current state-of-the-art SMT solvers.
In this paper we try to close this gap. We present several techniques for interpolant generation in SMT which overcome the limitations of the current generators mentioned above, and which take full advantage of state-of-the-art SMT technology. These novel techniques can lead to substantial performance improvements wrt. the currently available tools.
We support our claims with an extensive experimental evaluation of our implementation of the proposed techniques in the MathSAT SMT solver.

1 Introduction

Since the seminal paper of McMillan [19], interpolation has been recognized to be a substantial tool for verification in the case of boolean systems [7, 17, 18]. The tremendous improvements of Satisfiability Modulo Theory (SMT) solvers in the recent years have enabled the lifting of SAT-based verification algorithms to the non-boolean case [2, 1], and made it practical the implementation of other approaches such as CEGAR [21].

However, the research on interpolation for SMT has not kept the pace of the SMT solvers. In fact, the current approaches to producing interpolants for SMT [20, 30, 27, 16, 15] all suffer from a number of limitations. Some of the approaches are severely limited in terms of their expressiveness. For instance, the tool described in [27] can only deal with conjunctions of literals, whilst the recent work described in [16] can not deal with many useful theories. Furthermore, very few tools are available [27, 20], and these tools do not seem to scale particularly well. More than to naïve implementation, this appears to be due to the underlying algorithms, that substantially deviate from or ignore choices common in state-of-the-art SMT. For instance, in the domain

* This work has been partly supported by ORCHID, a project sponsored by Provincia Autonoma di Trento, and by a grant from Intel Corporation.

of linear arithmetic over the rationals ($\mathcal{LA}(\mathbb{Q})$), strict inequalities are encoded in [20] as the conjunction of a weak inequality and a disequality; although sound, this choice destroys the structure of the constraints, requires additional splitting, and ultimately results in a larger search space. Similarly, the fragment of Difference Logic ($\mathcal{DL}(\mathbb{Q})$) is dealt with by means of a general-purpose algorithm for full $\mathcal{LA}(\mathbb{Q})$, rather than one of the well-known and much faster specialized algorithms. An even more fundamental example is the fact that state-of-the-art SMT reasoners use dedicated algorithms for Linear Arithmetic [10].

In this paper, we tackle the problem of generating interpolants within a state of the art SMT solver. We present a fully general approach that can generate interpolants for the most effective algorithms in SMT, most notably the algorithm for deciding $\mathcal{LA}(\mathbb{Q})$ presented in [10] and those for $\mathcal{DL}(\mathbb{Q})$ in [9, 23]. Our approach is also applicable to the combination of theories, based on the Delayed Theory Combination (DTC) method [5, 6], as an alternative to the traditional Nelson-Oppen method.

We carried out an extensive experimental evaluation on a wide range of benchmarks. The proposed techniques substantially advance the state of the art: our interpolator can deal with problems that can not be expressed in other solvers; furthermore, a comparison on problems that can be dealt with by other tools shows dramatic improvements in performance, often by orders of magnitude.

The paper is structured as follows. In §2 we present some background on interpolation in SMT. In §3 and §4 we show how to efficiently interpolate $\mathcal{LA}(\mathbb{Q})$ and the subcase of $\mathcal{DL}(\mathbb{Q})$. In §5 we discuss interpolation for combined theories. In §6 we analyze the experimental evaluation, whilst in §7 we draw some conclusions. For lack of space, we omit the proofs of the theorems. They can be found in the extended technical report [8].

2 Background

2.1 Satisfiability Modulo Theory – SMT

Our setting is standard first order logic. A 0-ary function symbol is called a *constant*. A *term* is a first-order term built out of function symbols and variables. A *linear term* is either a linear combination $c_1x_1 + \ldots + c_nx_n + c$, where c and c_i are numeric constants and x_i are variables. When doing arithmetic on terms, simplifications are performed where needed. We write $t_1 \equiv t_2$ when the two terms t_1 and t_2 are syntactically identical. If $t_1, \ldots, t_n$ are terms and p is a predicate symbol, then $p(t_1, \ldots, t_n)$ is an *atom*. A *literal* is either an atom or its negation. A *(quantifier-free) formula* ϕ is an arbitrary boolean combination of atoms. We use the standard notions of theory, satisfiability, validity, logical consequence. We consider only theories with equality. We call *Satisfiability Modulo (the) Theory* $\mathcal{T}$, SMT($\mathcal{T}$), the problem of deciding the satisfiability of quantifier-free formulas wrt. a background theory $\mathcal{T}$. [3]

We denote formulas with ϕ, ψ, A, B, C, I, variables with x, y, z, and numeric constants with a, b, c, l, u. Given a theory $\mathcal{T}$, we write $\phi \models_{\mathcal{T}} \psi$ (or simply $\phi \models \psi$) to denote

[3] The general definition of SMT deals also with quantified formulas. Nevertheless, in this paper we restrict our interest to quantifier-free formulas.

that the formula ψ is a logical consequence of ϕ in the theory $\mathcal{T}$. With $\phi \preceq \psi$ we denote that all uninterpreted (in $\mathcal{T}$) symbols of ϕ appear in ψ. Without loss of generality, we also assume that the formulas are in Conjunctive Normal Form (CNF). If C is a clause, $C \downarrow B$ is the clause obtained by removing all the literals whose atoms do not occur in B, and $C \setminus B$ that obtained by removing all the literals whose atoms do occur in B. With a little abuse of notation, we might sometimes denote conjunctions of literals $l_1 \wedge \ldots \wedge l_n$ as sets $\{l_1, \ldots, l_n\}$ and vice versa. If $\eta \equiv \{l_1, \ldots, l_n\}$, we might write $\neg\eta$ to mean $\neg l_1 \vee \ldots \vee \neg l_n$.

We call $\mathcal{T}$-solver a procedure that decides the consistency of a conjunction of literals in $\mathcal{T}$. If $S \equiv \{l_1, \ldots, l_n\}$ is a set of literals in $\mathcal{T}$, we call *($\mathcal{T}$)-conflict set* any subset η of S which is inconsistent in $\mathcal{T}$. [4] We call $\neg\eta$ a $\mathcal{T}$-lemma (notice that $\neg\eta$ is a $\mathcal{T}$-valid clause). Given a set of clauses $S \equiv \{C_1, \ldots, C_n\}$ and a clause C, we call a *resolution proof* that $\bigwedge_i C_i \models_{\mathcal{T}} C$ a DAG $\mathcal{P}$ such that:

1. C is the root of $\mathcal{P}$;
2. the leaves of $\mathcal{P}$ are either elements of S or $\mathcal{T}$-lemmas;
3. each non-leaf node C' has two parents C_{p_1} and C_{p_2} such that $C_{p_1} \equiv p \vee \phi_1$, $C_{p_2} \equiv \neg p \vee \phi_2$, and $C' \equiv \phi_1 \vee \phi_2$. The atom p is called the *pivot* of C_{p_1} and C_{p_2}.

If C is the empty clause (denoted with $\bot$), then $\mathcal{P}$ is a *resolution proof of unsatisfiability* for $\bigwedge_i C_i$.

A standard technique for solving the SMT($\mathcal{T}$) problem is to integrate a DPLL-based SAT solver and a $\mathcal{T}$-solver in a *lazy* manner (see, e.g., [28] for a detailed description). DPLL is used as an enumerator of truth assignments for the propositional abstraction of the input formula. At each step, the set of $\mathcal{T}$-literals S corresponding to the current assignment is sent to the $\mathcal{T}$-solver to be checked for consistency in $\mathcal{T}$. If S is inconsistent, the $\mathcal{T}$-solver returns a conflict set η, and the corresponding $\mathcal{T}$-lemma $\neg\eta$ is added as a blocking clause in DPLL, and used to drive the backjump mechanism. With a small modification of the embedded DPLL engine, a lazy SMT solver can also be used to generate a resolution proof of unsatisfiability.

2.2 Interpolation in SMT

We consider the SMT($\mathcal{T}$) problem for some background theory $\mathcal{T}$. Given an ordered pair (A, B) of formulas such that $A \wedge B \models_{\mathcal{T}} \bot$, a *Craig interpolant* (simply "interpolant" hereafter) is a formula I s.t.:

a) $A \models_{\mathcal{T}} I$,
b) $I \wedge B \models_{\mathcal{T}} \bot$,
c) $I \preceq A$ and $I \preceq B$.

The use of interpolation in formal verification has been introduced by McMillan in [19] for purely-propositional formulas, and it was subsequently extended to handle SMT($\mathcal{EUF} \cup \mathcal{LA}(\mathbb{Q})$) formulas in [20], $\mathcal{EUF}$ being the theory of equality and uninterpreted functions. The technique is based on earlier work by Pudlák [25], where

[4] In the next sections, as we are in an SMT($\mathcal{T}$) context, we often omit specifying "in the theory $\mathcal{T}$" when speaking of consistency, validity, etc.

two interpolant-generation algorithms are described: one for computing interpolants for propositional formulas from resolution proofs of unsatisfiability, and one for generating interpolants for conjunctions of (weak) linear inequalities in $\mathcal{LA}(\mathbb{Q})$. An interpolant for (A, B) is constructed from a resolution proof of unsatisfiability of $A \wedge B$, generated as outlined in §2.1. The algorithm can be described as follows:

Algorithm 1: Interpolant generation for SMT($\mathcal{T}$)

1. Generate a proof of unsatisfiability $\mathcal{P}$ for $A \wedge B$.
2. For every $\mathcal{T}$-lemma $\neg\eta$ occurring in $\mathcal{P}$, generate an interpolant $I_{\neg\eta}$ for $(\eta \setminus B, \eta \downarrow B)$.
3. For every input clause C in $\mathcal{P}$, set $I_C \equiv C \downarrow B$ if $C \in A$, and $I_C \equiv \top$ if $C \in B$.
4. For every inner node C of $\mathcal{P}$ obtained by resolution from $C_1 \equiv p \vee \phi_1$ and $C_2 \equiv \neg p \vee \phi_2$, set $I_C \equiv I_{C_1} \vee I_{C_2}$ if p does not occur in B, and $I_C \equiv I_{C_1} \wedge I_{C_2}$ otherwise.
5. Output $I_\perp$ as an interpolant for (A, B).

Notice that Step 2. of the algorithm is the only part which depends on the theory $\mathcal{T}$, so that the problem of interpolant generation in SMT($\mathcal{T}$) reduces to that of finding interpolants for $\mathcal{T}$-lemmas. To this extent, in [20] McMillan gives a set of rules for constructing interpolants for $\mathcal{T}$-lemmas in the theory of $\mathcal{EUF}$, that of weak linear inequalities $(0 \leq t)$ in $\mathcal{LA}(\mathbb{Q})$, and their combination. Linear equalities $(0 = t)$ can be reduced to conjunctions $(0 \leq t) \wedge (0 \leq -t)$ of inequalities. Thanks to the combination of theories, also strict linear inequalities $(0 < t)$ can be handled in $\mathcal{EUF} \cup \mathcal{LA}(\mathbb{Q})$ by replacing them with the conjunction $(0 \leq t) \wedge (0 \neq t)$,[5] but this solution can be very inefficient. The combination $\mathcal{EUF} \cup \mathcal{LA}(\mathbb{Q})$ can also be used to compute interpolants for other theories, such as those of lists, arrays, sets and multisets [15].

In [20], interpolants in the combined theory $\mathcal{EUF} \cup \mathcal{LA}(\mathbb{Q})$ are obtained by means of ad-hoc combination rules. The work in [30], instead, presents a method for generating interpolants for $\mathcal{T}_1 \cup \mathcal{T}_2$ using the interpolant-generation procedures of $\mathcal{T}_1$ and $\mathcal{T}_2$ as black-boxes, using the Nelson-Oppen approach [22].

Also the method of [27] allows to compute interpolants in $\mathcal{EUF} \cup \mathcal{LA}(\mathbb{Q})$. Its peculiarity is that it is not based on unsatisfiability proofs. Instead, it generates interpolants in $\mathcal{LA}(\mathbb{Q})$ by solving a system of constraints using an off-the-shelf Linear Programming (LP) solver. The method allows both weak and strict inequalities. Extension to uninterpreted functions is achieved by means of reduction to $\mathcal{LA}(\mathbb{Q})$ using a hierarchical calculus. The algorithm works only with conjunctions of atoms, although in principle it could be integrated in Algorithm 1 to generate interpolants for $\mathcal{T}$-lemmas in $\mathcal{LA}(\mathbb{Q})$. As an alternative, the authors show in [27] how to generate interpolants for formulas that are in Disjunctive Normal Form (DNF).

Another different approach is explored in [16]. There, the authors use the *eager* SMT approach to encode the original SMT problem into an equisatisfiable propositional problem, for which a propositional proof of unsatisfiability is generated. This proof is later "lifted" to the original theory, and used to generate an interpolant in a way similar to Algorithm 1. At the moment, the approach is however limited to the theory of equality only (without uninterpreted functions).

[5] The details are not given in [20]. One possible way of doing this is to rewrite $(0 \neq t)$ as $(y = t) \wedge (z = 0) \wedge (z \neq y)$, z and y being fresh variables.

$$\textsc{Hyp}\ \frac{}{\Gamma \vdash \phi}\ \phi \in \Gamma \quad \textsc{LeqEq}\ \frac{\Gamma \vdash 0 = t}{\Gamma \vdash 0 \leq t} \quad \textsc{Comb}\ \frac{\Gamma \vdash 0 \leq t_1 \qquad \Gamma \vdash 0 \leq t_2}{\Gamma \vdash 0 \leq c_1 t_1 + c_2 t_2}\ c_1, c_2 > 0$$

Fig. 1. Proof rules for $\mathcal{LA}(\mathbb{Q})$ (without strict inequalities).

All the above techniques construct *one* interpolant for (A, B). In general, however, interpolants are not unique. In particular, some of them can be better than others, depending on the particular application domain. In [12], it is shown how to manipulate proofs in order to obtain stronger interpolants. In [13, 14], instead, a technique to restrict the language used in interpolants is presented and shown to be useful in preventing divergence of techniques based on predicate abstraction.

3 Interpolation for Linear Arithmetic with a state-of-the-art solver

Traditionally, SMT solvers used some kind of incremental simplex algorithm [29] as $\mathcal{T}$-solver for the $\mathcal{LA}(\mathbb{Q})$ theory. Recently, Dutertre and de Moura [10] have proposed a new simplex-based algorithm, specifically designed for integration in a lazy SMT solver. The algorithm is extremely efficient and was shown to significantly outperform (often by orders of magnitude) the traditional ones. It has now been integrated in several SMT solvers, including ARGOLIB, CVC3, MATHSAT, YICES, and Z3. Remarkably, this algorithm allows for handling also strict inequalities.

In this Section, we show how to exploit this algorithm to efficiently generate interpolants for $\mathcal{LA}(\mathbb{Q})$ formulas. In §3.1 we begin by considering the case in which the input atoms are only equalities and non-strict inequalities. In this case, we only need to show how to generate a proof of unsatisfiability, since then we can use the interpolation rules defined in [20]. Then, in §3.2 we show how to generate interpolants for problems containing also strict inequalities and disequalities.

3.1 Interpolation with non-strict inequalities

Similarly to [20], we use the proof rules of Figure 1: HYP for introducing hypotheses, LEQEQ for deriving inequalities from equalities, and COMB for performing linear combinations.[6] As in [20], we consider an atom "$0 \leq c$", c being a negative numerical constant, as a synonym of $\bot$.

The original Dutertre-de Moura algorithm. In its original formulation, the Dutertre-de Moura algorithm assumes that the variables x_i are partitioned a priori in two sets, hereafter denoted as $\hat{\mathcal{B}}$ ("initially basic") and $\hat{\mathcal{N}}$ ("initially non-basic"), and that the algorithm receives as inputs two kinds of atomic formulas: [7]

[6] In [20] the LEQEQ rule is not used in $\mathcal{LA}(\mathbb{Q})$, because the input is assumed to consist only of inequalities.

[7] Notationally, we use the hat symbol $\hat{}$ to denote the initial value of the generic symbol.

– a set of *equations* eq_i, one for each $x_i \in \hat{\mathcal{B}}$, of the form $\sum_{x_j \in \hat{\mathcal{N}}} \hat{a}_{ij} x_j + \hat{a}_{ii} x_i = 0$ s.t. all $\hat{a}_{ij}$'s are numerical constants;
– *elementary atoms* of the form $x_j \geq l_j$ or $x_j \leq u_j$ s.t. l_j, u_j are numerical constants.

The initial equations eq_i are then used to build a tableau T:

$$\{x_i = \textstyle\sum_{x_j \in \mathcal{N}} a_{ij} x_j \mid x_i \in \mathcal{B}\}, \tag{1}$$

where $\mathcal{B}$ ("basic"), $\mathcal{N}$ ("non-basic") and a_{ij} are such that initially $\mathcal{B} \equiv \hat{\mathcal{B}}$, $\mathcal{N} \equiv \hat{\mathcal{N}}$ and $a_{ij} \equiv -\hat{a}_{ij}/\hat{a}_{ii}$.

In order to decide the satisfiability of the input problem, the algorithm performs manipulations of the tableau that change the sets $\mathcal{B}$ and $\mathcal{N}$ and the values of the coefficients a_{ij}, always keeping the tableau T in (1) equivalent to its initial version. An inconsistency is detected when it is not possible to satisfy all the bounds on the variables introduced by the elementary atoms: as the algorithm ensures that the bounds on the variables in $\mathcal{N}$ are always satisfied, then there is a variable $x_i \in \mathcal{B}$ such that the inconsistency is caused either by the elementary atom $x_i \geq l_i$ or by the atom $x_i \leq u_i$ [10]. In the first case, [8] a conflict set η is generated as follows:

$$\eta = \{x_j \leq u_j | x_j \in \mathcal{N}^+\} \cup \{x_j \geq l_j | x_j \in \mathcal{N}^-\} \cup \{x_i \geq l_i\}, \tag{2}$$

where $x_i = \sum_{x_j \in \mathcal{N}} a_{ij} x_j$ is the row of the current version of the tableau T (1) corresponding to x_i, $\mathcal{N}^+$ is $\{x_j \in \mathcal{N} | a_{ij} > 0\}$ and $\mathcal{N}^-$ is $\{x_j \in \mathcal{N} | a_{ij} < 0\}$.

Notice that η is a conflict set in the sense that it is made inconsistent by (some of) the equations in the tableau T (1), i.e. $T \cup \eta \models_{\mathcal{LA}(\mathbb{Q})} \bot$.

In order to handle problems that are not in the above form, a satisfiability-preserving preprocessing step is applied upfront, before invoking the algorithm.

Our variant. In our variant of the algorithm, instead, the input is an arbitrary set of inequalities $l_k \leq \sum_h \hat{a}_{kh}\, y_h$ or $u_k \geq \sum_h \hat{a}_{kh}\, y_h$, and the preprocessing step is applied internally. In particular, we introduce a "slack" variable s_k for each distinct term $\sum_h \hat{a}_{kh}\, y_h$ occurring in the input inequalities. Then, we replace such term with s_k (thus obtaining $l_k \leq s_k$ or $u_k \geq s_k$) and add an equation $s_k = \sum_h \hat{a}_{kh}\, y_h$. Notice that we introduce a slack variable even for "elementary" inequalities $(l_k \leq y_k)$. With this transformation, the initial tableau T (1) is:

$$\{s_k = \textstyle\sum_h \hat{a}_{kh}\, y_h\}_k, \tag{3}$$

s.t. $\hat{\mathcal{B}}$ is made of all the slack variables s_k's, $\hat{\mathcal{N}}$ is made of all the original variables y_h's, and the elementary atoms contain only slack variables s_k's.

In our variant, we can use η to generate a conflict set η', thanks to the following lemma.

[8] Here we do not consider the second case $x_i \leq u_i$ as it is analogous to the first one.

Lemma 1. *In the set η of* (2)*, x_i and all the x_j's are slack variables introduced by our preprocessing step. Moreover, the set $\eta' \equiv \eta_{\mathcal{N}^+} \cup \eta_{\mathcal{N}^-} \cup \eta_i$ is a conflict set, where*

$$\begin{aligned}
\eta_{\mathcal{N}^+} &\equiv \{u_k \geq \textstyle\sum_h \hat{a}_{kh}\, y_h | s_k \equiv x_j \text{ and } x_j \in \mathcal{N}^+\},\\
\eta_{\mathcal{N}^-} &\equiv \{l_k \leq \textstyle\sum_h \hat{a}_{kh}\, y_h | s_k \equiv x_j \text{ and } x_j \in \mathcal{N}^-\},\\
\eta_i &\equiv \{l_k \leq \textstyle\sum_h \hat{a}_{kh}\, y_h | s_k \equiv x_i\}.
\end{aligned}$$

We construct a proof of inconsistency as follows. From the set η of (2) we build a conflict set η' by replacing each elementary atom in it with the corresponding original atom, as shown in Lemma 1. Using the HYP rule, we introduce all the atoms in $\eta_{\mathcal{N}^+}$, and combine them with repeated applications of the COMB rule: if $u_k \geq \sum_h \hat{a}_{kh}\, y_h$ is the atom corresponding to s_k, we use as coefficient for the COMB the a_{ij} (in the i-th row of the current tableau) such that $s_k \equiv x_j$. Then, we introduce each of the atoms in $\eta_{\mathcal{N}^-}$ with HYP, and add them to the previous combination, again using COMB. In this case, the coefficient to use is $-a_{ij}$. Finally, we introduce the atom in η_i and add it to the combination with coefficient 1.

Lemma 2. *The result of the linear combination described above is the atom $0 \leq c$, such that c is a numerical constant strictly lower than zero.*

Besides the case just described (and its dual when the inconsistency is due to an elementary atom $x_i \leq u_i$), another case in which an inconsistency can be detected is when two contradictory atoms are asserted: $l_k \leq \sum_h \hat{a}_{kh}\, y_h$ and $u_k \geq \sum_h \hat{a}_{kh}\, y_h$, with $l_k > u_k$. In this case, the proof is simply the combination of the two atoms with coefficient 1.

The extension for handling also equalities like $b_k = \sum_h \hat{a}_{kh}\, y_h$ is straightforward: we simply introduce two elementary atoms $b_k \leq s_k$ and $b_k \geq s_k$ and, in the construction of the proof, we use the LEQEQ rule to introduce the proper inequality.

Finally, notice that the current implementation in MATHSAT (see §6) is slightly different from what presented here, and significantly more efficient. In practice, η, η' are not constructed in sequence; rather, they are built simultaneously. Moreover, some optimizations are applied to eliminate some slack variables when they are not needed.

3.2 Interpolation with strict inequalities and disequalities

Another benefit of the Dutertre-de Moura algorithm is that it can handle strict inequalities directly. Its method is based on the following lemma.

Lemma 3 (Lemma 1 in [10]). *A set of linear arithmetic atoms Γ containing strict inequalities $S = \{0 < p_1, \ldots, 0 < p_n\}$ is satisfiable iff there exists a rational number $\varepsilon > 0$ such that $\Gamma_\varepsilon = (\Gamma \cup S_\varepsilon) \setminus S$ is satisfiable, where $S_\varepsilon = \{\varepsilon \leq p_1, \ldots, \varepsilon \leq p_n\}$.*

The idea of [10] is that of treating the *infinitesimal parameter* ε symbolically instead of explicitly computing its value. Strict bounds $(x < b)$ are replaced with weak ones $(x \leq b - \varepsilon)$, and the operations on bounds are adjusted to take ε into account.

We use the same idea also for computing interpolants. We transform every atom $(0 < t_i)$ occurring in the proof of unsatisfiability into $(0 \leq t_i - \varepsilon)$. Then we compute an interpolant I_ε in the usual way. As a consequence of the rules of [20], I_ε is always a single atom. As shown by the following lemma, if I_ε contains ε, then it must be in the form $(0 \leq t - c\,\varepsilon)$ with $c > 0$, and we can rewrite I_ε into $(0 < t)$.

Lemma 4 (Interpolation with strict inequalities). *Let Γ, S, Γ_ε and S_ε be defined as in Lemma 3. Let Γ be partitioned into A and B, and let A_ε and B_ε be obtained from A and B by replacing atoms in S with the corresponding ones in S_ε. Let I_ε be an interpolant for $(A_\varepsilon, B_\varepsilon)$. Then:*

- *If $\varepsilon \not\preceq I_\varepsilon$, then I_ε is an interpolant for (A, B).*
- *If $\varepsilon \preceq I_\varepsilon$, then $I_\varepsilon \equiv (0 \leq t - c\,\varepsilon)$ for some $c > 0$, and $I \equiv (0 < t)$ is an interpolant for (A, B).*

Thanks to Lemma 4, we can handle also negated equalities $(0 \neq t)$ directly. Suppose our set S of input atoms (partitioned into A and B) is the union of a set S' of equalities and inequalities (both weak and strict) and a set $S^{\neq}$ of disequalities, and suppose that S' is consistent. (If not so, an interpolant can be computed from S'.) Since $\mathcal{LA}(\mathbb{Q})$ is convex, S is inconsistent iff exists $(0 \neq t) \in S^{\neq}$ such that $S' \cup \{(0 \neq t)\}$ is inconsistent, that is, such that both $S' \cup \{(0 < t)\}$ and $S' \cup \{(0 > t)\}$ are inconsistent.

Therefore, we pick one element $(0 \neq t)$ of $S^{\neq}$ at a time, and check the satisfiability of $S' \cup \{(0 < t)\}$ and $S' \cup \{(0 > t)\}$. If both are inconsistent, from the two proofs we can generate two interpolants I^- and I^+. We combine I^+ and I^- to obtain an interpolant I for (A, B): if $(0 \neq t) \in A$, then I is $I^+ \vee I^-$; if $(0 \neq t) \in B$, then I is $I^+ \wedge I^-$, as shown by the following lemma.

Lemma 5 (Interpolation for negated equalities). *Let A and B two conjunctions of $\mathcal{LA}(\mathbb{Q})$ atoms, and let $n \equiv (0 \neq t)$ be one such atom. Let $g \equiv (0 < t)$ and $l \equiv (0 > t)$. If $n \in A$, then let $A^+ \equiv A \setminus \{n\} \cup \{g\}$, $A^- \equiv A \setminus \{n\} \cup \{l\}$, and $B^+ \equiv B^- \equiv B$. If $n \in B$, then let $A^+ \equiv A^- \equiv A$, $B^+ \equiv B \setminus \{n\} \cup \{g\}$, and $B^- \equiv B \setminus \{n\} \cup \{l\}$. Assume that $A^+ \wedge B^+ \models_{\mathcal{LA}(\mathbb{Q})} \bot$ and that $A^- \wedge B^- \models_{\mathcal{LA}(\mathbb{Q})} \bot$, and let I^+ and I^- be two interpolants for (A^+, B^+) and (A^-, B^-) respectively, and let*

$$I \equiv \begin{cases} I^+ \vee I^- \text{ if } n \in A \\ I^+ \wedge I^- \text{ if } n \in B. \end{cases}$$

Then I is an interpolant for (A, B).

4 Graph-based Interpolation for Difference Logic

Several interesting verification problems can be encoded using only a subset of $\mathcal{LA}(\mathbb{Q})$, the theory of Difference Logic ($\mathcal{DL}(\mathbb{Q})$), in which all atoms are inequalities of the form $(0 \leq y - x + c)$, where x and y are variables and c is a numerical constant. Equalities can be handled as conjunctions of inequalities. Here we do not consider the case when we also have strict inequalities $(0 < y - x + c)$, because in $\mathcal{DL}(\mathbb{Q})$ they can be handled in a way which is similar to that described in §3.2 for $\mathcal{LA}(\mathbb{Q})$. Moreover, we believe that our method may be extended straightforwardly to $\mathcal{DL}(\mathbb{Z})$ because the graph-based algorithm described in this section applies also to $\mathcal{DL}(\mathbb{Z})$; in $\mathcal{DL}(\mathbb{Z})$ a strict inequality $(0 < y - x + c)$ can be safely rewritten a priori into the inequality $(0 \leq y - x + c - 1)$.

$\mathcal{DL}(\mathbb{Q})$ is simpler than full linear arithmetic. Many SMT solvers use dedicated, graph-based algorithms for checking the consistency of a set of $\mathcal{DL}(\mathbb{Q})$ atoms [9, 23].

Intuitively, a set S of $\mathcal{DL}(\mathbb{Q})$ atoms induces a graph whose vertexes are the variables of the atoms, and there exists an edge $x \xrightarrow{c} y$ for every $(0 \leq y - x + c) \in S$. S is inconsistent if and only if the induced graph has a cycle of negative weight.

We now extend the graph-based approach to generate interpolants. Consider the interpolation problem (A, B) where A and B are sets of inequalities as above, and let C be (the set of atoms in) a negative cycle in the graph corresponding to $A \cup B$.

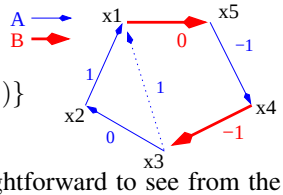
If $C \subseteq A$, then A is inconsistent, in which case the interpolant is $\bot$. Similarly, when $C \subseteq B$, the interpolant is $\top$. If neither of these occurs, then the edges in the cycle can be partitioned in subsets of A and B. We call maximal A-paths of C a path $x_1 \xrightarrow{c_1} \ldots \xrightarrow{c_{n-1}} x_n$ such that (I) $x_i \xrightarrow{c_i} x_{i+1} \in A$, and (II) C contains $x' \xrightarrow{c'} x_1$ and $x_n \xrightarrow{c''} x''$ that are in B. Clearly, the end-point variables x_1, x_n of the maximal A-path are such $x_1, x_n \preceq A$ and $x_1, x_n \preceq B$.

Let the *summary constraint* of a maximal A-path $x_1 \xrightarrow{c_1} \ldots \xrightarrow{c_{n-1}} x_n$ be the inequality $0 \leq x_n - x_1 + \sum_{i=1}^{n-1} c_i$. We claim that the conjunction of summary constraints of the A-paths of C is an interpolant. In fact, using the rules for $\mathcal{LA}(\mathbb{Q})$ it is easy to see that a maximal A-path entails its summary constraint. Hence, A entails the conjunction of the summary constraints of maximal A-paths. Then, we notice that the conjunction of the summary constraints is inconsistent with B. In fact, the weight of a maximal A-path and the weight of its summary constraint are the same. Thus the cycle obtained from C by replacing each maximal A-path with the corresponding summary constraint is also a negative cycle. Finally, we notice that every variable x occurring in the conjunction of the summary constraints is an end-point variable, and thus $x \preceq A$ and $x \preceq B$.

A final remark is in order. In principle, to generate a proof of unsatisfiability for a conjunction of $\mathcal{DL}(\mathbb{Q})$ atoms, the same rules used for $\mathcal{LA}(\mathbb{Q})$ [20] could be used. However, the interpolants generated from such proofs are in general not $\mathcal{DL}(\mathbb{Q})$ formulas anymore and, if computed starting from the same inconsistent set C, they are either identical or weaker than those generated with our method. In fact, due to the interpolation rules in [20], it is easy to see that the interpolant obtained is in the form $(0 \leq \sum_i t_i)$ s.t. $\bigwedge_i (0 \leq t_i)$ is the interpolant generated with our method.

Example 1. Consider the following sets of $\mathcal{DL}(\mathbb{Q})$ atoms:

$$A = \{(0 \leq x_1 - x_2 + 1), (0 \leq x_2 - x_3), (0 \leq x_4 - x_5 - 1)\}$$
$$B = \{(0 \leq x_5 - x_1), (0 \leq x_3 - x_4 - 1)\}.$$



corresponding to the negative cycle on the right. It is straightforward to see from the graph that the resulting interpolant is $(0 \leq x_1 - x_3 + 1) \wedge (0 \leq x_4 - x_5 - 1)$, because the first conjunct is the summary constraint of the first two conjuncts in A.

Applying instead the rules of Figure 1, the proof of unsatisfiability is:

$$\dfrac{\dfrac{\dfrac{\dfrac{\textsc{Hyp}}{(0 \leq x_1 - x_2 + 1)} \quad \dfrac{\textsc{Hyp}}{(0 \leq x_2 - x_3)}}{\textsc{Comb}\quad (0 \leq x_1 - x_3 + 1)} \quad \dfrac{\textsc{Hyp}}{(0 \leq x_4 - x_5 - 1)}}{\textsc{Comb}\quad (0 \leq x_1 - x_3 + x_4 - x_5)} \quad \dfrac{\textsc{Hyp}}{(0 \leq x_5 - x_1)}}{\textsc{Comb}\quad (0 \leq -x_3 + x_4)} \quad \dfrac{\textsc{Hyp}}{(0 \leq x_3 - x_4 - 1)}}{\textsc{Comb}\quad (0 \leq -1)}$$

By using the interpolation rules for $\mathcal{LA}(\mathbb{Q})$ (see [20]), the interpolant we obtain is $(0 \leq x_1 - x_3 + x_4 - x_5)$, which is not in $\mathcal{DL}(\mathbb{Q})$, and is weaker than that computed above.

5 Computing interpolants for combined theories via DTC

One of the typical approaches to the SMT problem in combined theories, $SMT(\mathcal{T}_1 \cup \mathcal{T}_2)$, is that of combining the solvers for $\mathcal{T}_1$ and for $\mathcal{T}_2$ with the Nelson-Oppen (NO) integration schema [22].

The NO framework works for combinations of *stably-infinite* and *signature-disjoint* theories $\mathcal{T}_i$ with equality. Moreover, it requires the input formula to be *pure* (i.e., s.t. all the atoms contain only symbols in one theory): if not, a *purification* step is performed, which might introduce some additional variables but preserves satisfiability. In this setting, the two decision procedures for $\mathcal{T}_1$ and $\mathcal{T}_2$ cooperate by exchanging (disjunctions of) implied *interface equalities*, that is, equalities between variables appearing in atoms of different theories (*interface variables*).

The work in [30] gives a method for generating an interpolant for a pair (A, B) of $\mathcal{T}_1 \cup \mathcal{T}_2$-formulas using the NO schema. Besides the requirements on $\mathcal{T}_1$ and $\mathcal{T}_2$ needed to use NO, it requires also that $\mathcal{T}_1$ and $\mathcal{T}_2$ are *equality-interpolating*. A theory $\mathcal{T}$ is said to be equality-interpolating when for all pairs of formulas (A, B) in $\mathcal{T}$ and for all equalities $x_a = x_b$ such that (i) $x_a \not\preceq B$ and $x_b \not\preceq A$ (i.e. $x_a = x_b$ is an AB-*mixed equality*), and (ii) $A \wedge B \models_{\mathcal{T}} x_a = x_b$, there exists a term t such that $A \wedge B \models_{\mathcal{T}} x_a = t \wedge t = x_b$, $t \preceq A$ and $t \preceq B$. E.g., both $\mathcal{EUF}$ and $\mathcal{LA}(\mathbb{Q})$ are equality-interpolating.

Recently, an alternative approach for combining theories in SMT has been proposed, called Delayed Theory Combination (DTC) [5, 6]. With DTC, the solvers for $\mathcal{T}_1$ and $\mathcal{T}_2$ do not communicate directly. The integration is performed by the SAT solver, by augmenting the boolean search space with up to all the possible interface equalities. DTC has several advantages wrt. NO, both in terms of ease of implementation and in reduction of search space [5, 6], so that many current SMT tools implement variants of DTC. In this Section, we give a method for generating interpolants for a pair of $\mathcal{T}_1 \cup \mathcal{T}_2$-formulas (A, B) when $\mathcal{T}_1$ and $\mathcal{T}_2$ are combined using DTC. As in [30], we assume that A and B have been purified using disjoint sets of auxiliary variables.

5.1 Combination without AB-mixed interface equalities

Let Eq be the set of all interface equalities introduced by DTC. We first consider the case in which Eq does not contain AB-mixed equalities. That is, Eq can be partitioned

into two sets $(Eq \setminus B) \equiv \{(x = y) | (x = y) \preceq A \; and \; (x = y) \not\preceq B\}$ and $(Eq \downarrow B) \equiv \{(x = y) | (x = y) \preceq B\}$. In this restricted case, nothing special needs to be done, despite the fact that the interface equalities in Eq do not occur neither in A nor in B, but might be introduced in the resolution proof $\mathcal{P}$ by $\mathcal{T}$-lemmas. This is because —as observed in [20]— as long as for an atom p either $p \preceq A$ or $p \preceq B$ holds, it is possible to consider it part of A (resp. of B) simply by assuming the tautology clause $p \vee \neg p$ to be part of A (resp. of B). Therefore, we can treat the interface equalities in $(Eq \setminus B)$ as if they appeared in A, and those in $(Eq \downarrow B)$ as if they appeared in B.

5.2 Combination with AB-mixed interface equalities

We can handle the case in which some of the equalities in Eq are AB-mixed under the hypothesis that $\mathcal{T}_1$ and $\mathcal{T}_2$ are equality-interpolating. Currently, we also require that $\mathcal{T}_1$ and $\mathcal{T}_2$ are convex, although the extension of the approach to non-convex theories is part of ongoing work.

The idea is similar to that used in [30] in the case of NO: using the fact that the $\mathcal{T}_i$'s are equality-interpolating, we reduce this case to the previous one by "splitting" every AB-mixed interface equality $(x_a = x_b)$ into the conjunction of two parts $(x_a = t) \wedge (t = x_b)$, such that $(x_a = t) \preceq A$ and $(t = x_b) \preceq B$. The main difference is that we do this *a posteriori*, after the construction of the resolution proof of unsatisfiability $\mathcal{P}$. This makes it possible to compute different interpolants for different partitions of the input problem into an A-part and a B-part *from the same proof* $\mathcal{P}$. Besides the advantage in performance of not having to recompute the proof every time, this is particularly important in some application domains like abstraction refinement [11], where the relation between interpolants obtained from the same proof tree is exploited to prove some properties of the refinement procedure. [9] To do this, we traverse $\mathcal{P}$ and split every AB-mixed equality in it, performing also the necessary manipulations to ensure that the modified DAG is still a resolution proof of unsatisfiability (according to the definition in §2.2). As long as this requirement is met, our technique is independent from the exact procedure implementing it. In the rest of this Section, we describe the algorithm that we have implemented, for the combination $\mathcal{EUF} \cup \mathcal{LA}(\mathbb{Q})$. Due to lack of space, we can not describe it in detail, rather we only provide the main intuitions.

First, we control the branching and learning heuristics of the SMT solver to ensure that the generated resolution proof of unsatisfiability $\mathcal{P}$ has a property that we call *locality wrt. interface equalities*. We say that $\mathcal{P}$ is local wrt. interface equalities (ie-local) if the interface equalities occur only in subproofs $\mathcal{P}_i^{\mathsf{ie}}$ of $\mathcal{P}$, in which both the root and the leaves are $\mathcal{T}_1 \cup \mathcal{T}_2$-valid, the leaves of $\mathcal{P}_i^{\mathsf{ie}}$ are also leaves of $\mathcal{P}$, the root of $\mathcal{P}_i^{\mathsf{ie}}$ does not contain any interface equality, and in $\mathcal{P}_i^{\mathsf{ie}}$ all the pivots are interface equalities.

[9] In particular, the following relation: $I_{A,B\cup C}(\mathcal{P}) \wedge C \implies I_{A\cup C,B}(\mathcal{P})$ (where $I_{A,B}(\mathcal{P})$ is an interpolant for (A, B) generated from the proof $\mathcal{P}$) is used to show that for every spurious counterexample found, the interpolation-based refinement procedure is able to rule-out the counterexample in the refined abstraction [11]. It is possible to show that a similar relation holds also for $I_{A,B\cup C}(\mathcal{P}_1)$ and $I_{A\cup C,B}(\mathcal{P}_2)$, when $\mathcal{P}_1$ and $\mathcal{P}_2$ are obtained from the same $\mathcal{P}$ by splitting AB-mixed interface equalities with the technique described here. However, for lack of space we can not include such proof.

In order to generate ie-local proofs, we adopt a variant of the DTC Strategy 1 of [6]. We never select an interface equality for case splitting if there is some other unassigned atom, and we always assign false to interface equalities first. Moreover, when splitting on interface equalities, we restrict both the backjumping and the learning procedures of the DPLL engine as follows. Let d be the depth in the DPLL tree at which the first interface equality is selected for case splitting. If during the exploration of the current DPLL branch we have to backjump above d, then we generate by resolution a conflict clause that does not contain any interface equality, and "deactivate" all the $\mathcal{T}$-lemmas containing some interface equality, so that they can not be used elsewhere in the search tree. Only when we start splitting on interface equalities again, we can re-activate such $\mathcal{T}$-lemmas.

The idea of the Strategy just described is that of "emulating" the NO combination of the two $\mathcal{T}_i$-solvers. The conflict clause generated by resolution plays the role of the $\mathcal{T}$-lemma generated by the NO-based $\mathcal{T}_1 \cup \mathcal{T}_2$ solver, and the $\mathcal{T}$-lemmas containing positive interface equalities are used for exchanging implied equalities. The difference is that the combination is performed by the DPLL engine, and encoded directly in the ie-local subproofs $\mathcal{P}_i^{\mathsf{ie}}$ of $\mathcal{P}$.

Since AB-mixed equalities can only occur in $\mathcal{P}_i^{\mathsf{ie}}$ subproofs, we can handle the rest of $\mathcal{P}$ in the usual way. Therefore, we now describe only how to manipulate the $\mathcal{P}_i^{\mathsf{ie}}$'s such that all the AB-mixed equalities are split.

In order accomplish this task, we exploit the following fact: since we are considering only convex theories, all the $\mathcal{T}_i$-lemmas generated by the $\mathcal{T}_i$-solvers contain at most one positive interface equality $(x = y)$.[10] Let $C \equiv (x = y) \vee \neg\eta$ be one such $\mathcal{T}_i$-lemma. Then $\eta \models_{\mathcal{T}_i} (x = y)$. Since $\mathcal{T}_i$ is equality-interpolating, if $(x = y)$ is AB-mixed, we can split C into $C_1 \equiv (x = t) \vee \neg\eta$ and $C_2 \equiv (t = y) \vee \neg\eta$. (E.g. by using the algorithms given in [30] for $\mathcal{EUF}$ and $\mathcal{LA}(\mathbb{Q})$.) Then, we replace every occurrence of $\neg(x = y)$ in the leaves of $\mathcal{P}_i^{\mathsf{ie}}$ with the disjunction $\neg(x = t) \vee \neg(t = y)$. Finally, we replace the subproof

$$\dfrac{(x = y) \vee \neg\eta \qquad \neg(x = y) \vee \phi}{\neg\eta \vee \phi} \quad \text{with} \quad \dfrac{\dfrac{(x = t) \vee \neg\eta \qquad \neg(x = t) \vee \neg(t = y) \vee \phi}{\neg\eta \vee \neg(t = y) \vee \phi} \qquad (t = y) \vee \neg\eta}{\neg\eta \vee \phi}.$$

If this is done recursively, starting from $\mathcal{T}_i$-lemmas $\neg\eta \vee (x = y)$ such that $\neg\eta$ contains no negated AB-mixed equality, then the procedure terminates and the new proof $\mathcal{P}_i^{\mathsf{ie}\,\prime}$ contains no AB-mixed equality.

Finally, we wish to remark that what just described is only one possible way of splitting AB-mixed equalities in $\mathcal{P}$. In particular, the restrictions on the branching and learning heuristics needed to generate ie-local proofs might have a negative impact in the performance of the SMT solver. In fact, we are currently investigating some alternative strategies.

[10] There is a further technical condition that must be satisfied by the $\mathcal{T}_i$-solvers, i.e. they must not generate conflict sets containing redundant disequalities. This is true for all the $\mathcal{T}_i$-solvers on $\mathcal{EUF}$, $\mathcal{DL}(\mathbb{Q})$ and $\mathcal{LA}(\mathbb{Q})$ implemented in MATHSAT.

Family	# of problems	MATHSAT-ITP	FOCI	CLP-PROVER
kbfiltr.i	64	0.16	0.36	1.47
diskperf.i	119	0.33	0.78	3.08
floppy.i	235	0.73	1.64	5.91
cdaudio.i	130	0.35	1.07	2.98

Fig. 2. Comparison of execution times of MATHSAT-ITP, FOCI and CLP-PROVER on problems generated by BLAST.

6 Experimental evaluation

The techniques presented in previous sections have been implemented within MATHSAT 4 [4] (Hereafter, we will refer to such implementation as MATHSAT-ITP). MATHSAT is an SMT solver supporting a wide range of theories and their combinations. In the last SMT solvers competition (SMT-COMP'07), it has proved to be competitive with the other state-of-the-art solvers. In this Section, we experimentally evaluate our approach.

6.1 Description of the benchmark sets

We have performed our experiments on two different sets of benchmarks. The first is obtained by running the BLAST software model checker [11] on some Windows device drivers; these are similar to those used in [27]. This is one of the most important applications of interpolation in formal verification, namely abstraction refinement in the context of CEGAR. The problem represents an abstract counterexample trace, and consists of a conjunction of atoms. In this setting, the interpolant generator is called very frequently, each time with a relatively simple input problem.

The second set of benchmarks originates from the SMT-LIB [26], and is composed of a subset of the unsatisfiable problems used in the 2007 SMT solvers competition (`http://www.smtcomp.org`). The instances have been converted to CNF and then split in two consistent parts of approximately the same size. The set consists of problems of varying difficulty and with a nontrivial boolean structure.

The experiments have been performed on a 3GHz Intel Xeon machine with 4GB of RAM running Linux. All the tools were run with a timeout of 600 seconds and a memory limit of 900 MB.

6.2 Comparison with the state-of-the-art tools available

In this section, we compare with the only other interpolant generators which are available: FOCI [20, 13] and CLP-PROVER [27]. Other natural candidates for comparison would have been ZAP [3] and LIFTER [16]; however, it was not possible to obtain them from the authors.

The comparison had to be adapted to the limitations of FOCI and CLP-PROVER. In fact, the current version of FOCI does not handle the full $\mathcal{LA}(\mathbb{Q})$, but only the $\mathcal{DL}(\mathbb{Q})$

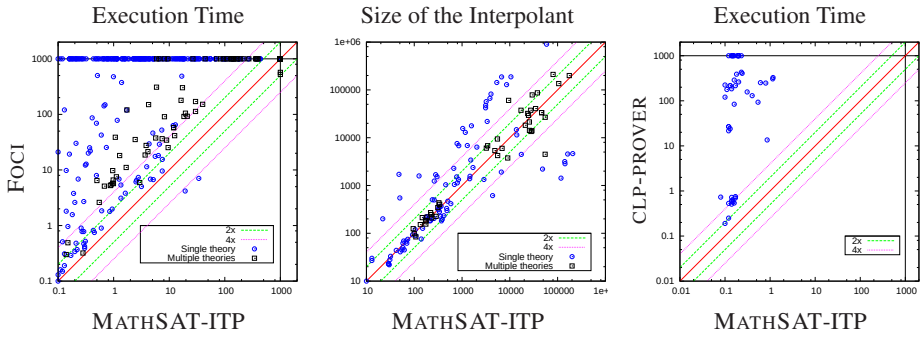


Fig. 3. Comparison of MATHSAT-ITP and FOCI on SMT-LIB instances: execution time (left), and size of the interpolant (right). In the left plot, points on the horizontal and vertical lines are timeouts/failures.

Fig. 4. Comparison of MATHSAT-ITP and CLP-PROVER on conjunctions of $\mathcal{LA}(\mathbb{Q})$ atoms.

fragment[11]. We also notice that the interpolants it generates are not always $\mathcal{DL}(\mathbb{Q})$ formulas. (See, e.g., Example 1 of Section 4.) CLP-PROVER, on the other hand, does handle the full $\mathcal{LA}(\mathbb{Q})$, but it accepts only conjunctions of atoms, rather than formulas with arbitrary boolean structure. These limitations made it impossible to compare all the three tools on all the instances of our benchmark sets. Therefore, we perform the following comparisons:

- We compare all the three solvers on the problems generated by BLAST;
- We compare MATHSAT-ITP with FOCI on SMT-LIB instances in the theories of $\mathcal{EUF}$, $\mathcal{DL}(\mathbb{Q})$ and their combination. In this case, we compare both the execution times and the sizes of the generated interpolants (in terms of number of nodes in the DAG representation of the formula). For computing interpolants in $\mathcal{EUF}$, we apply the algorithm of [20], using an extension of the algorithm of [24] to generate $\mathcal{EUF}$ proof trees. The combination $\mathcal{EUF} \cup \mathcal{DL}(\mathbb{Q})$ is handled with the technique described in §5;
- We compare MATHSAT-ITP and CLP-PROVER on $\mathcal{LA}(\mathbb{Q})$ problems consisting of conjunctions of atoms. These problems are single branches of the search trees explored by MATHSAT for some $\mathcal{LA}(\mathbb{Q})$ instances in the SMT-LIB. We have collected several problems that took more than 0.1 seconds to MATHSAT to solve, and then randomly picked 50 of them. In this case, we do not compare the sizes of the interpolants as they are always atomic formulas.

The results are collected in Figures 2, 3 and 4. We can observe the following facts:

- Interpolation problems generated by BLAST are trivial for all the tools. In fact, we even had some difficulties in measuring the execution times reliably. Despite this, MATHSAT-ITP seems to be a little faster than the others.

[11] For example, it fails to detect the $\mathcal{LA}(\mathbb{Q})$-unsatisfiability of the following problem: $(0 \leq y - x + w) \wedge (0 \leq x - z - w) \wedge (0 \leq z - y - 1)$.

– For problems with a nontrivial boolean structure, MATHSAT-ITP outperforms FOCI in terms of execution time. This is true even for problems in the combined theory $\mathcal{EUF} \cup \mathcal{DL}(\mathbb{Q})$, despite the fact that the current implementation is still preliminary.
– In terms of size of the generated interpolants, the gap between MATHSAT-ITP and FOCI is smaller on average. However, the right plot of Figure 3 (which considers only instances for which both tools were able to generate an interpolant) shows that there are more cases in which MATHSAT-ITP produces a smaller interpolant.
– On conjunctions of $\mathcal{LA}(\mathbb{Q})$ atoms, MATHSAT-ITP outperforms CLP-PROVER, sometimes by more than two orders of magnitude.

7 Conclusions

In this paper, we have shown how to efficiently build interpolants using state-of-the-art SMT solvers. Our methods encompass a wide range of theories (including $\mathcal{EUF}$, difference logic, and linear arithmetic), and their combination (based on the Delayed Theory Combination schema). A thorough experimental evaluation shows that the proposed methods are vastly superior to the state of the art interpolants, both in terms of expressiveness, and in terms of efficiency.

In the future, we plan to investigate the following issues. First, we will improve the implementation of the interpolation method for combined theories, that is currently rather naïve, and limited to the case of convex theories. Second, we will investigate interpolation with other rules, in particular Ackermann's expansion. Finally, we will integrate our interpolator within a CEGAR loop based on decision procedures, such as BLAST or the new version of NuSMV. In fact, such an integration raises interesting problems related to controlling the structure of the generated interpolants [13, 14], e.g. in order to limit the number or the size of constants occurring in the proof.

References

1. G. Audemard, M. Bozzano, A. Cimatti, and R. Sebastiani. Verifying industrial hybrid systems with mathsat. *Electr. Notes Theor. Comput. Sci.*, 119(2), 2005.
2. G. Audemard, A. Cimatti, A. Kornilowicz, and R. Sebastiani. Bounded model checking for timed systems. In *Proc. FORTE*, volume 2529 of *LNCS*. Springer, 2002.
3. T. Ball, S. K. Lahiri, and M. Musuvathi. Zap: Automated theorem proving for software analysis. In *Proc. LPAR*, volume 3835 of *LNCS*. Springer, 2005.
4. M. Bozzano, R. Bruttomesso, A. Cimatti, T. Junttila, P. Rossum, S. Schulz, and R. Sebastiani. MathSAT: A Tight Integration of SAT and Mathematical Decision Procedure. *Journal of Automated Reasoning*, 35(1-3), October 2005.
5. M. Bozzano, R. Bruttomesso, A. Cimatti, T. Junttila, P. van Rossum, S. Ranise, and R. Sebastiani. Efficient Theory Combination via Boolean Search. *Information and Computation*, 204(10), 2006.
6. R. Bruttomesso, A. Cimatti, A. Franzén, A. Griggio, and R. Sebastiani. Delayed Theory Combination vs. Nelson-Oppen for Satisfiability Modulo Theories: A Comparative Analysis. In *Proc. LPAR*, volume 4246 of *LNCS*. Springer, 2006.
7. G. Cabodi, M. Murciano, S. Nocco, and S. Quer. Stepping forward with interpolants in unbounded model checking. In *Proc. ICCAD'06,*. ACM, 2006.

8. A. Cimatti, A. Griggio, and R. Sebastiani. Efficient Interpolant Generation in Satisfiability Modulo Theories. Technical Report DIT-07-075, DISI - University of Trento, 2007.
9. S. Cotton and O. Maler. Fast and Flexible Difference Constraint Propagation for DPLL(T). In *Proc. SAT*, volume 4121 of *LNCS*. Springer, 2006.
10. B. Dutertre and L. de Moura. A Fast Linear-Arithmetic Solver for DPLL(T). In *Proc .CAV*, volume 4144 of *LNCS*, 2006.
11. T. A. Henzinger, R. Jhala, R. Majumdar, and K. L. McMillan. Abstractions from proofs. In N. D. Jones and X. Leroy, editors, *POPL*. ACM, 2004.
12. R. Jhala and K. McMillan. Interpolant-based transition relation approximation. In *Proc. CAV*, volume 3576 of *LNCS*. Springer, 2005.
13. R. Jhala and K. L. McMillan. A Practical and Complete Approach to Predicate Refinement. In H. Hermanns and J. Palsberg, editors, *TACAS*, volume 3920 of *LNCS*. Springer, 2006.
14. R. Jhala and K. L. McMillan. Array Abstractions from Proofs. In W. Damm and H. Hermanns, editors, *CAV*, volume 4590 of *LNCS*. Springer, 2007.
15. D. Kapur, R. Majumdar, and C. G. Zarba. Interpolation for data structures. In M. Young and P. T. Devanbu, editors, *SIGSOFT FSE*. ACM, 2006.
16. D. Kroening and G. Weissenbacher. Lifting Propositional Interpolants to the Word-Level. In *FMCAD*, pages 85–89, Los Alamitos, CA, USA, 2007. IEEE Computer Society.
17. B. Li and F. Somenzi. Efficient Abstraction Refinement in Interpolation-Based Unbounded Model Checking. In *Proc. TACAS*, volume 3920 of *LNCS*. Springer, 2006.
18. J. Marques-Silva. Interpolant Learning and Reuse in SAT-Based Model Checking. *Electr. Notes Theor. Comput. Sci.*, 174(3):31–43, 2007.
19. K. McMillan. Interpolation and SAT-based model checking. In *Proc. CAV*, 2003.
20. K. L. McMillan. An interpolating theorem prover. *Theor. Comput. Sci.*, 345(1), 2005.
21. K. L. McMillan. Lazy Abstraction with Interpolants. In *Proc CAV*, volume 4144 of *LNCS*. Springer, 2006.
22. G. Nelson and D. Oppen. Simplification by Cooperating Decision Procedures. *ACM Trans. on Programming Languages and Systems*, 1(2), 1979.
23. R. Nieuwenhuis and A. Oliveras. DPLL(T) with Exhaustive Theory Propagation and Its Application to Difference Logic. In *Proc. CAV*, volume 3576 of *LNCS*. Springer, 2005.
24. R. Nieuwenhuis and A. Oliveras. Fast Congruence Closure and Extensions. *Inf. Comput.*, 2005(4):557–580, 2007.
25. P. Pudlák. Lower bounds for resolution and cutting planes proofs and monotone computations. *J. of Symb. Logic*, 62(3), 1997.
26. S. Ranise and C. Tinelli. The Satisfiability Modulo Theories Library (SMT-LIB). `www.SMT-LIB.org`, 2006.
27. A. Rybalchenko and V. Sofronie-Stokkermans. Constraint Solving for Interpolation. In *VMCAI*, LNCS. Springer, 2007.
28. R. Sebastiani. Lazy Satisfiability Modulo Theories. *Journal on Satisfiability, Boolean Modeling and Computation, JSAT*, Volume 3, 2007.
29. R. J. Vanderbei. *Linear Programming: Foundations and Extensions*. Springer, 2001.
30. G. Yorsh and M. Musuvathi. A combination method for generating interpolants. In R. Nieuwenhuis, editor, *CADE*, volume 3632 of *LNCS*. Springer, 2005.