Preprint from Proceedings of DATE 14, Dresden, Germany, March 2014

Simple Interpolants for Linear Arithmetic

Christoph Scholl, Florian Pigorsch, and Stefan Disch
University of Freiburg, Germany
{scholl, pigorsch, disch}@informatik.uni-freiburg.de

Ernst Althaus
University of Mainz, Germany
ernst.althaus@uni-mainz.de

***Abstract*—Craig interpolation has turned out to be an essential method for many applications in formal verification. In this paper we focus on the computation of *simple* interpolants for the theory of linear arithmetic with rational coefficients. We successfully minimize the number of linear constraints in the final interpolant by several methods including proof transformations, linear programming, and SMT solving. Experimental results comparing the approach to standard methods from the literature prove the effectiveness of the approach and show reductions of up to 70% in the number of linear constraints.**

I. Introduction

During the last years the computation of *Craig interpolants* [1] for SAT and SMT formulas has attracted a lot of interest, mainly for applications in formal verification. For mutually unsatisfiable formulas A and B, a Craig Interpolant is a formula I, such that I is implied by A, I and B are mutually unsatisfiable, and the uninterpreted symbols in I occur both in A and B as well as the free variables in I occur freely both in A and B.

Efficient interpolation algorithms have first been introduced for Boolean systems. They rely on the enormous gain in efficiency of modern SAT solvers and the observation that DPLL-based SAT solving with learning of conflict clauses can provide resolution proofs of unsatisfiability as a byproduct [2]. According to [3], [4] a Craig interpolant can be computed in linear time based on a resolution proof of unsatisfiability for $\mathsf{A} \wedge \mathsf{B}$. In [4] interpolants have been introduced into the verification domain and have been used as over-approximations of reachable state sets; their use turns bounded model-checking into a complete method.

Modeling by Boolean formulas is not adequate for many systems of practical interest which go beyond hardware components (such as software programs, timed systems, or hybrid systems). For handling such systems SAT solvers have been generalized to SMT ("SAT Modulo Theory") solvers. SMT solvers for several fragments of first-order logic have been developed [5] and SMT interpolation has been introduced [6], [7]. Those interpolants have been successfully applied in *software verification* using predicate abstraction and refinement [8]–[11]. Moreover, for the verification of *hybrid systems* interpolants have been used to optimize symbolic state set representations [12], [13]. Interpolants play another role in the verification of timed and hybrid systems, when bounded model checking for those systems [14]–[16] is combined with the ideas from [4], [6].

In general, an interpolant between two formulas A and B is by far not unique. Therefore many researchers have been looking for *simple* interpolants.

In the context of Boolean interpolation simplicity is often understood as compact size, and interpolants with small And-Inverter-Graph representations are preferred. For applications of interpolation in logic synthesis [17], [18] this optimization goal is near at hand, but also in verification applications (when interpolants may be used as symbolic state set predicates, e.g.) not only their logical strength, but also their size has an essential impact on the efficiency of the overall verification algorithm. A number of approaches restructure resolution proofs of unsatisfiability in various ways to obtain smaller interpolants afterwards [19]–[23]. Other approaches consider proof transformations [24] and new interpolation systems [25], [26] which aim at influencing the strength (in a logical sense) of interpolants and not their size.

Simplicity of interpolants has been considered for formulas of (fragments of) first-order logic as well. In [27] both the size of interpolants and the number of linear constraints have been considered as measures of simplicity. Proofs are modified with the goal of replacing linear constraints in interpolants by constants (additionally leading to smaller interpolant sizes by constant propagation). Interpolants are used there in order to optimize or approximate state set representations of hybrid systems [13]. In [28] a general interpolation technique which applies for arbitrary theories has been presented (possibly leading to interpolants with quantifiers). [28] computes "simple interpolants" with several optimization goals: the (weighted or unweighted) number of ground atoms in the interpolant or the number of quantifiers in the interpolant. In [29] interpolation is used for software verification. [29] shows that simple invariants (in terms of the number of linear constraints in the interpolant) may be beneficial for an improved generation of program invariants.

In our paper we consider interpolation for the theory of linear arithmetic with rational coefficients $\mathcal{LA}(\mathbb{Q})$. Our interpolant computation is based on [6], [7], i.e., the Boolean structure of the interpolant results from the resolution proof graph whereas clauses corresponding to conflicts in the underlying theory (called *theory lemmata*) contribute to the interpolant by linear inequations (so-called $\mathcal{LA}(\mathbb{Q})$-interpolants). During SMT solving each theory lemma results from a *theory conflict*, i.e., an inconsistent conjunction of linear constraints. The mentioned $\mathcal{LA}(\mathbb{Q})$-interpolants are computed by $\mathcal{LA}(\mathbb{Q})$-interpolation for a partition of theory conflicts into two parts. We compute *simple* interpolants with less linear inequations by generalizing the approach from [30] which computes $\mathcal{LA}(\mathbb{Q})$-interpolants by linear programming. In contrast to [30] we do not compute $\mathcal{LA}(\mathbb{Q})$-interpolants for single theory conflicts, but we compute *shared* $\mathcal{LA}(\mathbb{Q})$-interpolants for a maximal number of theory conflicts, and thus we minimize the number of linear inequations occurring in the interpolant. In that way our interpolation approach fits seamlessly into existing approaches for interpolation based on proofs [6], [7]. We provide two algorithms for minimizing the number of linear inequations: The first algorithm greedily constructs larger and larger sets of shared $\mathcal{LA}(\mathbb{Q})$-interpolants by linear programming; the second algorithm maximizes the number of shared $\mathcal{LA}(\mathbb{Q})$-interpolants by solving an SMT problem. From first experiments we learned that especially with the *minimized* theory conflicts learned by modern SMT solvers the potential of finding shared $\mathcal{LA}(\mathbb{Q})$-interpolants was much smaller than expected. For that reason we propose several methods to increase the degrees of freedom for selecting

This work was partly supported by the German Research Council (DFG) as part of the Transregional Collaborative Research Center "Automatic Verification and Analysis of Complex Systems" (http://www.avacs.org/).

978-3-9815370-2-4/DATE14/©2014 EDAA

$\mathcal{LA}(\mathbb{Q})$-interpolants: (1) We relax the constraints given in [30] in a natural way, (2) we extend theory conflicts by so-called *implied literals*, and (3) we use the *push-up* method from [27] to extend theory conflicts. Whereas (3) transforms a resolution proof of unsatisfiability into another valid resolution proof [27] (but with more degrees of freedom for $\mathcal{LA}(\mathbb{Q})$-interpolants), (2) may destroy the resolution proof in general. However we can prove that the interpolants computed after this transformation are still valid interpolants though.

Our approach coincides with [28] in the general goal of computing simple interpolants. Whereas [28] is very general by considering arbitrary closed first-order formulas and reduces interpolant minimization to solving pseudo-boolean constraints with a number of variables which is linear in the size of the (potentially large) proof, our method is currently restricted to a special theory ($\mathcal{LA}(\mathbb{Q})$) and focusses more on efficiency. In [28] the interpolants are Boolean combinations of subformulas occurring in a proof of unsatisfiablity. Our approach computes "smooth" interpolants with a small number of linear constraints not necessarily occurring in the proof. Recently, [29] considered shared $\mathcal{LA}(\mathbb{Q})$-interpolants as well, but not in the context of interpolant generation from resolution proofs, but in the context of "compositional SMT" which constructs interpolants step by step by considering so-called samplesets (disjunctions of polytopes implying the original formulas A and B for which shared $\mathcal{LA}(\mathbb{Q})$-interpolants are computed). In compositional SMT the SMT solver never sees the formula $\mathsf{A} \wedge \mathsf{B}$, but only combinations of the current candidate interpolant and A (resp. B). By construction, the computed interpolant in [29] is always in disjunctive normal form, whereas our approach produces interpolants with an arbitrary Boolean structure (which is certainly advantageous for examples combining linear constraints with non-trivial Boolean subformulas with many variables). [27] also considers linear arithmetic, but is more restricted than our method, since its Lemma Localization approach is only able to optimize interpolants by replacing linear constraints by constants.

Besides the motivation given by invariant generation given in [29], interpolants with a small number of linear constraints may also play an important role when state sets are represented by formulas in linear arithmetic, e.g. in hybrid system verification [13], [31], [32]. Especially when those state set representations are subject to operations whose complexity strongly depends on the number of linear constraints in the representation (like quantifier elimination methods for rational variables which in the worst-case lead to a quadratic increase of the number of linear constraints after elimination of a single variable), such applications may profit from simple interpolants with a minimized number of linear constraints. Simple interpolants may optimize given state set representations or approximate them by a "smoother shape" with less linear constraints. For applications with fully symbolic representations of state sets like [13] it has already been shown that intensive compaction efforts based on interpolation are the key ingredient to avoid exploding state set representations. In those applications interpolants can be directly used for further processing, in other applications with semi-symbolic representations (e.g. unions of polyhedra) [31], [32] a back-translation into the used representation form may be necessary after "smoothening" the state set.

The paper is structured as follows: We will give a brief review of SMT solving, theory proofs, and interpolation in Sect. II. Our approach computing simple interpolants is presented in Sect. III, together with an illustration by means of a running example. Our method is extensively evaluated and compared to interpolation in MathSAT [33] and to interpolation with Lemma Localization [27] in Sect. IV. Sect. V summarizes the paper and gives directions for future research.

II. Preliminaries

A *signature* Σ is a collection of function symbols and predicate symbols. A *theory* $\mathcal{T}$ gives interpretations to a subset of the symbols occuring in Σ. These symbols are called $\mathcal{T}$-symbols, symbols without interpretations are are called *uninterpreted*. A *term* is a first-order term built from the function symbols of Σ. For terms $t_1, \ldots, t_n$ and an n-ary predicate p, $p(t_1, \ldots, t_n)$ is an *atom*. An uninterpreted 0-ary atom is called proposition or *Boolean variable*. A (quantifier-free) *formula* is a Boolean combination of atoms. A *literal* is either an atom or the negation of an atom. A literal built from an n-ary interpreted atom with $n > 0$ is called $\mathcal{T}$-literal. A *clause* is a disjunction of literals; for a clause $l_1 \vee \ldots \vee l_n$ we also use the set-notation $\{l_1, \ldots, l_n\}$. An empty clause, which is equivalent to $\bot$, is denoted with $\emptyset$. A clause, which contains a literal l and its negation $\neg l$, is called *tautologic* clause, since it is equivalent to $\top$. In this paper we only consider *non-tautologic* clauses.

Let C be a clause and ϕ be a formula. With $C \setminus \phi$, we denote the clause that is created from C by removing all atoms occurring in ϕ; $C \downarrow \phi$ denotes the clause that is created from C by removing all atoms that are not occurring in ϕ.

A formula is $\mathcal{T}$-satisfiable if it is satisfiable in $\mathcal{T}$, i.e., if there is a model for the formula where the $\mathcal{T}$-symbols are interpreted according to the theory $\mathcal{T}$. If a formula ϕ logically implies a formula ψ in all models of $\mathcal{T}$, we write $\phi \models_{\mathcal{T}} \psi$. *Satisfiability Modulo Theory* $\mathcal{T}$ (SMT($\mathcal{T}$)) is the problem of deciding the $\mathcal{T}$-satisfiability of a formula ϕ.

Typical SMT($\mathcal{T}$)-solvers combine DPLL-style SAT-solving [34] with a separate decision procedure for reasoning on $\mathcal{T}$ [5]. Such a solver treats all atomic predicates in a formula ϕ as free Boolean variables. Once the DPLL-part of the solver finds a satisfying assignment, e. g. $l_1 \wedge \ldots \wedge l_n$, to this "Boolean abstraction", it passes the atomic predicates corresponding to the assignment to a decision procedure for $\mathcal{T}$, which then checks whether the assignment is feasible when interpreted in the theory $\mathcal{T}$.[1] If the assignment is feasible, the solver terminates since a satisfying assignment to the formula ϕ has been found. If the assignment is infeasible in $\mathcal{T}$, the decision procedure derives a cause for the infeasibility of the assignment, say $\eta = m_1 \wedge \ldots \wedge m_k$, where $\{m_1, \ldots, m_k\} \subseteq \{l_1, \ldots, l_n\}$. We call the cause η a $\mathcal{T}$-*conflict*, since $\eta \models_{\mathcal{T}} \bot$. The SMT($\mathcal{T}$)-solver then adds the negation of the cause, $\neg\eta = \{\neg m_1, \ldots, \neg m_k\}$, which we call $\mathcal{T}$-*lemma*, to its set of clauses and starts backtracking. The added $\mathcal{T}$-lemma prevents the DPLL-procedure from selecting the same invalid assignment again. Usually, the $\mathcal{T}$-conflicts η used in modern SMT-solvers are reduced to minimal size (i. e. η becomes $\mathcal{T}$-satisfiable, if one of its literals is removed) in order to prune the search space as much as possible. Such $\mathcal{T}$-conflicts η are often called *minimal infeasible subsets*.

One can extend an SMT($\mathcal{T}$)-solver of this style in a straightforward way to produce proofs for the unsatisfiability of formulas [7].

Definition 1 ($\mathcal{T}$-Proof): Let $S = \{c_1, \ldots, c_n\}$ be a set of non-tautologic clauses and C a clause. A DAG P is a *resolution proof* for the deduction of $\bigwedge c_i \models_{\mathcal{T}} C$, if
(1) each leaf $n \in P$ is associated with a clause n_{cl}; n_{cl} is either a clause of S or a $\mathcal{T}$-lemma ($n_{cl} = \neg\eta$ for some $\mathcal{T}$-conflict η);
(2) each non-leaf $n \in P$ has exactly two parents n^L and n^R, and is associated with the clause n_{cl} which is derived from n^L_{cl} and n^R_{cl} by resolution, i. e. the parents' clauses share a common variable (the *pivot*) n_p such that $n_p \in n^L_{cl}$ and $\neg n_p \in n^R_{cl}$, and $n_{cl} = n^L_{cl} \setminus \{n_p\} \cup n^R_{cl} \setminus \{\neg n_p\}$; n_{cl} (the *resolvent*) must not

[1] For simplicity this review describes the lazy SMT approach [35] using an *off-line schema* instead of an *on-line schema* where already *partial* assignments to Boolean abstraction variables are checked for consistency with the theory.

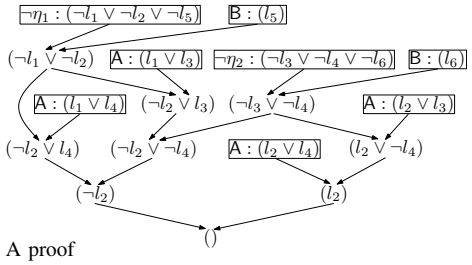

Figure 1. A proof

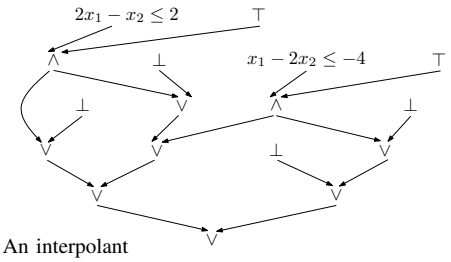

Figure 2. An interpolant

be a tautology;
(3) there is exactly one root node $r \in P$; r is associated with clause C; $r_{cl} = C$.

Intuitively, a resolution proof provides a means to derive a clause C from the set of clauses S and some additional facts of the theory $\mathcal{T}$. If C is the empty clause, P is proving the $\mathcal{T}$-unsatisfiability of S.

Example 1: Fig. 1 shows a resolution proof for the unsatisfiability of $S = (l_1 \vee l_3) \wedge (l_1 \vee l_4) \wedge (l_2 \vee l_3) \wedge (l_2 \vee l_4) \wedge l_5 \wedge l_6$ with $l_1 = (-x_2 \leq 0)$, $l_2 = (x_1 \leq 1)$, $l_3 = (-x_2 \leq -5)$, $l_4 = (x_1 \leq 6)$, $l_5 = (-2x_1 + x_2 \leq -6)$, $l_6 = (-x_1 + 2x_2 \leq 0)$. To prove the unsatisfiability, the solver added two $\mathcal{T}$-lemmata $\neg\eta_1 = (\neg l_1 \vee \neg l_2 \vee \neg l_5)$ and $\neg\eta_2 = (\neg l_3 \vee \neg l_4 \vee \neg l_6)$.

Definition 2 (Craig Interpolant [1]): Let A and B be two formulas, such that $\mathsf{A} \wedge \mathsf{B} \models_{\mathcal{T}} \bot$. A *Craig interpolant* I is a formula such that (1) $\mathsf{A} \models_{\mathcal{T}} I$, (2) $\mathsf{B} \wedge I \models_{\mathcal{T}} \bot$, (3) the uninterpreted symbols in I occur both in A and B, the free variables in I occur freely both in A and B.

Given a $\mathcal{T}$-unsatisfiable set of clauses $S = \{c_1, \ldots, c_n\}$, a disjoint partition (A, B) of S, and a proof P for the $\mathcal{T}$-unsatisfiability of S, an interpolant for (A, B) can be constructed by the following procedure [6]:
(1) For every leaf $n \in P$ associated with a clause $n_{cl} \in S$, set $n_I = n_{cl} \downarrow \mathsf{B}$ if $n_{cl} \in \mathsf{A}$, and set $n_I = \top$ if $n_{cl} \in \mathsf{B}$.
(2) For every leaf $n \in P$ associated with a $\mathcal{T}$-lemma $\neg\eta$ ($n_{cl} = \neg\eta$), set $n_I = \mathcal{T}\text{-INTERPOLANT}(\eta \setminus \mathsf{B}, \eta \downarrow \mathsf{B})$.
(3) For every non-leaf node $n \in P$, set $n_I = n_I^L \vee n_I^R$ if $n_p \notin \mathsf{B}$, and set $n_I = n_I^L \wedge n_I^R$ if $n_p \in \mathsf{B}$.
(4) Let $r \in P$ be the root node of P associated with the empty clause $r_{cl} = \emptyset$. r_I is an interpolant of A and B.

Note that the interpolation procedure differs from pure Boolean interpolation [4] only in the handling of $\mathcal{T}$-lemmata. $\mathcal{T}\text{-INTERPOLANT}(\cdot, \cdot)$ produces an interpolant for an unsatisfiable pair of conjunctions of $\mathcal{T}$-literals. (In [7], the authors list interpolation algorithms for several theories.)

In this paper we consider the *theory of linear arithmetic over rationals* $\mathcal{LA}(\mathbb{Q})$. We write $Ax \leq a$ for a conjunction of m linear inequations over rational variables $(x_1, \ldots, x_n)^T = x$ with $A \in \mathbb{Q}^{m \times n}$, $a \in \mathbb{Q}^m$.[2] Every row vector in the $m \times n$-matrix A describes the coefficients of the corresponding linear inequation.

There exist several methods to construct an $\mathcal{LA}(\mathbb{Q})$-interpolant from conflicts in an $\mathcal{LA}(\mathbb{Q})$-proof as described in [6], [7], [30]. Here we review the approach from [30], since our method is based on this approach.

We assume an $\mathcal{LA}(\mathbb{Q})$-conflict η which is produced during the proof of unsatisfiability of two formulas A and B. From η we may extract a conjunction $\eta \setminus \mathsf{B}$ of linear inequations only occurring in formula A and a conjunction $\eta \downarrow \mathsf{B}$ of linear inequations occurring in formula B. $\eta \setminus \mathsf{B}$ and $\eta \downarrow \mathsf{B}$ are represented by the inequation systems $Ax \leq a$ and $Bx \leq b$, respectively ($A \in \mathbb{Q}^{m_\mathsf{A} \times n}$, $a \in \mathbb{Q}^{m_\mathsf{A}}$, $B \in \mathbb{Q}^{m_\mathsf{B} \times n}$, $b \in \mathbb{Q}^{m_\mathsf{B}}$). Since η is an $\mathcal{LA}(\mathbb{Q})$-conflict, the conjunction of $Ax \leq a$ and $Bx \leq b$ has no solution. Then, according to Farkas' lemma, there exists a linear inequation $i^T x \leq \delta$ ($i \in \mathbb{Q}^n$, $\delta \in \mathbb{Q}$) which is an $\mathcal{LA}(\mathbb{Q})$-interpolant for $Ax \leq a$ and $Bx \leq b$. $i^T x \leq \delta$ can be computed by linear programming from the following (in)equations with additional variables $\lambda \in \mathbb{Q}^{m_\mathsf{A}}$, $\mu \in \mathbb{Q}^{m_\mathsf{B}}$:
(1) $\lambda^T A + \mu^T B = \mathbf{0}^T$, (2) $\lambda^T a + \mu^T b \leq -1$,
(3) $\lambda^T A = i^T$, (4) $\lambda^T a = \delta$, (5) $\lambda \geq \mathbf{0}$, $\mu \geq \mathbf{0}$.

The coefficients λ and μ define a positive linear combination of the inequations in $Ax \leq a$ and $Bx \leq b$ leading to a contradiction $0 \leq \lambda^T a + \mu^T b$ with $\lambda^T a + \mu^T b \leq -1$ (see (1) and (2)). The interpolant $i^T x \leq \delta$ just "sums up" the "$Ax \leq a$"-part of the linear combination leading to the contradiction (see (3) and (4)), thus $i^T x \leq \delta$ is implied by $Ax \leq a$. $i^T x \leq \delta$ is clearly inconsistent with $Bx \leq b$, since it derives the same contradiction as before. Altogether $i^T x \leq \delta$ is an interpolant of $Ax \leq a$ and $Bx \leq b$.

Example 2 (cont.): Fig. 2 shows a Craig interpolant resulting from the proof in Fig. 1, when partitioning S into (A, B) with $\mathsf{A} = (l_1 \vee l_3) \wedge (l_1 \vee l_4) \wedge (l_2 \vee l_3) \wedge (l_2 \vee l_4)$ and $\mathsf{B} = l_5 \wedge l_6$. The $\mathcal{LA}(\mathbb{Q})$-interpolant for the $\mathcal{LA}(\mathbb{Q})$-conflict η_1 is a positive linear combination of η_1's A-literals (i.e. l_1 and l_2), which is conflicting with a positive linear-combination of the remaining literals (i.e. l_5), e. g. $1 \cdot (-x_2 \leq 0) + 2 \cdot (x_1 \leq 1) \equiv (2x_1 - x_2 \leq 2)$ and $1 \cdot (-2x_1 + x_2 \leq -6)$ lead to the conflict $0 \leq -4$. Similarly, the interpolant $x_1 - 2x_2 \leq -4$ is derived from the $\mathcal{LA}(\mathbb{Q})$-conflict η_2. Propagating constants, the final interpolant of A and B becomes $(2x_1 - x_2 \leq 2) \vee (x_1 - 2x_2 \leq -4)$. Fig. 3 gives a geometric illustration of the example. A is depicted in blue, B in orange, the interpolant is represented by the green or blue areas. η_1 says that $l_1 \wedge l_2$ (the blue area with vertical lines) does not intersect with l_5 (leading to interpolant $l_7 = (2x_1 - x_2 \leq 2)$) and η_2 says that $l_3 \wedge l_4$ (the blue area with horizontal lines) does not intersect with l_6 (leading to interpolant $l_8 = (x_1 - 2x_2 \leq -4)$).

III. Computing Simple Interpolants

A. Basic idea

Now we present a method computing *simpler* interpolants than the standard methods mentioned in Sect. II. The basic idea is as follows: It is based upon the observation that in previous interpolation schemes the inconsistency proofs and thus the interpolants derived from different $\mathcal{T}$-conflicts are *uncorrelated*. In most cases different $\mathcal{T}$-conflicts lead to different $\mathcal{LA}(\mathbb{Q})$-interpolants contributing to the final interpolant, thus, complicated proofs with many $\mathcal{T}$-conflicts tend to lead to complicated Craig interpolants depending on many linear constraints.

Example 3 (cont.): In Ex. 2 we have two different $\mathcal{T}$-conflicts leading to two different interpolants (see green lines in Fig. 3). However, it is easy to see from Fig. 4 that there is a single inequation $l_9 = (x_1 - x_2 \leq 1)$ which can be used as an interpolant for A and B (A implies l_9 and l_9 does not intersect with B).

Our idea is to *share* $\mathcal{LA}(\mathbb{Q})$-interpolants between different $\mathcal{T}$-conflicts. In order to come up with an interpolation scheme using as many shared interpolants as possible, we first introduce a check whether a fixed set of $\mathcal{T}$-conflicts can be proved by a shared proof, leading to a single shared $\mathcal{LA}(\mathbb{Q})$-interpolant for that set of $\mathcal{T}$-conflicts.

[2] For simplicity we confine ourselves to non-strict inequations. A generalization to mixed strict and non-strict inequations is straightforward.

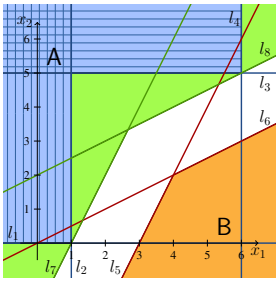

Figure 3. Two $\mathcal{LA}(\mathbb{Q})$-interpolants l_7 and l_8 for the interpolation between A and B.

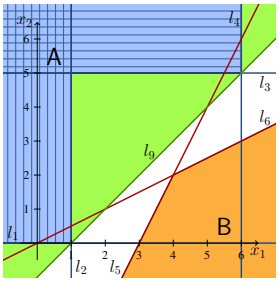

Figure 4. A single $\mathcal{LA}(\mathbb{Q})$-interpolant l_9 replacing l_7 and l_8.

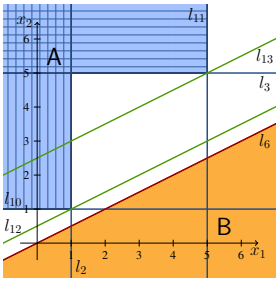

Figure 5. Relaxing constraints to enable l_{12} as shared interpolant.

We assume a fixed set $\{\eta_1, \ldots, \eta_r\}$ of $\mathcal{T}$-conflicts. Each $\mathcal{T}$-conflict η_j defines two systems of inequations: $A_j x \leq a_j$ for the A-part and $B_j x \leq b_j$ for the B-part. Extending [30] we ask whether there is a single inequation $i^T x \leq \delta$ and coefficients λ_j, μ_j with

(1_j) $\lambda_j^T A_j + \mu_j^T B_j = \mathbf{0}^T$, (2_j) $\lambda_j^T a_j + \mu_j^T b_j \leq -1$,
(3_j) $\lambda_j^T A_j = i^T$, (4_j) $\lambda_j^T a_j = \delta$, (5_j) $\lambda_j \geq \mathbf{0}$, $\mu_j \geq \mathbf{0}$

for all $j \in \{1, \ldots, r\}$. Note that the coefficients λ_j and μ_j for the different $\mathcal{T}$-conflicts may be different, but the interpolant $i^T x \leq \delta$ is required to be identical for all $\mathcal{T}$-conflicts. Again, the problem formulation consisting of all constraints (1_j)–(5_j) can be solved by linear programming in polynomial time.

Unfortunately, first results showed that the potential to find shared interpolant was not as high as expected using this basic idea. By a further analysis of the problem we observed that more degrees of freedom are needed to enable a larger number of shared interpolants.

B. Relaxing constraints

Consider Fig. 5 for motivating our first measure to increase the degrees of freedom for interpolant generation. Fig. 5 shows a slightly modified example compared to Figs. 3 and 4 with $\mathsf{A} = (l_{10} \wedge l_2) \vee (l_3 \wedge l_{11})$ and $\mathsf{B} = l_6$. Again we have two $\mathcal{T}$-conflicts: η_3 which says that $l_{10} \wedge l_2 \wedge l_6$ is infeasible and η_4 which says that $l_3 \wedge l_{11} \wedge l_6$ is infeasible. We can show that the interpolation generation according to [30] only computes interpolants which *touch* the A-part of the $\mathcal{T}$-conflict (as long as the corresponding theory conflict is minimized, and both A-part and B-part are not empty). Thus the only possible interpolants for η_3 and η_4 according to (1)–(5) are l_{12} and l_{13}, respectively. I.e. it is not possible to compute a *shared* interpolant for this example according to equations (1_j)–(5_j). On the other hand it is easy to see that l_{12} may also be used as an interpolant for η_4, if we do not require interpolants to touch the A-part (which is $l_3 \wedge l_{11}$ in the example). We achieve that goal simply by relaxing constraint (4_j) to $(4'_j)$ $\lambda_j a_j \leq \delta$ and by modifying (2_j) to $(2'_j)$ $\delta + \mu_j^T b_j \leq -1$ (all other constraints (i'_j) remain the same as (i_j)). An inequation $i^T x \leq \delta$ computed according to $(1'_j)$–$(5'_j)$ is still implied by $A_j x \leq a_j$ (since $i^T x \leq \lambda_j a_j$ is implied and $\lambda_j a_j \leq \delta$) and it contradicts $B_j x \leq b_j$, since $0 \leq \lambda_j^T a_j + \mu_j^T b_j \leq \delta + \mu_j^T b_j$ is conflicting with $\delta + \mu_j^T b_j \leq -1$.

C. Extending $\mathcal{T}$-conflicts

There is a second restriction to the degrees of freedom for shared interpolants which follows from the computation of *minimized* $\mathcal{T}$-conflicts in SMT–solvers (see Sect. II). (Note that minimized $\mathcal{T}$-conflicts are used with success in modern SMT–solvers in order to prune the search space as much as possible. Unfortunately, minimization of $\mathcal{T}$-conflicts impedes the search for shared interpolants.) We can prove the following lemma (the proof is omitted due to lack of space):

Lemma 1: If a $\mathcal{LA}(\mathbb{Q})$-conflict η is minimized, and both $\eta \setminus \mathsf{B}$ and $\eta \downarrow \mathsf{B}$ are not empty, then the direction of vector i of an $\mathcal{LA}(\mathbb{Q})$-interpolant $i^T x \leq \delta$ for $\eta \setminus \mathsf{B}$ and $\eta \downarrow \mathsf{B}$ is fixed.

Example 4 (cont.): Again consider Fig. 3. Since the $\mathcal{LA}(\mathbb{Q})$-conflict $\eta_1 = l_1 \wedge l_2 \wedge l_5$ is minimized, the direction vector of the interpolant l_7 is fixed. The same holds for $\mathcal{LA}(\mathbb{Q})$-conflict $\eta_2 = l_3 \wedge l_4 \wedge l_6$ and the direction vector of l_8. Thus, there is no shared interpolant for η_1 and η_2.

Fortunately, $\mathcal{T}$-conflicts which are extended by additional inequations remain $\mathcal{T}$-conflicts. (If the conjunction of some inequations is infeasible, then any extension of the conjunction is infeasible as well.) Therefore we may extend η_1 to $\eta'_1 = l_1 \wedge l_2 \wedge l_5 \wedge l_6$ and η_2 to $\eta'_2 = l_3 \wedge l_4 \wedge l_5 \wedge l_6$. It is easy to see that the linear inequation $l_9 = (x_1 - x_2 \leq 1)$ from Fig. 4 is a solution of $(1'_j)$–$(5'_j)$ applied to η'_1 and η'_2 (with coefficients $\lambda_{1,1} = \lambda_{1,2} = 1$, $\mu_{1,1} = \mu_{1,2} = \frac{1}{3}$, $\lambda_{2,1} = \lambda_{2,2} = 1$, $\mu_{2,1} = \mu_{2,2} = \frac{1}{3}$). This means that we really obtain the shared interpolant l_9 from Fig. 4 by $(1'_j)$–$(5'_j)$, if we extend the $\mathcal{T}$-conflicts appropriately.

We learn from Ex. 4 that an appropriate extension of $\mathcal{T}$-conflicts increases the degrees of freedom in the computation of interpolants, leading to new shared interpolants. Clearly, in the general case an extension of $\mathcal{T}$-conflicts η_j (and thus of $\mathcal{T}$-lemmata $\neg\eta_j$) may destroy proofs of $\mathcal{T}$-unsatisfiability. In the following we derive conditions when interpolants derived from proofs with extended $\mathcal{T}$-lemmata are still correct.

1) Implied literals:

Definition 3: Let A and B be two formulas, and let l be a literal. l is an *implied literal for* A (*implied literal for* B), if $\mathsf{A} \models_{\mathcal{T}} l$ and l does not occur in B (if $\mathsf{B} \models_{\mathcal{T}} l$).

Lemma 2: Let P be a proof of $\mathcal{T}$-unsatisfiability of $\mathsf{A} \wedge \mathsf{B}$, let $\neg\eta$ be a $\mathcal{T}$-lemma in P not containing literal $\neg l$, and let l be implied for A (for B). Then Craig interpolation according to [6] (see Sect. II) applied to P with $\neg\eta$ replaced by $\neg\eta \vee \neg l$ computes a Craig interpolant for A and B.

Proof: Here we prove only the case that l is an implied literal for A. In P we replace the node labeled by $\neg\eta$ with a new non-leaf node n with parents n^L and n^R. n is labeled by $n_{cl} = \neg\eta$, too. Its pivot variable is $n_p = l$. n_L is a leaf labeled by clause (l) and n_R is a leaf labeled by the $\mathcal{T}$-lemma $\neg\eta \vee \neg l$. It is easy to see that the resulting proof P' is a proof of $\mathcal{T}$-unsatisfiability of $(\mathsf{A} \wedge l) \wedge \mathsf{B}$. Therefore the Craig interpolant I computed from P' is an interpolant for $(\mathsf{A} \wedge l)$ and B. Since l is implied by A, $(\mathsf{A} \wedge l)$ and A are equivalent modulo theory, i.e., I is also an interpolant for A and B. Since according to the rules in [6] the partial interpolant at node n is $n_I = \bot \vee \mathcal{T}\text{-INTERPOLANT}((\eta \wedge l) \setminus \mathsf{B}, (\eta \wedge l) \downarrow \mathsf{B}) \equiv \mathcal{T}\text{-INTERPOLANT}((\eta \wedge l) \setminus \mathsf{B}, (\eta \wedge l) \downarrow \mathsf{B})$, I coincides with the interpolant which results by interpolation in P after replacing $\neg\eta$ by $\neg\eta \vee \neg l$. ■

We can conclude from Lemma 2 that we are free to arbitrarily add negations of implied literals for A or B to $\mathcal{T}$-lemmata without losing the property that the resulting formula according to [6] is an interpolant of A and B.

Example 5 (cont.): Again consider Fig. 3. l_1 and l_4 are clearly implied literals for A, l_5 and l_6 are implied literals for B. Therefore we can extend $\mathcal{T}$-conflict η_1 to $\eta_1'' = l_1 \wedge l_2 \wedge l_4 \wedge l_5 \wedge l_6$ and η_2 to $\eta_2'' = l_1 \wedge l_3 \wedge l_4 \wedge l_5 \wedge l_6$. $(1_j')$–$(5_j')$ applied to η_1'' and η_2'' and interpolation according to [6] leads to l_9 as an interpolant of A and B (similarly to Ex. 4, see Fig. 4).

2) Lemma Localization: In [27] the authors introduced another method called Lemma Localization that extends $\mathcal{T}$-conflicts with additional $\mathcal{T}$-literals. The additional $\mathcal{T}$-literals are derived by the so-called *pushup operation* which detects redundant $\mathcal{T}$-literals in the proof structure. A $\mathcal{T}$-literal l is called redundant in a proof node, if the clauses of all successor nodes contain the literal l and the current node does not use it as the pivot variable for the resolution. Such a redundant $\mathcal{T}$-literal can then be added to the current node without losing correctness of the proof. A redundant literal l may eventually be pushed into a leaf of the proof which represents a $\mathcal{T}$-lemma $\neg\eta$. The corresponding $\mathcal{T}$-*conflict* η is extended by $\neg l$. The method in [27] uses the pushup-algorithm in order to replace potentially complex $\mathcal{T}$-interpolants by constants: If $(\eta \wedge \neg l) \setminus \mathsf{B}$ is still a $\mathcal{T}$-conflict, then $\bot$ is a valid $\mathcal{T}$-interpolant, and if $(\eta \wedge \neg l) \downarrow \mathsf{B}$ is still a $\mathcal{T}$-conflict, then $\top$ is a valid $\mathcal{T}$-interpolant. Of course, extending theory conflicts by additional literals increases the chance of obtaining constant $\mathcal{T}$-interpolants. In our work we make use of the pushup operation to *increase the degrees of freedom for computing shared $\mathcal{T}$-interpolants*. (Nevertheless, before computing shared interpolants we look for constant $\mathcal{T}$-interpolants as in [27], since this method contributes to a minimization of non-trivial $\mathcal{T}$-interpolants as well and the sizes of overall Craig-interpolants may decrease significantly due to the propagation of constants.)

D. Overall algorithm

Our overall algorithm starts with a $\mathcal{T}$-unsatisfiable set of clauses S and a disjoint partition (A, B) of S, and computes a proof P for the $\mathcal{T}$-unsatisfiability of S. P contains r $\mathcal{T}$-lemmata $\neg\eta_1, \ldots, \neg\eta_r$. The system of (in)equations $(1_j')$–$(5_j')$ from Sect. III-B with $j \in \{j_1, \ldots, j_k\}$ provides us with a check whether there is a shared interpolant $i^T x \leq \delta$ for the subset $\{\eta_{j_1}, \ldots, \eta_{j_k}\}$ of $\mathcal{T}$-conflicts. We call this check `SharedInterpol`$(\{\eta_{j_1}, \ldots, \eta_{j_k}\})$. Our goal is to find an interpolant for A and B with a minimal number of different $\mathcal{T}$-interpolants. At first, we use `SharedInterpol` to precompute an (undirected) compatibility graph $G_{cg} = (V_{cg}, E_{cg})$ with $V_{cg} = \{\eta_1, \ldots, \eta_r\}$ and $\{\eta_i, \eta_j\} \in E_{cg}$ iff there is a shared interpolant of η_i and η_j.

1) Iterative greedy algorithm: Our first algorithm is a simple iterative greedy algorithm based on `SharedInterpol`. We iteratively compute sets SI_i of $\mathcal{T}$-conflicts which have a shared interpolant. We start with $SI_1 = \{\eta_s\}$ for some $\mathcal{T}$-conflict η_s. To extend a set SI_i we select a new $\mathcal{T}$-conflict $\eta_c \notin \cup_{j=1}^{i} SI_j$ with $\{\eta_c, \eta_j\} \in E_{cg}$ for all $\eta_j \in SI_i$. Then we check whether `SharedInterpol`$(SI_i \cup \{\eta_c\})$ returns true or false. If the result is true, we set $SI_i := SI_i \cup \{\eta_c\}$, otherwise we select a new $\mathcal{T}$-conflict as above. If there is no appropriate new $\mathcal{T}$-conflict, then we start a new set SI_{i+1}. The algorithm stops when all $\mathcal{T}$-conflicts are inserted into a set SI_j.

Of course, the quality of the result depends on the selection of $\mathcal{T}$-conflicts to start new sets SI_i and on the decision which candidate $\mathcal{T}$-conflicts to select if there are several candidates. So the cardinality of sets SI_i and their total number (i.e. the number of computed $\mathcal{LA}(\mathbb{Q})$-interpolants) is not necessarily minimal.

2) Maximum subsets of shared interpolants: We present a second algorithm to improve on the order dependency of the iterative greedy algorithm. The second algorithm is based on a procedure `MaxSubsetSI`$(\{\eta_{j_1}, \ldots, \eta_{j_k}\})$ which computes a *maximum* subset of $\mathcal{T}$-conflicts in $\{\eta_{j_1}, \ldots, \eta_{j_k}\}$ which has a shared interpolant.

First of all we extend (in)equations $(1_j')$–$(5_j')$ from Sect. III-B with activation variables $\alpha_1, \ldots, \alpha_r \in \{0, 1\}$ for each $\mathcal{T}$-conflict and obtain an SMT-formula MS which is a conjunction of r subformulas of the form

$$\Big(\alpha_j \Rightarrow \Big[(\lambda_j^T A_j + \mu_j^T B_j = \mathbf{0}^T) \wedge (\delta + \mu_j^T b_j \leq -1) \wedge (\lambda_j^T A_j = i^T) \wedge (\lambda_j^T a_j \leq \delta)\Big]\Big) \wedge (\lambda_j \geq \mathbf{0}) \wedge (\mu_j \geq \mathbf{0}).$$

A solution to MS with $\alpha_{i_1} = \ldots = \alpha_{i_l} = 1$ provides a shared interpolant for $\{\eta_{i_1}, \ldots, \eta_{i_l}\}$. Thus our goal is to find a solution to MS which maximizes $\sum_{j=1}^{r} \alpha_j$.

To increase the efficiency of our search we partition the graph G_{cg} into connected components and restrict our search for maximum subsets with shared interpolants to connected components. Let $\{\eta_{j_1}, \ldots, \eta_{j_k}\}$ be the set of $\mathcal{T}$-conflicts in the current connected component CC. We set $\alpha_j = 0$ for all $j \notin \{j_1, \ldots, j_k\}$ to "turn off the constraints for $\mathcal{T}$-conflicts outside the current connected component". For maximization we introduce the Boolean cardinality constraint $\sum_{i=1}^{k} \alpha_{j_i} \geq b$ into MS and perform a binary search for a maximum b. There are several approaches for translating Boolean cardinality constraints into SAT or SMT (see [36], [37], e.g.). In our implementation we use a sorter network [38] with inputs $\alpha_1, \ldots, \alpha_r$ and constrain the bth output of the sorter network to 1. If CC contains more than one $\mathcal{T}$-conflict, the binary search starts with $b = 2$ as a lower bound; an upper bound for b results from an upper bound on the size of the largest clique in CC. After a maximum subset of $\mathcal{T}$-conflicts in the current connected component has been found, the corresponding nodes are removed from G_{cg} and we continue with searching for the next maximum subset.

IV. Experimental Results

We implemented the approach from Sect. III and applied it to a set of benchmarks representing intermediate state sets produced by a hybrid model checker [13]. As in [27] the formula "A" for interpolation is given by the original state set and the formula "¬B" is a "bloated version" of A where all inequations are pushed outwards by a positive distance ϵ. The formulas representing A and B contain up to 5 rational variables, up to 1,380 inequations, up to 18,914 Boolean variables, and up to 56,721 clauses.

The $\mathcal{LA}(\mathbb{Q})$-proofs of unsatisfiability of A ∧ B were generated with MathSat 5 [33]. We compare the results of the two algorithms from Sect. III-D1 and III-D2 to the original interpolation technique implemented in MathSat 5 and to the Lemma Localization method from [27]. All experiments were conducted on one core of an Intel Xeon machine with 3.0 GHz and a memory limit of 4GB RAM.

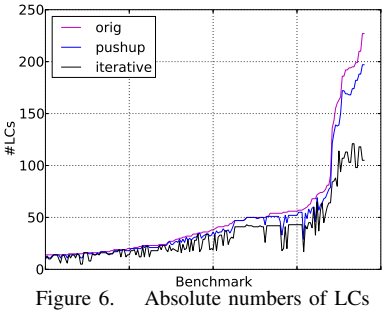

Figure 6. Absolute numbers of LCs

Figs. 6 and 7 show a comparison of the original interpolation ('orig', magenta line) with [27] ('pushup', blue line) and the iterative greedy method from Sect. III-D1 ('iterative', black line). The x-axis represents the different benchmarks, ordered by the number of linear constraints in the original interpolant. In Fig. 6 the y-axis represents the absolute numbers of linear constraints in the corresponding interpolants. (Here benchmarks with less than 15 linear constraints in the original interpolant are omitted for facility of inspection.) In Fig. 7 the y-axis represents the *relative* numbers of linear

constraints (the original interpolants are all normalized to 100%, so the values for 'original' are 1.0 by definition); again the benchmarks are ordered by the number of linear constraints in the original interpolant. Whereas 'pushup' leads to an overall reduction of the number of linear constraints by 9.9% compared to 'orig' (with a maximum reduction of 60.0%), 'iterative' leads to an overall reduction by 34.6% compared to 'orig' (with a maximum reduction of 70.7%).

The CPU times for computing the original interpolants are all below 3.7 CPU seconds. Due to the effort of minimizing the numbers of linear constraints the CPU times for 'iterative' increase by an average factor of 27.5; all CPU times for 'iterative' remain below 7 min, 22 s.

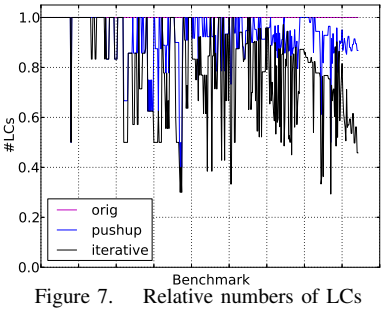

Figure 7. Relative numbers of LCs

Again for facility of inspection, we omitted the results for the algorithm computing maximum subsets of shared interpolants from Sect. III-D2 ('max_subset') in Fig. 6, since they do not differ much from the results of the iterative algorithm. Compared to algorithm 'iterative', the maximum reduction of linear constraints in one interpolant obtained by 'max_subset' was 3 and the overall improvement was only by 0.07%. Since the CPU times for 'max_subset' again increase by a factor of 6 compared to 'iterative', we can conclude that the increased effort made by 'max_subset' obviously does not pay off for this set of benchmarks.

In summary, the experimental results demonstrate a considerable potential of our minimization method computing shared $\mathcal{LA}(\mathbb{Q})$-interpolants. The results definitely suggest to use our iterative method in applications which profit from simple interpolants with a minimized number of linear constraints.

V. Conclusion and Future Work

In this paper we demonstrated that interpolants based on proofs of unsatisfiablity may be *simplified* to a great extent by a method computing shared interpolants. The key to successful simplification is a step which preprocesses the proofs and increases the degrees of freedom in the selection of interpolants for theory conflicts. Our current implementation is restricted to linear arithmetic. In the future we will investigate generalizations to other theories. Certain generalizations, like a generalization to the combination of linear arithmetic and uninterpreted functions, are straightforward, since in modular approaches like [30], [39] we only need to exchange the interpolation construction method for linear arithmetic by our method.

References

[1] W. Craig, "Three Uses of the Herbrand-Gentzen Theorem in Relating Model Theory and Proof Theory," *Journal of Symbolic Logic*, vol. 22, no. 3, pp. pp. 269–285, 1957.

[2] L. Zhang and S. Malik, "Validating SAT solvers using an independent resolution-based checker: Practical implementations and other applications," in *DATE*, 2003, pp. 880–885.

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

[4] K. L. McMillan, "Interpolation and SAT-Based Model Checking," in *Proc. of CAV*, 2003, vol. 2742, pp. 1–13.

[5] L. de Moura, H. Ruess, and M. Sorea, "Lazy Theorem Proving for Bounded Model Checking over Infinite Domains," in *Proc. of CADE*, 2002, pp. 1–4.

[6] K. L. McMillan, "An Interpolating Theorem Prover," *Theoretical Computer Science*, vol. 345, no. 1, pp. 101 – 121, 2005.

[7] A. Cimatti, A. Griggio, and R. Sebastiani, "Efficient Generation of Craig Interpolants in Satisfiability Modulo Theories," *ACM Trans. Comput. Logic*, vol. 12, no. 1, pp. 7:1–7:54, Nov. 2010.

[8] T. A. Henzinger, R. Jhala, R. Majumdar, and K. L. McMillan, "Abstractions from Proofs," in *Proc. of POPL*, 2004, pp. 232–244.

[9] K. L. McMillan, "Lazy abstraction with interpolants," in *Proc. of CAV*, 2006, pp. 123–136.

[10] A. Cimatti, A. Griggio, A. Micheli, I. Narasamdya, and M. Roveri, "Kratos - A Software Model Checker for SystemC," in *Proc. of CAV*, 2011, pp. 310–316.

[11] D. Kroening and G. Weissenbacher, "Interpolation-Based Software Verification with Wolverine," in *Proc. of CAV*, 2011, pp. 573–578.

[12] C. Scholl, S. Disch, F. Pigorsch, and S. Kupferschmid, "Computing Optimized Representations for Non-convex Polyhedra by Detection and Removal of Redundant Linear Constraints," in *Proc. of TACAS*, 2009, pp. 383–397.

[13] W. Damm, H. Dierks, S. Disch, W. Hagemann, F. Pigorsch, C. Scholl, U. Waldmann, and B. Wirtz, "Exact and Fully Symbolic Verification of Linear Hybrid Automata with Large Discrete State Spaces," *Science of Computer Programming*, vol. 77, no. 10-11, pp. 1122–1150, 2012.

[14] G. Audemard, A. Cimatti, A. Kornilowicz, and R. Sebastiani, "Bounded Model Checking for Timed Systems," in *FORTE*, 2002, pp. 243–259.

[15] G. Audemard, M. Bozzano, A. Cimatti, and R. Sebastiani, "Verifying Industrial Hybrid Systems with MathSAT," *Electr. Notes Theor. Comput. Sci.*, vol. 119, no. 2, pp. 17–32, 2005.

[16] M. Fränzle and C. Herde, "HySAT: An Efficient Proof Engine for Bounded Model Checking of Hybrid Systems," *Formal Methods in System Design*, vol. 30, no. 3, pp. 179–198, 2007.

[17] R.-R. Lee, J.-H. R. Jiang, and W.-L. Hung, "Bi-decomposing large boolean functions via interpolation and satisfiability solving," in *DAC*, 2008, pp. 636–641.

[18] H.-P. Lin, J.-H. R. Jiang, and R.-R. Lee, "To SAT or not to SAT: Ashenhurst Decomposition in a Large Scale," in *ICCAD*, 2008, pp. 32–37.

[19] C. Sinz, "Compressing Propositional Proofs by Common Subproof Extraction," in *Proc. of EUROCAST*, 2007, pp. 547–555.

[20] S. F. Rollini, R. Bruttomesso, and N. Sharygina, "An Efficient and Flexible Approach to Resolution Proof Reduction," in *Proc. of HVC*, 2011, pp. 182–196.

[21] J. D. Backes and M. D. Riedel, "Reduction of Interpolants for Logic Synthesis," in *Proc. of ICCAD*, 2010, pp. 602–609.

[22] O. Bar-Ilan, O. Fuhrmann, S. Hoory, O. Shacham, and O. Strichman, "Reducing the size of resolution proofs in linear time," *STTT*, vol. 13, no. 3, pp. 263–272, 2011.

[23] A. Gupta, "Improved single pass algorithms for resolution proof reduction," in *ATVA*, ser. LNCS, vol. 7561, 2012, pp. 107–121.

[24] R. Jhala and K. L. McMillan, "Interpolant-Based Transition Relation Approximation," *Logical Methods in Computer Science*, vol. 3, no. 4, 2007.

[25] V. D'Silva, M. Purandare, G. Weissenbacher, and D. Kroening, "Interpolant Strength," in *Proc. of VMCAI*, 2010, pp. 129–145.

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

[27] F. Pigorsch and C. Scholl, "Lemma localization: A practical method for downsizing SMT-interpolants," in *Proc. of DATE*, 2013, pp. 1405–1410.

[28] K. Hoder, L. Kovács, and A. Voronkov, "Playing in the grey area of proofs," in *POPL*. ACM, 2012, pp. 259–272.

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

[30] A. Rybalchenko and V. Sofronie-Stokkermans, "Constraint Solving for Interpolation," in *Proc. of VMCAI*, 2007, pp. 346–362.

[31] G. Frehse, "PHAVer: Algorithmic verification of hybrid systems past HyTech," *STTT*, vol. 10, no. 3, pp. 263–279, 2008.

[32] G. Frehse, C. L. Guernic, A. Donzé, S. Cotton, R. Ray, O. Lebeltel, R. Ripado, A. Girard, T. Dang, and O. Maler, "SpaceEx: Scalable verification of hybrid systems," in *CAV*, ser. LNCS, vol. 6806. Springer, 2011, pp. 379–395.

[33] A. Cimatti, A. Griggio, B. Schaafsma, and R. Sebastiani, "The MathSAT5 SMT Solver," in *Proceedings of TACAS*, ser. LNCS, vol. 7795. Springer, 2013.

[34] M. Davis, G. Logemann, and D. Loveland, "A machine program for theorem-proving," *Commun. ACM*, vol. 5, no. 7, pp. 394–397, Jul. 1962.

[35] R. Sebastiani, "Lazy satisability modulo theories," *JSAT*, vol. 3, no. 3-4, pp. 141–224, 2007.

[36] C. Sinz, "Towards an optimal CNF encoding of boolean cardinality constraints," in *Principles and Practice of Constraint Programming*, ser. LNCS, vol. 3709. Springer, 2005, pp. 827–831.

[37] N. Eén and N. Sörensson, "Translating pseudo-boolean constraints into SAT," *JSAT*, vol. 2, no. 1-4, pp. 1–26, 2006.

[38] K. E. Batcher, "Sorting networks and their applications," in *Proceedings of the AFIPS Spring Joint Computer Conference*, 1968, pp. 307–314.

[39] D. Beyer, D. Zufferey, and R. Majumdar, "CSIsat: Interpolation for LA+EUF," in *CAV*, ser. LNCS, vol. 5123. Springer, 2008, pp. 304–308.